Modelling and Optimization

edited by
Jan Sikora
Waldemar Wójcik

Politechnika Lubelska
Lublin 2011

Reviewers:
prof. dr hab. inż. Zbigniew Łukasik
prof. dr hab. inż. Ryszard Romaniuk

Publication approved by Rector of Lublin University of Technology

ISBN: 978-83-62596-34-8

Publisher: Lublin University of Technology
ul. Nadbystrzycka 38D, 20-618 Lublin, Poland
Realization: Lublin University of Technology Library
ul. Nadbystrzycka 36A, 20-618 Lublin, Poland
tel. (81) 538-46-59, email: wydawca@pollub.pl
www.biblioteka.pollub.pl
Printed by : ESUS Tomasz Przybylak, Poznań, Poland
www.esus.pl

The digital version is available at the Digital Library of Lublin University of Technology: www.bc.pollub.pl
Impression: 100 copies

Contents

Part I

Parallel computing

Chapter 1

Control of fractional-order dynamic systems under uncertainty

A. Chikrii, I. Matychyn, K. Gromaszek, A. Smolarz

1.1 Summary

In this chapter we explore the linear non-homogeneous fractional-order differential systems with classical Riemann-Liouville [9], the Caputo regularized [33, 16], and Miller-Ross [2] sequential derivatives. Solutions to these systems are presented in the form of Cauchy formula analogs by the use of the Mittag-Leffler generalized matrix functions [12].

We also treat sequential derivatives of special form [16]. Their relation to the Riemann-Liouville and Caputo fractional derivatives and to each other is established.

Differential games of approach for the systems with the fractional derivatives of Riemann-Liouville, Caputo, as well as with the sequential derivatives are studied. The Method of Resolving Functions [34, 35, 15, 14] allows to derive

sufficient conditions for solvability of the mentioned game problems. These conditions are based on the modified Pontryagin's condition [34]. The results are illustrated on a model example where a dynamic system of order π pursues another system of order e. The case of plain matrix is also examined, where asymptotic representation of the scalar Mittag-Leffler generalized function are employed.

The problem of group pursuit, which illustrates on an example the situation of encirclement by Pshenichnyi [6], is studied separately.

1.2 Introduction

Operations of fractional differentiation and integration go back a long way. It seems likely that most straightforward way to their definitions is associated with the Abel integral equation and the Cauchy formula for multiple integration of functions. These issues are widely covered in the monograph [9], also one can use the book [2]. Various kinds of fractional derivatives were generated by the needs of practice. Detailed historical review of this subject matter is contained in [9]. The fractional derivatives under study, namely, Riemann-Liouville, Caputo, Miller-Ross, and Hilfer also have their own specifics. The Riemann-Liouville fractional derivative has singularity at the origin. Therefore, the trajectories of corresponding differential system start at the infinity, which seems not always justified from the physical point of view. That is why Caputo regularized derivatives appeared, in which this defect was eliminated. This means that the trajectories of corresponding systems do not arrive at the infinity at any finite moment of time. However, both the Riemann-Liouville and the Caputo fractional operators do not possess neither semigroup nor commutative properties, which are inherent to the derivatives of integer order. This gap is filled by the Miller-Ross fractional derivatives, which, in particular, make it feasible to lower the order of a system of differential equations by increasing their number. Hilfer fractional derivatives provide a framework that encompasses both the Riemann-Liouville and Caputo derivatives as particular cases.

Representations of solutions of the above mentioned fractional order linear non- homogeneous systems play an important role in the mathematical control theory and the theory of dynamic games [35, 15, 14]. Some formulas can be found in [2, 7]. In [12], by introducing of the Mittag-Leffler generalized

matrix functions, an analogue of the Cauchy formula was derived in the case of Riemann-Liouville and Caputo fractional derivatives for order $0 < \alpha < 1$ (without the help of the Laplace transform). Corresponding formulas for derivatives of arbitrary order can be found in [16].

It seems likely that research into the game problems for the fractional-order systems go back to the paper [12]. The basic method in these studies is the method of resolving functions [34, 35, 15, 12, 13, 14, 16, 11]. This method is sometimes called the method of the inverse Minkowski functionals. Originally, the method was developed to solve the group pursuit problems (see [6, 34, 25]). In the paper [25] it is referred to as the method of the guaranteed position non-deterioration. The method of resolving functions is based on the Pontryagin condition or its modifications. This method is sufficiently universal: it allows exploring the conflict-controlled processes for objects of different inertia as well as of oscillatory and rotary dynamics, the game problems under state constraints, integral constraints of control, and game problems with impulse controls [34, 11]. The method of resolving functions fully substantiates the classical rule of parallel pursuit [34]. The gist of the method consists in constructing of special measurable set-valued mappings with closed images and their support functions. These functions integrally characterize the course of the conflict-controlled process [34, 13]. They are referred to as the resolving functions. Instead of the Filippov-Castaing lemma, $\Lambda \times \mathcal{B}$-measurability [13, 17] and resulting superpositional measurability of certain set-valued mappings and their selections is employed to substantiate a measurable choice of the pursuer's control.

1.3 Riemann-Liouville and Caputo Fractional Derivatives

Let $\mathbb{R}^m$ be the m-dimensional Euclidean space, $\mathbb{R}_+$ the positive semi-axis, and $f(t)$, $f : \mathbb{R}_+ \to \mathbb{R}^m$, an absolutely continuous function.

Here we consider fractional derivatives of arbitrary order. Suppose $n - 1 < \alpha < n$, $n \in \mathbb{N}$ ($\mathbb{N}$ stands for the set of natural numbers) and let the fractional part of α be denoted by $\{\alpha\}$ and its integer part by $[\alpha]$. Thus, $[\alpha] = n - 1$, $\{\alpha\} = \alpha - n + 1$. The Riemann-Liouville fractional derivative of arbitrary

order α ($n-1<\alpha<n$, $n\in\mathbb{N}$) is defined as follows [9]:

$$\begin{aligned} D^{\alpha}f(t) &= \left(\tfrac{d}{dt}\right)^{[\alpha]} D^{\{\alpha\}}f(t) = \\ &= \left(\tfrac{d}{dt}\right)^{[\alpha]} \tfrac{1}{\Gamma(1-\{\alpha\})}\tfrac{d}{dt}\int_0^t \tfrac{f(\tau)}{(t-\tau)^{\{\alpha\}}}d\tau = \\ &= \tfrac{1}{\Gamma(n-\alpha)}\left(\tfrac{d}{dt}\right)^{n}\int_0^t \tfrac{f(\tau)}{(t-\tau)^{\alpha-n+1}}d\tau. \end{aligned} \tag{1.1}$$

Lemma 1 *Let $n-1<\alpha<n$, $n\in\mathbb{N}$, and the function $f(t)$ have absolutely continuous derivatives up to the order $(n-1)$. Then the following formula is true*

$$D^{\alpha}f(t)=\sum_{k=0}^{n-1}\frac{t^{k-\alpha}}{\Gamma(k-\alpha+1)}f^{(k)}(0)+\frac{1}{\Gamma(n-\alpha)}\int_0^t\frac{f^{(n)}(\tau)}{(t-\tau)^{\alpha-n+1}}d\tau. \tag{1.2}$$

Proof. The proof is by induction on n.

Let for $n-2<\alpha<n-1$

$$D^{\alpha}f(t)=\sum_{k=0}^{n-2}\frac{t^{k-\alpha}}{\Gamma(k-\alpha+1)}f^{(k)}(0)+\frac{1}{\Gamma(n-1-\alpha)}\int_0^t\frac{f^{(n-1)}(\tau)}{(t-\tau)^{\alpha-n+2}}d\tau. \tag{1.3}$$

Then for $n-1<\alpha<n$, taking into account $\{\alpha\}=\{\alpha-1\}$ and $[\alpha-1]=[\alpha]-1$ for non-integer α, we obtain

$$D^{\alpha}f(t)=\left(\frac{d}{dt}\right)^{[\alpha]}D^{\{\alpha\}}f(t)=\frac{d}{dt}\left(\frac{d}{dt}\right)^{[\alpha]-1}D^{\{\alpha\}}f(t)=\frac{d}{dt}D^{\alpha-1}f(t).$$

Since $n-2<\alpha-1<n-1$, we can employ the equation (1.3):

$$D^{\alpha}f(t)=\frac{d}{dt}D^{\alpha-1}f(t)=$$

$$=\frac{d}{dt}\left(\sum_{k=0}^{n-2}\frac{t^{k-\alpha+1}}{\Gamma(k-\alpha+2)}f^{(k)}(0)+\frac{1}{\Gamma(n-\alpha)}\int_0^t\frac{f^{(n-1)}(\tau)}{(t-\tau)^{\alpha-n+1}}d\tau\right).$$

Integrating by parts, we find that

$$D^{\alpha}f(t)=\frac{d}{dt}(\sum_{k=0}^{n-2}\frac{t^{k-\alpha+1}}{\Gamma(k-\alpha+2)}f^{(k)}(0)+\frac{t^{n-\alpha}}{\Gamma(n-\alpha)(n-\alpha)}f^{(n-1)}(0)+$$

$$+\frac{1}{\Gamma(n-\alpha)(n-\alpha)}\int_0^t \frac{f^{(n)}(\tau)}{(t-\tau)^{\alpha-n}}d\tau) = \sum_{k=0}^{n-2}\frac{(k-\alpha+1)t^{k-\alpha}}{\Gamma(k-\alpha+2)}f^{(k)}(0)+$$

$$+\frac{(n-\alpha)t^{n-\alpha-1}}{\Gamma(n-\alpha)(n-\alpha)}f^{(n-1)}(0) + \frac{(n-\alpha)}{\Gamma(n-\alpha)(n-\alpha)}\int_0^t \frac{f^{(n)}(\tau)}{(t-\tau)^{\alpha-n+1}}d\tau =$$

$$= \sum_{k=0}^{n-1}\frac{t^{k-\alpha}}{\Gamma(k-\alpha+1)}f^{(k)}(0) + \frac{1}{\Gamma(n-\alpha)}\int_0^t \frac{f^{(n)}(\tau)}{(t-\tau)^{\alpha-n+1}}d\tau,$$

which was to be proved.

As before, the integral term in (1.2) is the regularized Caputo derivative of order α ($n-1<\alpha<n$, $n\in\mathbb{N}$):

$$\begin{aligned} D^{(\alpha)}f(t) &=_0^C D_t^\alpha f(t) \\ &= \tfrac{1}{\Gamma(n-\alpha)}\int_0^t \tfrac{f^{(n)}(\tau)}{(t-\tau)^{\alpha+1-n}}d\tau \\ &= D^\alpha f(t) - \textstyle\sum_{k=0}^{n-1}\tfrac{t^{k-\alpha}}{\Gamma(k-\alpha+1)}f^{(k)}(0). \end{aligned} \tag{1.4}$$

1.4 The Miller-Ross Sequential Derivatives

Both the Riemann-Liouville and Caputo derivatives possess neither semigroup nor commutative property, i.e. in general,

$$\mathbf{D}^{\alpha+\beta}f(t) \neq \mathbf{D}^\alpha\mathbf{D}^\beta f(t),$$

$$\mathbf{D}^\alpha\mathbf{D}^\beta f(t) \neq \mathbf{D}^\beta\mathbf{D}^\alpha f(t),$$

where $\mathbf{D}^\alpha$ stands for the Riemann-Liouville or Caputo fractional differentiation operator of order α.

This fact motivated introduction by [24] of sequential derivatives, defined as follows:

$$\mathcal{D}^{|\alpha|}f(t) = \mathbf{D}^{\alpha_1}\mathbf{D}^{\alpha_2}\ldots\mathbf{D}^{\alpha_m}f(t),$$

where $\alpha=(\alpha_1,\alpha_2,\ldots,\alpha_m)$ is an m-tuple, $|\alpha|=\alpha_1+\alpha_2+\ldots+\alpha_m$, and function $f(t)$ is sufficiently smooth. In general, the operator $\mathbf{D}^\alpha$ underlying sequential Miller-Ross derivative can be either the Riemann-Liouville or Caputo or any other kind of integro-differentiation operator. In particular, in the case of integer α_i it is conventional differentiation operator $\left(\frac{d}{dt}\right)^{\alpha_i}$.

It should be noted that the Riemann-Liouville and Caputo fractional derivatives in their turn can be considered as particular cases of sequential derivatives. Indeed, suppose $n-1<\alpha<n$ and denote $p=n-\alpha$, then, by definition (1.1),

$$\begin{aligned} D^{\alpha}f(t) &= \left(\tfrac{d}{dt}\right)^{n-1} D^{\alpha-n+1}f(t) = \left(\tfrac{d}{dt}\right)^{n} D^{-p}f(t), \\ D^{(\alpha)}f(t) &= D^{(\alpha-n+1)}\left(\tfrac{d}{dt}\right)^{n-1} f(t) = D^{-p}\left(\tfrac{d}{dt}\right)^{n} f(t), \end{aligned} \tag{1.5}$$

where D^{-p} is the Riemann-Liouville left-sided integral of order $p=n-\alpha$, $0<p<1$, also denoted by J^{p}:

$$D^{-p}f(t) = J^{p}f(t) = \frac{1}{\Gamma(p)}\int_{0}^{t}\frac{f(\tau)}{(t-\tau)^{1-p}}d\tau.$$

Hereafter we assume that J^{0} is the identity operator, i.e. $J^{0}f(t)=f(t)$. It should be noted, that for existence of the Riemann-Liouville integral it is sufficient that the function $f(t)$ is locally integrable on $\mathbb{R}_{+}$ [29].

The Miller-Ross sequential derivatives make it possible to lower the order of fractional differential equations.

Here we suggest an example of constructing sequential derivatives in order to establish their relation to the Riemann-Liouville and Caputo derivatives. Let us choose some ν, $n-1<\nu<n$, $n\in\mathbb{N}$, and let us study the case when

$$\alpha=(j,\nu-n+1,n-1-j)=(j,\{\nu\},[\nu]-j),\ j=0,\ldots,n-1.$$

Obviously $|\alpha|=\nu$. Let us introduce the following notation

$$\mathcal{D}_{j}^{\nu}f(t)=\left(\frac{d}{dt}\right)^{j}D^{\{\nu\}}\left(\frac{d}{dt}\right)^{[\nu]-j}f(t)=\left(\frac{d}{dt}\right)^{j}D^{\nu-n+1}\left(\frac{d}{dt}\right)^{n-1-j}f(t),$$

$$\mathcal{D}_{j}^{(\nu)}f(t)=\left(\frac{d}{dt}\right)^{j}D^{(\{\nu\})}\left(\frac{d}{dt}\right)^{[\nu]-j}f(t)=\left(\frac{d}{dt}\right)^{j}D^{(\nu-n+1)}\left(\frac{d}{dt}\right)^{n-1-j}f(t),$$

where $j=0,1,\ldots,n-1$.

The following lemma shows a relationship between the sequential derivatives $\mathcal{D}_{j}^{\nu}f(t)$, $\mathcal{D}_{j}^{(\nu)}f(t)$ and classical derivatives of Riemann-Liouville and Caputo.

Lemma 2 *Let $n-1<\nu<n$, $n\in\mathbb{N}$, and the function $f(t)$ have absolutely continuous derivatives up to the order $(n-1)$. Then the following equalities hold true*

$$\begin{gathered}
\mathcal{D}_0^{(\nu)}f(t)=D^{(\nu)}f(t)=D^{\nu}f(t)-\sum_{k=0}^{n-1}\frac{t^{k-\nu}}{\Gamma(k-\nu+1)}f^{(k)}(0),\\
\mathcal{D}_1^{(\nu)}f(t)=\mathcal{D}_0^{\nu}f(t)=D^{(\nu)}f(t)+\frac{t^{n-1-\nu}}{\Gamma(n-\nu)}f^{(n-1)}(0)\\
=D^{\nu}f(t)-\sum_{k=0}^{n-2}\frac{t^{k-\nu}}{\Gamma(k-\nu+1)}f^{(k)}(0),\\
\cdots\cdots\cdots\\
\mathcal{D}_j^{(\nu)}f(t)=\mathcal{D}_{j-1}^{\nu}f(t)=D^{(\nu)}f(t)+\sum_{k=n-j}^{n-1}\frac{t^{k-\nu}}{\Gamma(k-\nu+1)}f^{(k)}(0)\\
=D^{\nu}f(t)-\sum_{k=0}^{n-1-j}\frac{t^{k-\nu}}{\Gamma(k-\nu+1)}f^{(k)}(0),\\
\cdots\cdots\cdots\\
\mathcal{D}_{n-1}^{(\nu)}f(t)=\mathcal{D}_{n-2}^{\nu}f(t)=D^{(\nu)}f(t)+\sum_{k=1}^{n-1}\frac{t^{k-\nu}}{\Gamma(k-\nu+1)}f^{(k)}(0)\\
=D^{\nu}f(t)-\frac{t^{-\nu}}{\Gamma(1-\nu)}f(0),\\
\mathcal{D}_{n-1}^{\nu}f(t)=D^{(\nu)}f(t)+\sum_{k=0}^{n-1}\frac{t^{k-\nu}}{\Gamma(k-\nu+1)}f^{(k)}(0)=D^{\nu}f(t).
\end{gathered}$$

Proof. It is evident that

$$\mathcal{D}_0^{(\nu)}f(t)=D^{(\nu-n+1)}\left(\frac{d}{dt}\right)^{n-1}f(t)=\frac{1}{\Gamma(n-\nu)}\int_0^t\frac{f^{(n)}(\tau)}{(t-\tau)^{\nu+1-n}}d\tau,$$

whence, by virtue of (1.4),

$$\mathcal{D}_0^{(\nu)}f(t)=D^{(\nu)}f(t)=D^{\nu}f(t)-\sum_{k=0}^{n-1}\frac{t^{k-\nu}}{\Gamma(k-\nu+1)}f^{(k)}(0).$$

Let $j=1,\ldots,n-1$, then

$$\mathcal{D}_j^{(\nu)}f(t)=\left(\frac{d}{dt}\right)^{j}D^{(\nu-n+1)}\left(\frac{d}{dt}\right)^{n-1-j}f(t)=$$

$$=\left(\frac{d}{dt}\right)^{j}\frac{1}{\Gamma(n-\nu)}\int_0^t\frac{f^{(n-j)}(\tau)}{(t-\tau)^{\nu+1-n}}d\tau=$$

$$=\left(\frac{d}{dt}\right)^{j-1}\frac{1}{\Gamma(n-\nu)}\frac{d}{dt}\int_0^t\frac{f^{(n-j)}(\tau)}{(t-\tau)^{\nu+1-n}}d\tau=$$

$$=\left(\frac{d}{dt}\right)^{j-1}D^{\nu-n+1}\left(\frac{d}{dt}\right)^{n-j}f(t)=\mathcal{D}_{j-1}^{\nu}f(t).$$

On the other hand, in virtue of (1.4)

$$\mathcal{D}_j^{(\nu)} f(t) = \mathcal{D}_{j-1}^{\nu} f(t) = \left(\frac{d}{dt}\right)^j \frac{1}{\Gamma(n-\nu)} \int_0^t \frac{f^{(n-j)}(\tau)}{(t-\tau)^{\nu+1-n}} d\tau =$$

$$= \left(\frac{d}{dt}\right)^j D^{(\nu-j)} f(t) = \left(\frac{d}{dt}\right)^j \left[D^{\nu-j} f(t) - \sum_{k=0}^{n-1-j} \frac{t^{k-\nu+j}}{\Gamma(k-\nu+j+1)} f^{(k)}(0) \right] =$$

$$= D^{\nu} f(t) - \sum_{k=0}^{n-1-j} \frac{t^{k-\nu}}{\Gamma(k-\nu+1)} f^{(k)}(0).$$

The equality

$$\mathcal{D}_{n-1}^{\nu} f(t) = D^{(\nu)} f(t) + \sum_{k=0}^{n-1} \frac{t^{k-\nu}}{\Gamma(k-\nu+1)} f^{(k)}(0) = D^{\nu} f(t)$$

is the direct consequence of (1.1) and Lemma 1. Taking into account the equalities $\mathcal{D}_j^{(\nu)} f(t) = \mathcal{D}_{j-1}^{\nu} f(t)$ and setting $\mathcal{D}_n^{(\nu)} f(t) = D^{\nu} f(t)$ one can introduce common notation

$$\mathfrak{D}_j^{\nu} f(t) = \mathcal{D}_j^{(\nu)} f(t) = \mathcal{D}_{j-1}^{\nu} f(t),$$

where $n-1 < \nu < n$, $j = 0, \ldots, n$. It goes without saying that $\mathfrak{D}_0^{\nu} f(t) = D^{(\nu)} f(t)$ and $\mathfrak{D}_n^{\nu} f(t) = D^{\nu} f(t)$. In the sequel the formula

$$\mathfrak{D}_j^{\nu} f(t) = \left(\frac{d}{dt}\right)^j D^{(\nu-j)} f(t) \tag{1.6}$$

will be of use.

The Laplace transform is a powerful tool formulated to solve a wide variety of initial-value problems. The strategy is to transform the difficult differential equations into simple algebra problems where solutions can be easily obtained. One then applies the Inverse Laplace transform to retrieve the solutions of the original problems.

For a function $f(t)$, $f : \mathbb{R}_+ \to \mathbb{R}^m$, its Laplace transform is denoted as $L\{f(t); s\}$ obtained by the following integral:

$$L\{f(t); s\} = \int_0^{\infty} f(t) e^{-st} dt,$$

where $s \in \mathbb{C}$ ($\mathbb{C}$ stands for the set of complex numbers). The Laplace transform of $f(t)$ exists whenever the following conditions hold true [1]:

1. $f(t)$ is measurable and bounded, hence locally summable;
2. There exists real constants K, a, such that $\|f(t)\| \leq Ke^{at}$, which implies that $f(t)e^{-st}$ is integrable on $\mathbb{R}_+$ (in symbols $f(t)e^{-st} \in L_1(\mathbb{R}_+)$).

Hereafter, $\|\cdot\|$ stands for the Euclidean norm in $\mathbb{R}^m$.

If the Laplace transform of $f(t)$ is $F(s)$, then $f(t)$ is said to be the Inverse Laplace Transform of $F(s)$ or $L^{-1}\{F(s)\} = f(t)$, where L^{-1} is called the Inverse Laplace Transform Operator.

The following formulas for the Laplace transforms of the Riemann-Liouville and Caputo fractional derivatives hold true [27]:

$$L\{D^{\alpha}f(t);s\} = s^{\alpha}F(s) - \sum_{k=0}^{n-1} s^k D^{\alpha-k-1}f(t)|_{t=0}, \tag{1.7}$$

$$L\{D^{(\alpha)}f(t);s\} = s^{\alpha}F(s) - \sum_{k=0}^{n-1} s^{\alpha-k-1} f^{(k)}(0). \tag{1.8}$$

For integer n the following formula is true:

$$L\{f^{(n)}(t);s\} = s^{n}F(s) - \sum_{k=0}^{n-1} s^k f^{(n-k-1)}(0). \tag{1.9}$$

Using (1.6), (1.8), (1.9), one can derive Laplace transform of the derivative $\mathfrak{D}_j^{\nu}f(t)$:

$$\begin{gathered} L\{\mathfrak{D}_j^{\nu}f(t);s\} = L\left\{\left(\tfrac{d}{dt}\right)^j D^{(\nu-j)}f(t);s\right\} = \\ = s^j L\left\{D^{(\nu-j)}f(t);s\right\} - \textstyle\sum_{k=0}^{j-1} s^k \left(\tfrac{d}{dt}\right)^{j-k-1} D^{(\nu-j)}f(t)|_{t=0} = \\ = s^j L\left\{D^{(\nu-j)}f(t);s\right\} - \textstyle\sum_{k=0}^{j-1} s^k \mathfrak{D}_{j-k-1}^{\nu-k-1}f(t)|_{t=0} = \\ = s^{\nu}F(s) - \textstyle\sum_{l=0}^{n-j-1} s^{\nu-l-1}f^{(l)}(0) - \sum_{k=0}^{j-1} s^k \mathfrak{D}_{j-k-1}^{\nu-k-1}f(t)|_{t=0}. \end{gathered} \tag{1.10}$$

Setting $\sum_{l=0}^{-1} s^{\nu-l-1}f^{(l)}(0) = 0$, $\sum_{k=0}^{-1} s^k \mathfrak{D}_{j-k-1}^{\nu-k-1}f(t)|_{t=0} = 0$, and taking into account that $\mathfrak{D}_{n-k-1}^{\nu-k-1}f(t) = D^{\nu-k-1}f(t)$, one can see that (1.7), (1.8) are

particular cases of (1.10):

$$L\{\mathfrak{D}_0^\nu f(t);s\} = L\{D^{(\nu)} f(t);s\},$$
$$L\{\mathfrak{D}_n^\nu f(t);s\} = L\{D^{\nu} f(t);s\}.$$

1.5 Hilfer's Derivative

Hilfer's derivative is another generalization encompassing both the Riemann-Liouville and Caputo derivatives as particular cases.

According to [27], let us introduce the fractional derivative of order α, $0 < \alpha \leq 1$, and type μ, $0 \leq \mu \leq 1$, in the following way:

$$D^{\alpha,\mu} f(t) = J^{\mu(1-\alpha)} \frac{d}{dt} J^{(1-\mu)(1-\alpha)} f(t).$$

At $\mu = 0$ this definition yields classic Riemann-Liouville derivative and at $\mu = 1$ – the Caputo regularized derivative.

Note that the definition of the Hilfer derivative can be extended to higher orders. Let $n-1 < \alpha < n$, $n \in \mathbb{N}$, and the function f has absolutely continuous derivatives up to the order n. Let us set the derivative of order α and type μ, $0 \leq \mu \leq 1$, to be equal

$$D^{\alpha,\mu} f(t) = J^{\mu(n-\alpha)} \frac{d^n}{dt^n} J^{(1-\mu)(n-\alpha)} f(t). \tag{1.11}$$

In virtue of (1.5), at $\mu = 0$ we, as before, obtain the Riemann-Liouville derivative of arbitrary order α

$$D^{\alpha,0} f(t) = D^\alpha f(t) = \frac{d^n}{dt^n} J^{n-\alpha} f(t), \tag{1.12}$$

and at $\mu = 1$ – the Caputo derivative

$$D^{\alpha,1} f(t) = D^{(\alpha)} f(t) = J^{n-\alpha} \frac{d^n}{dt^n} f(t).$$

To derive the Laplace transform of the higher-order Hilfer derivative (1.11), we employ (1.9) and the formula:

$$L\{J^\alpha f(t);s\} = s^{-\alpha} F(s), \tag{1.13}$$

Then, in view of (1.9), (1.13), we obtain

$$L\{D^{\alpha,\mu}f(t);s\} = L\left\{J^{\mu(n-\alpha)}\frac{d^n}{dt^n}J^{(1-\mu)(n-\alpha)}f(t);s\right\} =$$

$$= s^{\mu(\alpha-n)}L\left\{\frac{d^n}{dt^n}J^{(1-\mu)(n-\alpha)}f(t);s\right\} = s^{\mu(\alpha-n)}[s^nL\{J^{(1-\mu)(n-\alpha)}f(t);s\}+$$

$$-\sum_{i=0}^{n-1}s^i\frac{d^{n-i-1}}{dt^{n-i-1}}J^{(1-\mu)(n-\alpha)}f(t)|_{t=0}].$$

Finally, we have

$$L\{D^{\alpha,\mu}f(t);s\} = s^\alpha F(s) - \sum_{i=0}^{n-1}s^{\mu(\alpha-n)+i}\frac{d^{n-i-1}}{dt^{n-i-1}}J^{(1-\mu)(n-\alpha)}f(t)|_{t=0}. \quad (1.14)$$

By virtue of (1.12), the equation (1.14) yields for for $\mu = 0$ the Laplace transform (1.7) of the Riemann-Liouville derivative. For $\mu = 1$, we obtain from (1.14) the Laplace transform (1.8) for the Caputo derivative of arbitrary order α, $n-1 < \alpha < n$.

1.6 The Mittag-Leffler Generalized Matrix Function

In [12] the Mittag-Leffler generalized matrix function was introduced:

$$E_\rho(B;\mu) = \sum_{k=0}^{\infty}\frac{B^k}{\Gamma(k\rho^{-1}+\mu)},$$

where $\rho > 0$, $\mu \in \mathbb{C}$, and B is an arbitrary square matrix of order m.

The Mittag-Leffler generalized matrix function plays important role in studying the linear systems of fractional order. Denote by I the identity matrix of order m. The following lemma allows to find the Laplace transforms of expressions involving the Mittag-Leffler matrix function.

Lemma 3 *Let $\alpha > 0$, $\beta > 0$, and let A be an arbitrary square matrix of order m. Then the following formula is true:*

$$L\left\{t^{\beta-1}E_{\frac{1}{\alpha}}(At^\alpha;\beta);s\right\} = s^{\alpha-\beta}(s^\alpha I - A)^{-1}.$$

Proof. Taking into account definitions of the Mittag-Leffler generalized matrix function, Gamma-function, and substituting $\tau = st$, we obtain:

$$L\left\{t^{\beta-1}E_{\frac{1}{\alpha}}(At^{\alpha};\beta);s\right\} = \int_0^\infty e^{-st}t^{\beta-1}E_{\frac{1}{\alpha}}(At^{\alpha};\beta)dt$$

$$= \int_0^\infty e^{-st}t^{\beta-1}\sum_{k=0}^{\infty}\frac{A^k t^{\alpha k}}{\Gamma(\alpha k+\beta)}dt = \sum_{k=0}^{\infty}\frac{A^k}{\Gamma(\alpha k+\beta)}\int_0^\infty e^{-st}t^{\alpha k+\beta-1}dt$$

$$= \sum_{k=0}^{\infty}\frac{A^k}{\Gamma(\alpha k+\beta)s^{\alpha k+\beta}}\int_0^\infty e^{-\tau}\tau^{\alpha k+\beta-1}d\tau = \sum_{k=0}^{\infty}A^k s^{-(\alpha k+\beta)}.$$

Now let us show that $\sum_{k=0}^{\infty}A^k s^{-(\alpha k+\beta)} = s^{\alpha-\beta}(s^\alpha I - A)^{-1}$. The last formula is equivalent to the equation

$$\sum_{k=0}^{\infty}A^k s^{-(k+1)\alpha} = (s^\alpha I - A)^{-1}. \tag{1.15}$$

Let us multiply both sides of (1.15) by $(s^\alpha I - A)$ (either on the left or on the right, it makes no difference as these matrices commute). We obtain

$$\sum_{k=0}^{\infty}A^k s^{-(k+1)\alpha}(s^\alpha I - A) = \sum_{k=0}^{\infty}A^k s^{-k\alpha} - \sum_{k=0}^{\infty}A^{(k+1)}s^{-(k+1)\alpha} = I.$$

Since the inverse matrix is unique, this completes the proof.

1.7 Fractional Order Systems, Cauchy Formula

Suppose $g(t)$, $g : \mathbb{R}_+ \to \mathbb{R}^m$, is a measurable and bounded function, then $g(t)e^{-st} \in L_1(\mathbb{R}_+)$, $s \in \mathbb{C}$. Thus, $g(t)$ is Laplace transformable.

Consider a dynamic system whose evolution is described by the equation:

$$D^\alpha z = Az + g, \quad n-1 < \alpha < n, \tag{1.16}$$

under the initial conditions

$$D^{\alpha-k}z(t)|_{t=0} = z_{k1}^0, \quad k = 1,\dots,n. \tag{1.17}$$

Hereafter, the state vector z belongs to the m-dimensional real Euclidean space $\mathbb{R}^m$, A is a square matrix of order m.

Lemma 4 *The trajectory of the system (1.16), (1.17) has the form:*

$$z(t) = \sum_{k=1}^{n} t^{\alpha-k} E_{\frac{1}{\alpha}}(At^{\alpha}; \alpha - k + 1) z_{k1}^{0} + \int_{0}^{t} (t-\tau)^{\alpha-1} E_{\frac{1}{\alpha}}(A(t-\tau)^{\alpha}; \alpha) g(\tau) d\tau. \tag{1.18}$$

Proof. Let us apply the Laplace transform to (1.16), taking into account (1.7). We get:

$$s^{\alpha} Z(s) - \sum_{k=1}^{n} s^{k-1} z_{k1}^{0} = AZ(s) + G(s),$$

or

$$Z(s) = \sum_{k=1}^{n} s^{k-1} (s^{\alpha} I - A)^{-1} z_{k1}^{0} + (s^{\alpha} I - A)^{-1} G(s). \tag{1.19}$$

Let us apply the inverse Laplace transform to (1.19). Taking into account Lemma 3 we have

$$z(t) = \sum_{k=1}^{n} t^{\alpha-k} E_{\frac{1}{\alpha}}(At^{\alpha}; \alpha - k + 1) z_{k1}^{0} + \int_{0}^{t} (t-\tau)^{\alpha-1} E_{\frac{1}{\alpha}}(A(t-\tau)^{\alpha}; \alpha) g(\tau) d\tau.$$

Which was to be proved.

Now consider a dynamic system of fractional order in the sense of Caputo described by the equation:

$$D^{(\alpha)} z = Az + g, \quad n - 1 < \alpha < n, \tag{1.20}$$

under the initial conditions

$$z^{(k)}(0) = z_{k2}^{0}, \quad k = 0, \ldots, n-1. \tag{1.21}$$

Lemma 5 *The trajectory of the system (1.20), (1.21) has the form:*

$$z(t) = \sum_{k=0}^{n-1} t^{k} E_{\frac{1}{\alpha}}(At^{\alpha}; k+1) z_{k2}^{0} + \int_{0}^{t} (t-\tau)^{\alpha-1} E_{\frac{1}{\alpha}}(A(t-\tau)^{\alpha}; \alpha) g(\tau) d\tau. \tag{1.22}$$

Proof. Let us apply the Laplace transform to (1.20). Taking into account (1.8), we deduce:

$$s^{\alpha}Z(s) - \sum_{k=0}^{n-1} s^{\alpha-k-1} z_{k2}^{0} = AZ(s) + G(s),$$

or

$$Z(s) = \sum_{k=0}^{n-1} s^{\alpha-k-1}(s^{\alpha}I - A)^{-1} z_{k2}^{0} + (s^{\alpha}I - A)^{-1} G(s). \tag{1.23}$$

Now apply the inverse Laplace transform to (1.23). Then, in view of Lemma 3, we have:

$$z(t) = \sum_{k=0}^{n-1} t^{k} E_{\frac{1}{\alpha}}(At^{\alpha}; k+1) z_{k2}^{0} + \int_{0}^{t} (t-\tau)^{\alpha-1} E_{\frac{1}{\alpha}}(A(t-\tau)^{\alpha}; \alpha) g(\tau) d\tau.$$

It should be noted that the formulas (1.18), (1.22) in some specific cases were obtained in [10, 29], by a different method.

Now, let us study systems involving sequential derivatives of special form $\mathfrak{D}_{j}^{\alpha}$. Consider a dynamic system whose evolution is described by the equation:

$$\mathfrak{D}_{j}^{\alpha} z = Az + g, \ n-1 < \alpha < n, \ j \in \{0, 1, \ldots, n\} \tag{1.24}$$

under the initial conditions

$$\begin{array}{ll} \mathfrak{D}_{j-k-1}^{\alpha-k-1} z(t)|_{t=0} & = \tilde{z}_{k}^{0}, \quad k = 0, \ldots, j-1, \\ z^{(l)}(0) & = z_{l}^{0}, \quad l = 0, \ldots, n-j-1. \end{array} \tag{1.25}$$

Lemma 6 *The trajectory of the system (1.24), (1.25) has the form:*

$$z(t) = \sum_{l=0}^{n-j-1} t^{l} E_{\frac{1}{\alpha}}(At^{\alpha}; l+1) z_{l}^{0} + \sum_{k=0}^{j-1} t^{\alpha-k-1} E_{\frac{1}{\alpha}}(At^{\alpha}; \alpha-k) \tilde{z}_{k}^{0} + \\ + \int_{0}^{t} (t-\tau)^{\alpha-1} E_{\frac{1}{\alpha}}(A(t-\tau)^{\alpha}; \alpha) g(\tau) d\tau.$$

Proof. Let us apply the Laplace transform to the system (1.24) with account of (1.10). Then we obtain:

$$s^{\alpha}Z(s) - \sum_{l=0}^{n-j-1} s^{\alpha-l-1} z_{l}^{0} - \sum_{k=0}^{j-1} s^{k} \tilde{z}_{k}^{0} = AZ(s) + G(s),$$

or

$$Z(s) = \sum_{l=0}^{n-j-1} s^{\alpha-l-1}(s^\alpha I - A)^{-1} z_l^0 + \sum_{k=0}^{j-1} s^k (s^\alpha I - A)^{-1} \tilde{z}_k^0 + (s^\alpha I - A)^{-1} G(s). \tag{1.26}$$

Applying the inverse Laplace transform to (1.26) and taking into account Lemma 3, we find:

$$z(t) = \sum_{l=0}^{n-j-1} t^l E_{\frac{1}{\alpha}}(At^\alpha; l+1) z_l^0 + \sum_{k=0}^{j-1} t^{\alpha-k-1} E_{\frac{1}{\alpha}}(At^\alpha; \alpha-k) \tilde{z}_k^0 +$$

$$+ \int_0^t (t-\tau)^{\alpha-1} E_{\frac{1}{\alpha}}(A(t-\tau)^\alpha; \alpha) g(\tau) d\tau.$$

Which was to be proved.

Finally, let us study systems involving Hilfer derivatives of order α and type μ. Consider a dynamic system whose evolution is described by the equation:

$$D^{\alpha,\mu} z = Az + f, \quad n-1 < \alpha < n, \quad 0 \le \mu \le 1, \tag{1.27}$$

under the initial conditions

$$\frac{d^i}{dt^i} J^{(1-\mu)(n-\alpha)} z(t)|_{t=0+} = \hat{z}_i^0, \quad i = 0, \ldots, n-1. \tag{1.28}$$

Lemma 7 *The trajectory of the system (1.27), (1.28) has the form:*

$$z(t) = \sum_{i=0}^{n-1} t^{i-(1-\mu)(n-\alpha)} E_{\frac{1}{\alpha}}(At^\alpha; i-(1-\mu)(n-\alpha)+1) \hat{z}_i^0 +$$

$$+ \int_0^t (t-\tau)^{\alpha-1} E_{\frac{1}{\alpha}}(A(t-\tau)^\alpha; \alpha) f(\tau) d\tau.$$

Proof. Applying the Laplace transform to the both sides of (1.27), we obtain:

$$s^\alpha Z - \sum_{i=0}^{m-1} s^{\mu(\alpha-m)+i} \hat{z}_{m-i-1}^0 = AZ + F,$$

whence:

$$Z(s) = \sum_{i=0}^{m-1} s^{\mu(\alpha-m)+i} (s^\alpha I - A)^{-1} \hat{z}_{m-i-1}^0 + (s^\alpha I - A)^{-1} F(s).$$

In view of Lemma 3 and of the obvious equality $\sum_{i=0}^{m-1} a_i = \sum_{i=0}^{m-1} a_{m-1-i}$, the inverse Laplace transform yields:

$$z(t) = \sum_{i=0}^{n-1} t^{i-(1-\mu)(n-\alpha)} E_{\frac{1}{\alpha}}(At^{\alpha}; i-(1-\mu)(n-\alpha)+1)\hat{z}_i^0 +$$

$$+ \int_0^t (t-\tau)^{\alpha-1} E_{\frac{1}{\alpha}}(A(t-\tau)^{\alpha}; \alpha) f(\tau) d\tau.$$

1.8 Game Problem Statement

In this section a statement for the problem of approaching the terminal set will be given for conflict-controlled processes, the dynamics of which is described using fractional derivatives of Riemann-Liouville, Caputo, and Miller-Ross.

Consider conflict-controlled process whose evolution is defined by the fractional order system:

$$\mathbf{D}^{\alpha} z = Az + \varphi(u, v), \quad n-1 < \alpha < n. \tag{1.29}$$

Hereafter $\mathbf{D}^{\alpha}$ stands for the operator of fractional differentiation in the sense of Riemann-Liouville, Caputo, Miller-Ross, or Hilfer. It will be clear from the context which type of the fractional differentiation operator is meant. Here, as before, $z \in \mathbb{R}^m$, A is a square matrix of order m. The control block is defined by the jointly continuous function $\varphi(u, v)$, $\varphi : U \times V \to \mathbb{R}^m$, where u and v, $u \in U$, $v \in V$, are control parameters of the first and second players respectively, and the control sets U and V are from the set $K(\mathbb{R}^m)$ of all nonempty compact subsets of $\mathbb{R}^m$.

When $\mathbf{D}^{\alpha}$ is the operator of fractional differentiation in the sense of Riemann-Liouville, i.e. $\mathbf{D}^{\alpha} = D^{\alpha}$, the initial conditions for the process (1.29) are given in the form: (1.17). In this case denote : $z^0 = (z_{11}^0, \ldots, z_{n1}^0)$. When the derivative in (1.29) is understood in Caputo's sense $\mathbf{D}^{\alpha} = D^{(\alpha)}$, the initial conditions are of the form (1.21) and $z^0 = (z_{02}^0, \ldots, z_{n-12}^0)$. For sequential derivatives of special form $\mathbf{D}^{\alpha} = \mathfrak{D}_j^{\alpha}$ the initial conditions are given by (1.25) and $z^0 = (\tilde{z}_0^0, \ldots, \tilde{z}_{j-1}^0, z_0^0, \ldots, z_{n-j-1}^0)$. Finally, if $\mathbf{D}^{\alpha}$ stands for the Hilfer

fractional differentiation operator i.e. $\mathbf{D}^{\alpha} = D^{\alpha,\mu}$ for some $0 \leq \mu \leq 1$, then the initial conditions are of the form (1.28) and we denote $z^0 = (\hat{z}_0^0, \ldots, \hat{z}_{n-1}^0)$.

Along with the process dynamics (1.29) and the initial conditions a terminal set of cylindrical form is given:

$$M^* = M_0 + M, \tag{1.30}$$

where M_0 is a linear subspace of $\mathbb{R}^m$, $M \in K(L)$, and $L = M_0^{\perp}$ is the orthogonal complement of the subspace M_0 in $\mathbb{R}^m$.

When the controls of the both players are chosen in the form of Lebesgue measurable functions $u(t)$ and $v(t)$ taking values from U and V respectively, the Cauchy problem for the process (1.29) with corresponding initial values has a unique absolutely continuous solution [24, 29].

Consider the following dynamic game. The first player aims to bring a trajectory of the process (1.29) to the set (1.30), while the other player strives to delay the moment of hitting the terminal set as much as possible. We assume that the second player's control is an arbitrary measurable function $v(t)$ taking values from V, and the first player at each time instant t, $t \geq 0$, forms his control on the basis of information about z^0 and $v(t)$:

$$u(t) = u(z^0, v(t)), \quad u(t) \in U. \tag{1.31}$$

Therefore, $u(t)$ is Krasovskii's counter-control [39] prescribed by the O. Hajek stroboscopic strategies [26].

By solving the problem stated above we employ the Method of Resolving Functions [34, 12]. Usually this method implements the pursuit process in the class of quasistrategies. However in this paper we use the results from [13] providing sufficient conditions for the termination of the pursuit in the aforementioned method with the help of counter-controls.

1.8.1 Method of Resolving Functions

Denote Π the orthoprojector from $\mathbb{R}^m$ onto L. Set $\varphi(U, v) = \{\varphi(u, v) : u \in U\}$ and consider set-valued mappings:

$$W(t, v) = \Pi t^{\alpha-1} E_{\frac{1}{\alpha}}(At^{\alpha}; \alpha)\varphi(U, v), \quad W(t) = \bigcap_{v \in V} W(t, v)$$

defined on the sets $\mathbb{R}_+ \times V$ and $\mathbb{R}_+$, respectively. The condition:

$$W(t) \neq \emptyset, \quad t \in \mathbb{R}_+, \tag{1.32}$$

is usually referred o as Pontryagin's condition. This condition reflects some kind of first player's advantage in resources over the second player. In the case when the condition (1.32) fails, i.e. for some $t \in \mathbb{R}_+$, $W(t) = \emptyset$, we will use the modified Pontryagin's condition. It consists in rearranging resources in favor of the first player. Namely, at the moments when $W(t) = \emptyset$ the players' control resources are equalized and the resource consumed for this action is then subtracted from the terminal set. Formally the procedure is arranged as follows. A measurable bounded with respect to t matrix function $C(t)$ is introduced. Consider set-valued mappings:

$$W^*(t,v) = \Pi t^{\alpha-1} E_{\frac{1}{\alpha}}(At^\alpha;\alpha)\varphi(U, C(t)v),$$

$$W^*(t) = \bigcap_{v \in V} W^*(t,v),$$

$$M(t) = M \stackrel{*}{-\!-} \int_0^t \tau^{\alpha-1} \Pi E_{\frac{1}{\alpha}}(A\tau^\alpha;\alpha)\varphi^*(\tau, U, V)d\tau,$$

where $\varphi^*(t,u,v) = \varphi(u,v) - \varphi(u, C(t)v)$ and $X \stackrel{*}{-\!-} Y = \{z : \ z + Y \subset X\} = \bigcap_{y \in Y}(X - y)$ is the Minkowski (geometrical) subtraction [8]. By the integral of the set-valued mapping we mean the Aumann integral, i.e. a union of integrals of all possible measurable selectors of the given set-valued mapping [30]. Hereafter, we will say that the modified Pontryagin condition is fulfilled whenever a measurable bounded matrix function $C(t)$ exists such that:

$$W^*(t) \neq \emptyset \ \forall t \in \mathbb{R}_+, \tag{1.33}$$

$$M(t) \neq \emptyset \ \forall t \in \mathbb{R}_+. \tag{1.34}$$

Thus, Pontryagin's condition (1.32) is replaced with the conditions (1.33), (1.34). Obviously, as $C(t) = I$, the condition (1.34) is fulfilled by default and the condition (1.33) coincides with the condition (1.32), since in this case $W^*(t) \equiv W(t)$. It follows that modified Pontryagin's condition (1.33), (1.34) is, generally speaking, less restrictive assumption than Pontryagin's condition (1.32).

By virtue of the properties of the process (1.29) parameters the set-valued mapping $\varphi(U, C(t)v)$, $v \in V$, is continuous in the Hausdorff metric. Therefore, taking into account the analytical properties of the Mittag-Leffler generalized matrix function, the set-valued mapping $W^*(t,v)$ is measurable with

respect to t, $t \in \mathbb{R}_+$, and closed with respect to v, $v \in V$. Then [4] the set-valued mapping $W^*(t)$ is measurable with respect to t and closed-valued. Let $P(\mathbb{R}^m)$ be the collection of all nonempty closed subsets of space $\mathbb{R}^m$. Then it is evident that:

$$W^*(t,v) : \mathbb{R}_+ \times V \to P(\mathbb{R}^m),$$
$$W^*(t) : \mathbb{R}_+ \to P(\mathbb{R}^m).$$

In this case the measurable with respect to t set-valued mappings $W^*(t,v)$, $W^*(t)$ are said to be normal [30].

It follows from the condition (1.33) and from the measurable choice theorem [4] that there exists at least one measurable selection $\gamma(\cdot)$ such that $\gamma(t) \in W^*(t)$, $t \in \mathbb{R}_+$. Denote by Γ the set of all such selections.

Let us also denote by $g(t, z^0)$ the solution of homogeneous system: $\mathbf{D}^\alpha z = Az$, obtained from (1.29) by setting $\varphi(u,v) \equiv 0$. Thus, when $\mathbf{D}^\alpha$ is the Riemann-Liouville fractional differentiation operator ($\mathbf{D}^\alpha = D^\alpha$):

$$g(t,z^0) = \sum_{k=1}^{n} t^{\alpha-k} E_{\frac{1}{\alpha}}(At^\alpha; \alpha - k + 1) z_k^0,$$

where:

$$z_k^0 = D^{\alpha-k} z(t)|_{t=0}, \quad k = 1, \ldots, n.$$

For the Caputo regularized fractional derivative ($\mathbf{D}^\alpha = D^{(\alpha)}$), we have:

$$g(t,z^0) = \sum_{k=0}^{n-1} t^k E_{\frac{1}{\alpha}}(At^\alpha; k+1) z_k^0,$$

where:

$$z_k^0 = z^{(k)}(0), \quad k = 0, \ldots, n-1.$$

And for the sequential derivatives $\mathbf{D}^\alpha = \mathsf{D}_j^\alpha$, we obtain:

$$g(t,z^0) = \sum_{l=0}^{n-j-1} t^l E_{\frac{1}{\alpha}}(At^\alpha; l+1) z_l^0 + \sum_{k=0}^{j-1} t^{\alpha-k-1} E_{\frac{1}{\alpha}}(At^\alpha; \alpha - k) \tilde{z}_k^0,$$

where:

$$\begin{aligned} \tilde{z}_k^0 &= \mathsf{D}_{j-k-1}^{\alpha-k-1} z(t)|_{t=0}, \quad k = 0, \ldots, j-1, \\ z_l^0 &= z^{(l)}(0), \quad l = 0, \ldots, n-j-1. \end{aligned}$$

In the case of the Hilfer derivative $\mathbf{D}^{\alpha} = D^{\alpha,\mu}$ the solution to the homogeneous system takes on the form:

$$g(t, z^0) = \sum_{i=0}^{n-1} t^{i-(1-\mu)(n-\alpha)} E_{\frac{1}{\alpha}}(At^{\alpha}; i-(1-\mu)(n-\alpha)+1)\hat{z}_i^0,$$

where:

$$\hat{z}_i^0 = \frac{d^i}{dt^i} J^{(1-\mu)(n-\alpha)} z(t)|_{t=0+}, \quad i = 0, \ldots, n-1.$$

Let us introduce the function:

$$\xi(t) = \Pi g(t, z^0) + \int_0^t \gamma(\tau) d\tau, \quad t \in \mathbb{R}_+,$$

where $\gamma(\cdot) \in \Gamma$ is a certain fixed selection. By virtue of the assumptions made, the selection $\gamma(\cdot)$ is summable.

Consider the set-valued mapping:

$$\Re(t, \tau, v) = \{\alpha \geq 0 : [W^*(t-\tau, v) - \gamma(t-\tau)] \bigcap \alpha[M(t) - \xi(t)] \neq \emptyset\},$$

defined on $\Delta \times V$, where $\Delta = \{(t, \tau) : \ 0 \leq \tau \leq t < \infty\}$. Let us study its support function in the direction of $+1$:

$$\rho(t, \tau, v) = \sup\{\rho : \ \rho \in \Re(t, \tau, v)\}, \quad (t, \tau) \in \Delta, \quad v \in V.$$

This function is called the resolving function [34].

Taking into account the modified Pontryagin condition (1.33), (1.34), the properties of the conflict-controlled process (1.29) parameters, as well as characterization and inverse image theorems, one can show that the set-valued mapping $\Re(t, \tau, v)$ is $\Lambda \times \mathcal{B}$-measurable [4, 10] with respect to τ, v, $\tau \in [0, t]$, $v \in V$, and the resolving function $\rho(t, \tau, v)$ is $\Lambda \times \mathcal{B}$-measurable in τ, v by virtue of the support function theorem [4] when $\xi(t) \notin M(t)$.

It should be noted that for $\xi(t) \in M(t)$ we have $\Re(t, \tau, v) = [0, \infty)$ and hence $\rho(t, \tau, v) = +\infty$ for any $\tau \in [0, t]$, $v \in V$.

Denote:

$$\mathfrak{T} = \left\{ t \in \mathbb{R}_+ : \ \int_0^t \inf_{v \in V} \rho(t, \tau, v) d\tau \geq 1 \right\}. \tag{1.35}$$

If for some $t > 0$ $\xi(t) \notin M(t)$, we assume the function $\inf\limits_{v\in V} \rho(t,\tau,v)$ to be measurable with respect to τ, $\tau \in [0,t]$. If it is not the case, then let us define the set $\mathfrak{T}$ as follows:

$$\mathfrak{T} = \left\{ t \in \mathbb{R}_+ : \inf_{v(\cdot)} \int_0^t \rho(t,\tau,v(\tau))d\tau \geq 1 \right\}.$$

Since the function $\rho(t,\tau,v)$ is $\Lambda \times \mathcal{B}$-measurable with respect to τ, v, it is superpositionally measurable [17, 3]. If $\xi(t) \in M(t)$, then $\rho(t,\tau,v) = +\infty$ for $\tau \in [0,t]$ and in this case it is natural to set the value of the integral in (1.63) to be equal $+\infty$. Then the inequality in (1.63) is fulfilled by default. In the case when the inequality in braces in (1.63) fails for all $t > 0$, we set $\mathfrak{T} = \emptyset$. Let $T \in \mathfrak{T} \neq \emptyset$.

Condition 1 [13] *The set $\Re(T,\tau,v)$ is convex-valued (or has values star-shaped with respect to the origin) for all $\tau \in [0,T]$, $v \in V$.*

Theorem 1 *Let for the game problem (1.29), (1.30) there exist a bounded measurable matrix function $C(t)$ such that the conditions (1.33), (1.34) hold true and the set M be convex. If there exists a finite number T, $T \in \mathfrak{T} \neq \emptyset$, such that the condition 1 is fulfilled, then the trajectory of the process (1.29) can be brought to the set (1.30) from the initial position z^0 at the time instant T using the control of the form (1.31).*

Proof. Let $v(\tau)$, $v : [0,T] \to V$, be an arbitrary measurable function. We first consider the case when $\xi(T) \notin M(T)$. Denote $\rho(T) = \int_0^T \inf\limits_{v\in V} \rho(T,\tau,v)d\tau$ and set:

$$\rho^*(T,\tau) = \frac{1}{\rho(T)} \inf_{v\in V} \rho(T,\tau,v).$$

Since $\rho(T) \geq 1$ due to (1.35) and Condition 1 is fulfilled, the function $\rho^*(T,\tau)$, $0 \leq \rho^*(T,\tau) \leq \rho(T,\tau,v)$, $\tau \in [0,T]$, $v \in V$, is a measurable selection for each of the set-valued mappings $\Re(T,\tau,v)$, $v \in V$, i.e. $\rho^*(T,\tau) \in \Re(T,\tau,v)$, $\tau \in [0,T]$, $v \in V$. Consider the multivalued mapping:

$$U(\tau,v) = \{u \in U :$$

$$\Pi(T-\tau)^{\alpha-1} E_{\frac{1}{\alpha}}(A(T-\tau)^\alpha;\alpha)\varphi(u, C(T-\tau)v) +$$

$$-\gamma(T-\tau) \in \rho^*(T,\tau)[M(T) - \xi(T)]\}. \tag{1.36}$$

Since the function $\rho^*(T,\tau)$ is measurable due to the assumptions made, $M(T) \in K(\mathbb{R}^m)$, as $M \in K(\mathbb{R}^m)$, and the vector $\xi(T)$ is bounded, it follows that the mapping $\rho^*(T,\tau)[M(T) - \xi(T)]$ is measurable with respect to τ. Moreover, the left-hand side of the inclusion in (1.36) is $\Lambda \times \mathcal{B}$-measurable with respect to (τ, v) and continuous in u. This implies [21] that the mapping $U(\tau, v)$ is $\Lambda \times \mathcal{B}$-measurable. Thus, according the theorem on measurable selection it contains an $\Lambda \times \mathcal{B}$-measurable selection $u(\tau, v)$, which, in turn, is a superpositionally measurable function. Set the first player's control to be $u(\tau) = u(\tau, v(\tau))$, $\tau \in [0, T]$.

In the case when $\xi(T) \in M(T)$ we construct the first player's control as follows. Let us set $\rho^*(T,\tau) \equiv 0$ in (1.36) and denote by $U_0(\tau, v)$ the set-valued mapping obtained in such a way from $U(\tau, v)$. Let us choose the first player's control in the form $u_0(\tau) = u_0(\tau, v(\tau))$, $\tau \in [0, T]$, where $u_0(\tau, v)$ is a measurable selection of the mapping $U_0(\tau, v)$.

Let us show that in each case treated above the trajectory of the process (1.29) hits the terminal set at the time instant T.

We have:

$$\Pi z(T) = \Pi g(T, z^0) + \int_0^T (T-\tau)^{\alpha-1} \Pi E_{\frac{1}{\alpha}}(A(T-\tau)^\alpha; \alpha)\varphi(u(\tau), v(\tau))d\tau. \tag{1.37}$$

Consider the case $\xi(T) \notin M(T)$. Let us add and subtract from the right-hand side of (1.37) the following vectors:

$$\begin{array}{c} \int_0^T (T-\tau)^{\alpha-1} \Pi E_{\frac{1}{\alpha}}(A(T-\tau)^\alpha; \alpha)\varphi(u(\tau), C(T-\tau)v(\tau))d\tau, \\ \int_0^t \gamma(T-\tau)d\tau. \end{array} \tag{1.38}$$

Taking into account the control rule of the first player, we obtain from (1.37) the following inclusion:

$$\Pi z(T) \in \xi(T)\left[1 - \int_0^T \rho^*(T,\tau)d\tau\right] + \int_0^T \rho^*(T,\tau)M(T)d\tau +$$

$$+ \int_0^T (T-\tau)^{\alpha-1} \Pi E_{\frac{1}{\alpha}}(A(T-\tau)^\alpha; \alpha)\varphi^*(T-\tau, u(\tau), v(\tau))d\tau.$$

Since $M(T)$ is a convex compact set, as $M \in coK(\mathbb{R}^m)$, and $\rho^*(T,\tau)$ is a

non-negative function and $\int_0^T \rho^*(T,\tau)d\tau = 1$, it follows that:

$$\int_0^T \rho^*(T,\tau)M(T)d\tau = M(T)\,;$$

hence:

$$\Pi z(T) \in M(T) + \int_0^T (T-\tau)^{\alpha-1}\Pi E_{\frac{1}{\alpha}}(A(T-\tau)^{\alpha};\alpha)\varphi^*(T-\tau,u(\tau),v(\tau))d\tau.$$

From which, taking into account the definition of the Minkowski subtraction and the form of the set $M(T)$, follows the inclusion $\Pi z(T) \in M$.

Now assume $\xi(T) \in M(T)$. Adding and subtracting from the right-hand side of (1.37) the vectors (1.38) and taking into account the first player's control rule we obtain:

$$\Pi z(T) = \xi(T) + \int_0^T (T-\tau)^{\alpha-1}\Pi E_{\frac{1}{\alpha}}(A(T-\tau)^{\alpha};\alpha)\varphi^*(T-\tau,u(\tau),v(\tau))d\tau \in$$

$$\in M(T) + \int_0^T (T-\tau)^{\alpha-1}\Pi E_{\frac{1}{\alpha}}(A(T-\tau)^{\alpha};\alpha)\varphi^*(T-\tau,u(\tau),v(\tau))d\tau.$$

Which implies the inclusion $\Pi z(T) \in M$ or $z(T) \in M^*$.

1.9 Comparison with Pontryagin's first direct method

As mentioned in [34], the singular case when the resolving function becomes infinite is closely related to the Pontryagin first direct method. Let us state the result more precisely. Let us introduce the function:

$$P(z^0) = \min\left\{t \geq 0 :\ g(t,z^0) \in M(t) - \int_0^t W^*(\tau)d\tau\right\}.$$

Theorem 2 *Suppose that in game (1.29)–(1.30) there exists a measurable bounded matrix function $C(t)$ such that the modified Pontryagin condition holds. Then the trajectory of process (1.29) can be brought to the set (1.30) from the initial position z^0 at the moment $P(z^0)$ using the control of the form $u(t) = u(z^0, v(t))$.*

The proof is similar to the proof of Theorem 1.

Assertion 1 *Suppose that for conflict-controlled process (1.29)-(1.30) with a bounded measurable matrix function $C(t)$ the modified Pontryagin condition holds. Then for the inclusion:*

$$g(t, z^0) \in M(t) - \int_0^t W^*(\tau) d\tau \tag{1.39}$$

to be true, it is necessary and sufficient that there exists a summable in τ, $\tau \in [0, t]$, selection $\gamma(\tau)$ of the set-valued map $W^*(\tau)$ such that:

$$\xi(t) \in M(t). \tag{1.40}$$

The proof of this assertion follows from the definition of the function $\xi(t)$ and integral of a set-valued map.

This directly implies that there exists a measurable selection $\gamma(t)$, $\gamma(t) \in W^*(t)$, such that:

$$T(z^0, \gamma(\cdot)) \leq P(z^0) \quad \forall z^0,$$

where $T(z^0, \gamma(\cdot)) = \inf \mathfrak{T}$.

Remark 1*Inclusion (1.40) or (1.39) directly implies that the function $\rho(t, \tau, v)$ becomes infinite for all $\tau \in [0, t]$, $v \in V$.*

1.10 Separate Dynamics

Let us consider the case when the dynamics of each player is described by a separate fractional differential equation. Suppose that the motion of the first player hereafter referred to as the pursuer is described by the equation:

$$\mathbf{D}^\alpha x = Ax + u, \quad x \in \mathbb{R}^{m_1}, \quad n_1 - 1 < \alpha < n_1. \tag{1.41}$$

Dynamics of the second player whom we will refer to as the evader, is given by the equation:

$$\mathbf{D}^\beta y = By + v, \quad y \in \mathbb{R}^{m_2}, \quad n_2 - 1 < \beta < n_2. \tag{1.42}$$

Here A and B are square matrices of order m_1 and m_2, respectively, $U \in K(\mathbb{R}^{m_1})$, $V \in K(\mathbb{R}^{m_2})$. Let us note that the system (1.41), (1.42) is not a particular case of the process (1.29), since the numbers α and β are arbitrary. It is assumed that the initial conditions for the systems (1.41), (1.42) are given in the form corresponding to the operators $\mathbf{D}^\alpha$, $\mathbf{D}^\beta$ and defined by the initial state vectors x^0, y^0 of the pursuer and evader, respectively.

The terminal set is defined by the ε-distance by the first s (where $s \leq \min(m_1, m_2)$) components of the vectors x and y, i.e. the game is said to be terminated as soon as:

$$\|x - y\|_s \leq \varepsilon. \tag{1.43}$$

Here ε is a fixed number, $0 \leq \varepsilon < \infty$.

Let us introduce orthoprojectors $\Pi_1 : \mathbb{R}^{m_1} \to \mathbb{R}^s$, $\Pi_2 : \mathbb{R}^{m_2} \to \mathbb{R}^s$, which keep the first s coordinates in the vectors x, y, respectively, and discard the other coordinates. Then the inequality (1.43) can be rewritten in the form:

$$\|\Pi_1 x - \Pi_2 y\| \leq \varepsilon. \tag{1.44}$$

We will refer to the situation when the inequality (1.43) or equivalently (1.44) holds true as a capture.

In virtue of Lemmas 4-7, the trajectories of systems (1.41), (1.42) are of the form:

$$x(t) = g_x(t, x^0) + \int_0^t (t-\tau)^{\alpha-1} E_{\frac{1}{\alpha}}(A(t-\tau)^\alpha; \alpha) u(\tau) d\tau,$$

$$y(t) = g_y(t, y^0) + \int_0^t (t-\tau)^{\beta-1} E_{\frac{1}{\beta}}(B(t-\tau)^\beta; \beta) v(\tau) d\tau,$$

where $g_x(t, x^0)$ and $g_y(t, y^0)$ are general solutions to the homogeneous systems $D^\alpha x = Ax$, $D^\beta y = By$, respectively, with initial positions x^0, y^0.

Following the scheme of the method of resolving functions, let us consider the set-valued maps:

$$W_1(t, v) = t^{\alpha-1} \Pi_1 E_{\frac{1}{\alpha}}(At^\alpha; \alpha) U - t^{\beta-1} \Pi_2 E_{\frac{1}{\beta}}(Bt^\beta; \beta) C_1(t) v;$$

$$W_1(t) = t^{\alpha-1} \Pi_1 E_{\frac{1}{\alpha}}(At^\alpha; \alpha) U \overset{*}{-} t^{\beta-1} \Pi_2 E_{\frac{1}{\beta}}(Bt^\beta; \beta) C_1(t) V. \tag{1.45}$$

Here $C_1(t)$ is a matrix function to equalize the control resources. Along with the set-valued map $W_1(t)$ the modified Pontryagin condition involves the map:

$$M_1(t) = \varepsilon S \overset{*}{-\!-} \int_0^t \tau^{\beta-1} \Pi_2 E_{\frac{1}{\beta}}(B\tau^{\beta};\beta)(C_1(\tau) - I)V d\tau, \tag{1.46}$$

where S is a closed ball of the unit radius centered at the origin.

The modified Pontryagin condition is fulfilled if for some measurable bounded function $C_1(t)$ the set-valued maps defined by (1.45), (1.46) are nonempty for all $t \geq 0$.

Let us choose a measurable selection $\gamma_1(t)$ ($\gamma_1(t) \in W_1(t)\ \forall t \geq 0$) in $W_1(t)$ and set:

$$\xi_1(t) = \Pi_1 g_x(t, x^0) - \Pi_2 g_y(t, y^0) + \int_0^t \gamma_1(\tau) d\tau.$$

As before, let us introduce the set-valued map:

$$\Re_1(t,\tau,v) = \{\rho \geq 0 : [W_1(t-\tau) - \gamma_1(t-\tau)] \bigcap \rho[M_1(t) - \xi(t)] \neq \emptyset\},$$
$$\Re_1 : \Delta \times V \to 2^{\mathbb{R}_+},$$

and its support function in the direction $+1$ (resolving function):

$$\rho_1(t,\tau,v) = \sup\{\rho :\ \rho \in \Re_1\}, \quad \rho_1 : \Delta \times V \to \mathbb{R}_+.$$

On the basis of the resolving function, let us define the set:

$$\mathfrak{T}_1 = \left\{ t \geq 0 : \int_0^t \inf_{v \in V} \rho_1(t,\tau,v) d\tau \geq 1 \right\}.$$

Let $T \in \mathfrak{T}_1 \neq \emptyset$.

Condition 2 *The map $\Re_1(T,\tau,v)$ is convex-valued for all $\tau \in [0,T]$, $v \in V$.*

Theorem 3 *Let for the game (1.41)-(1.44) with separated dynamics of the players there exist a bounded measurable matrix function $C_1(t)$, $t \geq 0$, such that the set-valued maps $W_1(t)$ and $M_1(t)$ are nonempty-valued for all $t \geq 0$. If there exists a finite number T, $T \in \mathfrak{T}_1 \neq \emptyset$, such that Condition 2 is fulfilled, then the capture in game (1.41)-(1.44) occurs at the moment T.*

The proof is similar to that of Theorem 1 taking into account the specific character of the problem (1.41)-(1.44).

1.11 Example of a Pursuit Game for Systems of Fractional Order π and e

Here we consider an example of fractional order pursuit-evasion dynamic game. Let the dynamics of the first player whom we will refer to as *Pursuer* and denote by P be described by the equation:

$$\mathbf{D}^{\pi}x = u, \quad |u| \leq 1, \tag{1.47}$$

where $\pi = 3,14159\ldots$ is the ratio of a circle's circumference to its diameter.

The dynamics of the second player whom we will refer to as *Evader* and denote E,is governed by the equation:

$$\mathbf{D}^{e}y = v, \quad |v| \leq 1, \tag{1.48}$$

where $e = 2,71828\ldots$ is the base of the natural logarithm.

Here as before $\mathbf{D}^{\alpha}$ is the operator of fractional differentiation of order α in the sense of Riemann-Liouville, Caputo, Miller-Ross, or Hilfer. The phase vectors x and y define the current position in $\mathbb{R}^m$ of the pursuer and the evader, respectively. We suppose that $x = x(t)$ is triple and $y = y(t)$ is twice absolutely continuously differentiable on $\mathbb{R}_+$ functions of time t, i.e. $x(t) \in AC^3(\mathbb{R}_+)$, $y(t) \in AC^2(\mathbb{R}_+)$. The control vectors $u = u(t)$, $v = v(t)$, $u, v \in \mathbb{R}^m$ are measurable functions of time t.

Since A and B are $m \times m$ zero matrices, $E_{\frac{1}{\pi}}(At^{\pi};\pi) = \frac{1}{\Gamma(\pi)}I$ and $E_{\frac{1}{e}}(Bt^{e};e) = \frac{1}{\Gamma(e)}I$.

If the differentiation is taken in the sense of Riemann-Liouville, i.e. $\mathbf{D}^{\alpha} = D^{\alpha}$, the initial conditions for (1.47), (1.48) are of the form:

$$D^{\pi-1}x(t)|_{t=0} = x_{11}^0,\ D^{\pi-2}x(t)|_{t=0} = x_{21}^0,$$

$$D^{\pi-3}x(t)|_{t=0} = x_{31}^0,\ D^{\pi-4}x(t)|_{t=0} = x_{41}^0$$

and:

$$D^{e-1}y(t)|_{t=0} = y_{11}^0,\ D^{e-2}y(t)|_{t=0} = y_{21}^0,\ D^{e-3}y(t)|_{t=0} = y_{31}^0,$$

respectively. In this case we denote:

$$x^0 = (x_{11}^0, x_{21}^0, x_{31}^0, x_{41}^0),\ y^0 = (y_{11}^0, y_{21}^0, y_{31}^0),$$

$$g_x(t,x^0)=\frac{t^{\pi-1}}{\Gamma(\pi)}x_{11}^0+\frac{t^{\pi-2}}{\Gamma(\pi-1)}x_{21}^0+\frac{t^{\pi-3}}{\Gamma(\pi-2)}x_{31}^0+\frac{t^{\pi-4}}{\Gamma(\pi-3)}x_{41}^0,$$

$$g_y(t,y^0)=\frac{t^{e-1}}{\Gamma(e)}y_{11}^0+\frac{t^{e-2}}{\Gamma(e-1)}y_{21}^0+\frac{t^{e-3}}{\Gamma(e-2)}y_{31}^0.$$

Now suppose that $\mathbf{D}^\alpha$ stands for the operator of fractional differentiation in the sense of Caputo, i.e. $\mathbf{D}^\alpha=D^{(\alpha)}$. Then the initial conditions for (1.47), (1.48) can be written down the form:

$$x(0)=x_{02}^0,\ \dot{x}(0)=x_{12}^0,\ \ddot{x}(0)=x_{22}^0,\ \dddot{x}(0)=x_{32}^0$$

and:

$$y(0)=y_{02}^0,\ \dot{y}(0)=y_{12}^0,\ \ddot{y}(0)=y_{22}^0,$$

respectively. We denote:

$$x^0=(x_{02}^0,x_{12}^0,x_{22}^0,x_{32}^0),\ y^0=(y_{02}^0,y_{12}^0,y_{22}^0),$$

$$g_x(t,x^0)=x_{02}^0+tx_{12}^0+\frac{t^2}{2}x_{22}^0+\frac{t^3}{6}x_{32}^0,$$

$$g_y(t,y^0)=y_{02}^0+ty_{12}^0+\frac{t^2}{2}y_{22}^0.$$

If $\mathbf{D}^\pi=\mathsf{D}_i^\pi$ for some i, $i\in\{1,2,3\}$, and $\mathbf{D}^e=\mathsf{D}_j^e$, $j\in\{1,2\}$, the initial conditions for (1.47), (1.48) take on the form:

$$\mathsf{D}_{i-k-1}^{\pi-k-1}x(t)|_{t=0}=\tilde{x}_k^0,\quad k=0,\ldots,i-1,$$

$$x^{(l)}(0)=x_l^0,\quad l=0,\ldots,3-i,$$
$$\mathsf{D}_{j-r-1}^{e-r-1}y(t)|_{t=0}=\tilde{y}_r^0,\quad r=0,\ldots,j-1,$$
$$y^{(s)}(0)=y_s^0,\quad s=0,\ldots,2-j,$$

and:

$$x^0=(\tilde{x}_0^0,\ldots,\tilde{x}_{i-1}^0,x_0^0,\ldots,x_{3-i}^0),\ y^0=(\tilde{y}_0^0,\ldots,\tilde{y}_{j-1}^0,y_0^0,\ldots,y_{2-j}^0),$$

$$g_x(t,x^0)=\sum_{l=0}^{3-i}\frac{t^l}{l!}x_l^0+\sum_{k=0}^{i-1}\frac{t^{\pi-k-1}}{\Gamma(\pi-k)}\tilde{x}_k^0,$$

$$g_y(t,y^0)=\sum_{s=0}^{2-j}\frac{t^s}{s!}y_s^0+\sum_{r=0}^{j-1}\frac{t^{e-r-1}}{\Gamma(e-r)}\tilde{y}_r^0.$$

Finally, suppose that $\mathbf{D}^{\pi} = D^{\pi,\mu_1}$ and $\mathbf{D}^{e} = D^{e,\mu_2}$ for some $\mu_1, \mu_2 \in [0,1]$. Then the initial conditions are of the form:

$$\hat{x}_0^0 = J^{(1-\mu_1)(4-\pi)}x(t)|_{t=0+}, \hat{x}_1^0 = \frac{d}{dt}J^{(1-\mu_1)(4-\pi)}x(t)|_{t=0+},$$

$$\hat{x}_2^0 = \frac{d^2}{dt^2}J^{(1-\mu_1)(4-\pi)}x(t)|_{t=0+}, \hat{x}_3^0 = \frac{d^3}{dt^3}J^{(1-\mu_1)(4-\pi)}x(t)|_{t=0+},$$

$$\hat{y}_0^0 = J^{(1-\mu_2)(3-e)}y(t)|_{t=0+}, \hat{y}_1^0 = \frac{d}{dt}J^{(1-\mu_2)(3-e)}y(t)|_{t=0+},$$

$$\hat{y}_2^0 = \frac{d^2}{dt^2}J^{(1-\mu_2)(3-e)}y(t)|_{t=0+},$$

and we have:

$$x^0 = (\hat{x}_0, \hat{x}_1, \hat{x}_2, \hat{x}_3),\ y^0 = (\hat{y}_0, \hat{y}_1, \hat{y}_2),$$

$$g_x(t, x^0) = \frac{t^{-(1-\mu_1)(4-\pi)}}{\Gamma(1-(1-\mu_1)(4-\pi))}\hat{x}_0 + \frac{t^{1-(1-\mu_1)(4-\pi)}}{\Gamma(2-(1-\mu_1)(4-\pi))}\hat{x}_1 +$$
$$+\frac{t^{2-(1-\mu_1)(4-\pi)}}{\Gamma(3-(1-\mu_1)(4-\pi))}\hat{x}_2 + \frac{t^{3-(1-\mu_1)(4-\pi)}}{\Gamma(4-(1-\mu_1)(4-\pi))}\hat{x}_3,$$

$$g_y(t, y^0) = \frac{t^{-(1-\mu_2)(3-e)}}{\Gamma(1-(1-\mu_2)(3-e))}\hat{y}_0 + \frac{t^{1-(1-\mu_2)(3-e)}}{\Gamma(2-(1-\mu_2)(3-e))}\hat{y}_1 +$$
$$+\frac{t^{2-(1-\mu_2)(3-e)}}{\Gamma(3-(1-\mu_2)(3-e))}\hat{y}_2.$$

The goal of the pursuer is to achieve the fulfillment of the inequality:

$$\|x(T) - y(T)\| \leq \varepsilon, \quad \varepsilon > 0, \tag{1.49}$$

for some finite time instant T. The goal of the evader is to prevent the fulfillment of the inequality (1.49) or, provided it is impossible, to maximally postpone the time instant T.

The use of the Method of Resolving Functions described above makes it possible to derive sufficient condition for the solvability of the formulated pursuit problem.

Here $\Pi_1 = \Pi_2 = I$. The set-valued mappings (1.45) for this example take on the form:

$$W_1(t, v) = \frac{t^{\pi-1}}{\Gamma(\pi)}U - \frac{t^{e-1}}{\Gamma(e)}C_1(t)v = \frac{t^{\pi-1}}{\Gamma(\pi)}S - \frac{t^{e-1}}{\Gamma(e)}C_1(t)v,$$

$$W_1(t) = \bigcap_{v \in V} W_1(t,v) = \frac{t^{\pi-1}}{\Gamma(\pi)} S \overset{*}{-\!-} \frac{t^{e-1}}{\Gamma(e)} C_1(t) S,$$

where the matrix function $C_1(t)$ equalizing control resources can be chosen in the form:

$$C_1(t) = c_1(t) I,$$

$$c_1(t) = \begin{cases} \frac{\Gamma(e)}{\Gamma(\pi)} t^{\pi-e} & \text{if} \quad 0 \le t < \left(\frac{\Gamma(\pi)}{\Gamma(e)}\right)^{\frac{1}{\pi-e}} \\ 1 & \text{if} \quad t \ge \left(\frac{\Gamma(\pi)}{\Gamma(e)}\right)^{\frac{1}{\pi-e}} \end{cases} \tag{1.50}$$

Then:

$$W_1(t) = \begin{cases} \{0\} & \text{if} \quad 0 \le t < \left(\frac{\Gamma(\pi)}{\Gamma(e)}\right)^{\frac{1}{\pi-e}} \\ \left(\frac{t^{\pi-1}}{\Gamma(\pi)} - \frac{t^{e-1}}{\Gamma(e)}\right) S & \text{if} \quad t \ge \left(\frac{\Gamma(\pi)}{\Gamma(e)}\right)^{\frac{1}{\pi-e}} \end{cases}$$

So, $W_1(t) \neq \emptyset$ for all $t \ge 0$. Further,

$$M_1(t) = \varepsilon S \overset{*}{-} \int_0^t \frac{\tau^{e-1}}{\Gamma(e)} (c_1(\tau) - 1) V d\tau$$

$$= \begin{cases} \left[\varepsilon - \left|\frac{t^\pi}{\Gamma(\pi+1)} - \frac{t^e}{\Gamma(e+1)}\right|\right] S & \text{if} \quad 0 \le t < \left(\frac{\Gamma(\pi)}{\Gamma(e)}\right)^{\frac{1}{\pi-e}} \\ \left[\varepsilon - \left|\frac{(\Gamma(\pi)/\Gamma(e))^{\frac{\pi}{\pi-e}}}{\Gamma(\pi+1)} - \frac{(\Gamma(\pi)/\Gamma(e))^{\frac{e}{\pi-e}}}{\Gamma(e+1)}\right|\right] S & \text{if} \quad t \ge \left(\frac{\Gamma(\pi)}{\Gamma(e)}\right)^{\frac{1}{\pi-e}} \end{cases}$$

Thus, the modified Pontryagin condition holds if:

$$\varepsilon \ge \left| \frac{(\Gamma(\pi)/\Gamma(e))^{\frac{\pi}{\pi-e}}}{\Gamma(\pi+1)} - \frac{(\Gamma(\pi)/\Gamma(e))^{\frac{e}{\pi-e}}}{\Gamma(e+1)} \right|$$

$$= \frac{(\Gamma(\pi)/\Gamma(e))^{\frac{e}{\pi-e}}}{\Gamma(e+1)} - \frac{(\Gamma(\pi)/\Gamma(e))^{\frac{\pi}{\pi-e}}}{\Gamma(\pi+1)} \approx 0.358787.$$

Let us choose $\gamma_1(t) \equiv 0$ as a measurable selection of $W_1(t)$. Then:

$$\xi_1(t) = \Pi_1 g_x(t, x^0) - \Pi_2 g_y(t, y^0).$$

Denote:

$$m(t) = \begin{cases} \varepsilon - \left|\frac{t^\pi}{\Gamma(\pi+1)} - \frac{t^e}{\Gamma(e+1)}\right| & \text{if} \quad 0 \le t < \left(\frac{\Gamma(\pi)}{\Gamma(e)}\right)^{\frac{1}{\pi-e}} \\ \varepsilon - \left|\frac{(\Gamma(\pi)/\Gamma(e))^{\frac{\pi}{\pi-e}}}{\Gamma(\pi+1)} - \frac{(\Gamma(\pi)/\Gamma(e))^{\frac{e}{\pi-e}}}{\Gamma(e+1)}\right| & \text{if} \quad t \ge \left(\frac{\Gamma(\pi)}{\Gamma(e)}\right)^{\frac{1}{\pi-e}} \end{cases}$$

then:

$$M_1(t) = m(t)S.$$

Consider resolving function:

$$\rho_1(t,\tau,v) = \sup\{\rho \geq 0 : W_1(t-\tau,v) \bigcap \rho[M_1(t) - \xi_1(t)] \neq \emptyset\}.$$

This function can be evaluated as the greater root of the quadratic equation in ρ:

$$\left\| \rho\xi_1(t) - \frac{(t-\tau)^{e-1}c_1(t-\tau)}{\Gamma(e)}v \right\| = \frac{(t-\tau)^{\pi-1}}{\Gamma(\pi)} + \rho m(t).$$

Solving this equation, we find:

$$\rho_1(t,\tau,v) = \frac{\frac{(t-\tau)^{e-1}c_1(t-\tau)}{\Gamma(e)}(\xi_1(t),v) + \frac{(t-\tau)^{\pi-1}}{\Gamma(\pi)}m(t) + \sqrt{\Delta_1}}{\|\xi_1(t)\|^2 - m^2(t)},$$

where:

$$\Delta_1 = \left(\frac{(t-\tau)^{e-1}c_1(t-\tau)}{\Gamma(e)}(\xi_1(t),v) + \frac{(t-\tau)^{\pi-1}}{\Gamma(\pi)}m(t) \right)^2$$

$$+ \left(\frac{(t-\tau)^{2\pi-2}}{\Gamma^2(\pi)} - \frac{(t-\tau)^{2e-2}c_1^2(t-\tau)}{\Gamma^2(e)}\|v\|^2 \right) \left(\|\xi_1(t)\|^2 - m^2(t) \right).$$

At $v = -\frac{\xi_1(t)}{\|\xi_1(t)\|}$ the minimum value is attained:

$$\min_{\|v\|\leq 1} \rho_1(t,\tau,v) = \frac{\frac{(t-\tau)^{\pi-1}}{\Gamma(\pi)} - \frac{(t-\tau)^{e-1}c_1(t-\tau)}{\Gamma(e)}}{\|\xi_1(t)\| - m(t)}. \tag{1.51}$$

In virtue of continuity of the numerator and denominator in (1.51), time of (1.47)–(1.49) game termination can be found as the least positive root of the equation:

$$\int_0^t \left[\frac{\tau^{\pi-1}}{\Gamma(\pi)} - \frac{\tau^{e-1}}{\Gamma(e)}c_1(\tau) \right] d\tau = \|\xi_1(t)\| - m(t). \tag{1.52}$$

The equation (1.52) can be simplified taking into account form of the functions $c_1(t)$ and $m(t)$. Indeed, it follows from (1.50) that for $0 \leq t < \left(\frac{\Gamma(\pi)}{\Gamma(e)}\right)^{\frac{1}{\pi-e}}$

the integrand is equal to zero and the game cannot be terminated on this interval.

Suppose $t \geq \left(\frac{\Gamma(\pi)}{\Gamma(e)}\right)^{\frac{1}{\pi-e}}$. Then:

$$\int_0^t \left[\frac{\tau^{\pi-1}}{\Gamma(\pi)} - \frac{\tau^{e-1}}{\Gamma(e)}c_1(\tau)\right] d\tau = \int_{\left(\frac{\Gamma(\pi)}{\Gamma(e)}\right)^{\frac{1}{\pi-e}}}^t \left[\frac{\tau^{\pi-1}}{\Gamma(\pi)} - \frac{\tau^{e-1}}{\Gamma(e)}\right] d\tau\,,$$

and equation (1.52) for obtaining the time of game termination takes on its final form:

$$\frac{t^\pi}{\Gamma(\pi+1)} - \frac{t^e}{\Gamma(e+1)} + \varepsilon = \|\xi_1(t)\|.$$

This equation always has a positive solution since its left-hand side increases at the rate of $O(t^\pi)$, while the growth rate of the right-hand side is not greater than $O(t^3)$.

1.12 Game with Plain Matrix. Asymptotic Representation of the Scalar. Mittag-Leffler Functions

Consider a conflict-controlled process, evolution of which is defined by the system of fractional order in the sense of Caputo:

$$D^{(\alpha)}z = \lambda z + u - v,\ z \in \mathbb{R}^m,\ u \in aS,\ v \in S,\ a > 1, \tag{1.53}$$

where λ is a real number, $n-1 < \alpha \leq n$, with initial conditions:

$$z^{(i)}(0) = z_i^0, \quad i = 0, \ldots, n-1, \tag{1.54}$$

and terminal set $M^* = \varepsilon S$, $\varepsilon \geq 0$. Obviously, here $A = \lambda I$, $\varphi(u,v) = u - v$, $U = aS$, $V = S$, $M_0 = \{0\}$, and $M = \varepsilon S$. Therefore, $L = M_0^\perp = \mathbb{R}^m$ and the orthoprojector Π is an identity operator. Since for the matrix $A = \lambda I$:

$$E_{1/\alpha}(A;\mu) = E_{1/\alpha}(\lambda;\mu)I,$$

solution to the Cauchy problem for the system (1.53) with the initial conditions (1.54) has the form (1.22):

$$z(t) = \sum_{i=0}^{n-1} t^i E_{\frac{1}{\alpha}}(\lambda t^\alpha; i+1)z_i^0 + \int_0^t (t-\tau)^{\alpha-1} E_{\frac{1}{\alpha}}(\lambda(t-\tau)^\alpha;\alpha)(u(\tau)-v(\tau))d\tau.$$

According to the Method of Resolving Functions, let us check fulfillment of Pontryagin's condition:

$$W(t,v) = t^{\alpha-1}E_{\frac{1}{\alpha}}(\lambda t^{\alpha};\alpha)(aS-v), \quad W(t) = \left|t^{\alpha-1}E_{\frac{1}{\alpha}}(\lambda t^{\alpha};\alpha)\right|(a-1)S,$$

hence the Pontryagin condition holds true. Let us set $\gamma(t) \equiv 0$, then:

$$\xi(t) = \sum_{i=0}^{n-1} t^i E_{\frac{1}{\alpha}}(\lambda t^{\alpha}; i+1) z_i^0.$$

The resolving function has the form:

$$\rho(t,\tau,v) = \sup\{\rho \geq 0 :$$

$$(t-\tau)^{\alpha-1}E_{\frac{1}{\alpha}}(\lambda(t-\tau)^{\alpha};\alpha)(aS-v)$$

$$\bigcap \rho[\varepsilon S - \xi(t)] \neq \emptyset\}. \tag{1.55}$$

Denote $w(t) = t^{\alpha-1}E_{\frac{1}{\alpha}}(\lambda t^{\alpha};\alpha)$. The resolving function (1.55) can be found explicitly as the larger root of the quadratic equation in ρ:

$$\|w(t-\tau)v - \rho\xi(t)\| = aw(t-\tau) + \rho\varepsilon,$$

which implies:

$$\rho(t,\tau,v) = \frac{w(t-\tau)[(\xi(t),v) + a\varepsilon] + |w(t-\tau)|\sqrt{\Delta_2}}{\|\xi(t)\|^2 - \varepsilon^2},$$

here $\Delta_2 = ((\xi(t),v) + a\varepsilon)^2 - (|v|^2 - a^2)(\|\xi(t)\|^2 - \varepsilon^2)$. The resolving function $\rho(t,\tau,v)$ attains its minimum value:

$$\min_{\|v\|\leq 1} \rho(t,\tau,v) = w(t-\tau)\frac{a-1}{\|\xi(t)\| - \varepsilon}$$

at $v = -\frac{\xi(t)}{\|\xi(t)\|}$. Then the time of game termination is determined as the smallest positive root of the equation:

$$(a-1)\int_0^t (t-\tau)^{\alpha-1}E_{\frac{1}{\alpha}}(\lambda(t-\tau)^{\alpha};\alpha)d\tau = \left\|\sum_{i=0}^{n-1} t^i E_{\frac{1}{\alpha}}(\lambda t^{\alpha}; i+1) z_i^0\right\| - \varepsilon. \tag{1.56}$$

Taking into account:

$$\int_0^t (t-\tau)^{\alpha-1} E_{\frac{1}{\alpha}}(\lambda(t-\tau)^\alpha;\alpha)d\tau = t^\alpha E_{\frac{1}{\alpha}}(\lambda t^\alpha;\alpha+1),$$

we obtain the equation:

$$t^\alpha E_{\frac{1}{\alpha}}(\lambda t^\alpha;\alpha+1) = \frac{\left\|\sum_{i=0}^{n-1} t^i E_{\frac{1}{\alpha}}(\lambda t^\alpha; i+1) z_i^0\right\| - \varepsilon}{a-1}$$

for determination of termination time for the game (1.53), (1.54).

It is natural to assume that for $t=0$ the vector $z_0^0 = z(0)$ does not belong to the ball εS. In this case at $t=0$ the left-hand side of (1.56) equals zero and the right-hand side is positive.

Let us investigate the rate of growth of each side of the equation (1.56) as $t \to \infty$. If $\alpha < 2$ for any μ, $\lambda > 0$, and $p \in \mathbb{N}$ we have:

$$E_{\frac{1}{\alpha}}(\lambda t^\alpha;\mu) = \frac{1}{\alpha}(\lambda t^\alpha)^{\frac{1-\mu}{\alpha}} e^{(\lambda t^\alpha)^{\frac{1}{\alpha}}} - \sum_{j=1}^{p} \frac{(\lambda t^\alpha)^{-j}}{\Gamma(\mu - j\alpha)} + O((t^\alpha)^{-1-p}).$$

Applying this asymptotic formula, we obtain:

$$t^\alpha E_{\frac{1}{\alpha}}(\lambda t^\alpha;\alpha+1) = \frac{1}{\alpha}\lambda^{-1} e^{\lambda^{\frac{1}{\alpha}} t} - \lambda^{-1} + \ldots,$$

$$t^i E_{\frac{1}{\alpha}}(\lambda t^\alpha; i+1) = \frac{1}{\alpha}\lambda^{-\frac{i}{\alpha}} e^{\lambda^{\frac{1}{\alpha}} t} - \sum_{j=1}^{\frac{i}{\alpha}} \frac{\lambda^{-j} t^{i-j\alpha}}{\Gamma(i+1-j\alpha)} + \ldots$$

Hence,

$$\lim_{t\to\infty} \frac{t^\alpha E_{\frac{1}{\alpha}}(\lambda t^\alpha;\alpha+1)}{\left\|\sum_{i=0}^{n-1} t^i E_{\frac{1}{\alpha}}(\lambda t^\alpha; i+1) z_i^0\right\| - \varepsilon} = \frac{\frac{1}{\alpha}\lambda^{-1}}{\left\|\sum_{i=0}^{n-1} \frac{1}{\alpha}\lambda^{-\frac{i}{\alpha}} z_i^0\right\|} = \frac{1}{\left\|\sum_{i=0}^{n-1} \lambda^{1-\frac{i}{\alpha}} z_i^0\right\|}.$$

Therefore, the root of the equation (1.56) exists if:

$$a - 1 > \left\|\sum_{i=0}^{n-1} \lambda^{1-\frac{i}{\alpha}} z_i^0\right\|. \tag{1.57}$$

Now consider the case when $\lambda < 0$, $\alpha < 2$, and μ is arbitrary. Then for any natural p:

$$E_{\frac{1}{\alpha}}(\lambda t^\alpha; \mu) = -\sum_{j=1}^{p} \frac{(\lambda t^\alpha)^{-j}}{\Gamma(\mu - j\alpha)} + O((t^\alpha)^{-1-p}).$$

Applying this asymptotic representation, we get:

$$t^\alpha E_{\frac{1}{\alpha}}(\lambda t^\alpha; \alpha + 1) = -\lambda^{-1} - \frac{\lambda^{-2} t^{-\alpha}}{\Gamma(1-\alpha)} + \dots,$$

$$E_{\frac{1}{\alpha}}(\lambda t^\alpha; 1) = -\frac{\lambda^{-1} t^{-\alpha}}{\Gamma(1-\alpha)} + \dots,$$

$$tE_{\frac{1}{\alpha}}(\lambda t^\alpha; 2) = -\sum_{j=1}^{\frac{1}{\alpha}} \frac{\lambda^{-j} t^{1-j\alpha}}{\Gamma(2-j\alpha)} + \dots,$$

so,

$$\lim_{t\to\infty} t^\alpha E_{\frac{1}{\alpha}}(\lambda t^\alpha; \alpha + 1) = -\lambda^{-1},$$

$$\lim_{t\to\infty} \left\| \sum_{i=0}^{n-1} t^i E_{\frac{1}{\alpha}}(\lambda t^\alpha; i+1) z_i^0 \right\| = 0.$$

Thus, equation (1.56) has a finite positive root for any initial conditions.

If $\alpha \geq 2$, the following asymptotic formulas are to be applied:

$$\begin{aligned} E_{\frac{1}{\alpha}}(\lambda t^\alpha; \mu) = \; & \tfrac{1}{\alpha} \lambda^{\frac{1-\mu}{\alpha}} t^{1-\mu} \textstyle\sum_{|\arg\lambda + 2\pi n| \leq \frac{\alpha\pi}{2}} e^{\frac{2\pi i n(1-\mu)}{\alpha}} \exp(e^{\frac{2\pi i n(1-\mu)}{\alpha}} \lambda^{\frac{1}{\alpha}} t) + \\ & - \textstyle\sum_{j=1}^{p} \frac{(\lambda t^\alpha)^{-j}}{\Gamma(\mu - j\alpha)} + O((t^\alpha)^{-1-p}), \end{aligned} \tag{1.58}$$

where $p \geq 1$ is an arbitrary integer and the first sum is taken over $n = 0, \pm 1, \pm 2, \dots$ such that $|\arg\lambda + 2\pi n| \leq \frac{\alpha\pi}{2}$.

Now suppose that $\lambda > 0$ and $\alpha \in [2, 4)$. Then $\arg\lambda = 0$ and the first sum in (1.58) consists of a single term that corresponds to $n = 0$. Hence, for sufficiently large values of t we have:

$$E_{\frac{1}{\alpha}}(\lambda t^\alpha; \mu) = \frac{1}{\alpha} \lambda^{\frac{1-\mu}{\alpha}} t^{1-\mu} e^{\lambda^{\frac{1}{\alpha}} t} + \dots,$$

whence:

$$E_{\frac{1}{\alpha}}(\lambda t^\alpha; \alpha + 1) = \frac{1}{\alpha} \lambda^{-1} t^{-\alpha} e^{\lambda^{\frac{1}{\alpha}} t} + \dots,$$

$$t^{\alpha}E_{\frac{1}{\alpha}}(\lambda t^{\alpha};\alpha+1)=\frac{1}{\alpha}\lambda^{-1}e^{\lambda^{\frac{1}{\alpha}}t}+\ldots,$$

$$E_{\frac{1}{\alpha}}(\lambda t^{\alpha};i+1)=\frac{1}{\alpha}\lambda^{-\frac{i}{\alpha}}t^{-i}e^{\lambda^{\frac{1}{\alpha}}t}+\ldots,$$

$$t^{i}E_{\frac{1}{\alpha}}(\lambda t^{\alpha};i+1)=\frac{1}{\alpha}\lambda^{-\frac{i}{\alpha}}e^{\lambda^{\frac{1}{\alpha}}t}+\ldots.$$

Then taking into account the asymptotic representations obtained above, we have:

$$\lim_{t\to\infty}\frac{t^{\alpha}E_{\frac{1}{\alpha}}(\lambda t^{\alpha};\alpha+1)}{\left\|\sum_{i=0}^{n-1}t^{i}E_{\frac{1}{\alpha}}(\lambda t^{\alpha};i+1)z_i^0\right\|-\varepsilon}=\frac{1}{\left\|\sum_{i=0}^{n-1}\lambda^{1-\frac{i}{\alpha}}z_i^0\right\|}.$$

Thus, a positive finite root of equation (1.56) exists if the inequality (1.57) holds true.

In conclusion, we examine the case when $\lambda=0$. Then the equation (1.56) takes on the form:

$$\frac{t^{\alpha}}{\Gamma(\alpha+1)}=\frac{\sum_{i=0}^{n-1}\frac{t^{i}}{i!}z_i^0-\varepsilon}{a-1}. \tag{1.59}$$

At $t=0$ the left-hand side of this equation equals zero and its right-hand side is positive as the initial state of the dynamic system in question does not belong to the terminal set: $z_0^0>\varepsilon$. As $t\to\infty$, the growth rate of the left-hand side is $O(t^{\alpha})$ while the right-hand side increases at the rate of $O(t^{n-1})$. Since $\alpha>n-1$, the equation (1.59) has a positive solution for any initial conditions such that $z_0^0>\varepsilon$.

1.13 Group Pursuit

Let the motion of an object $z=\mathrm{col}(z_1,\ldots,z_{\mu})$, $z_i\in\mathbb{R}^{m_i}$, in the state space $\mathbb{R}^m$, $m=m_1+\ldots+m_{\mu}$, be described by the fractional-order differential equations:

$$\mathbf{D}^{\alpha}z_i=A_iz_i+\varphi_i(u_i,v),\quad 0<\alpha\le 1,\quad i=1,\ldots,\mu. \tag{1.60}$$

Here $\mathbf{D}^\alpha$ stands for the fractional differentiation operator in the sense of Riemann-Liouville or Caputo, A_i is a square matrix of order m_i, the control block is defined by the jointly continuous functions $\varphi_i(u_i, v)$, $\varphi_i : U_i \times V \to \mathbb{R}^{m_i}$, where u_i and v, $u_i \in U_i$, $v \in V$, are control parameters of the i-th pursuer and evader respectively, and the control sets U_i and V are from the set $K(\mathbb{R}^{m_i})$.

Let $g_i(t, z_i^0)$ be the solution to the homogeneous system $\mathbf{D}^\alpha z_i = A_i z_i$. Thus, when $\mathbf{D}^\alpha$ is the Riemann-Liouville fractional differentiation operator ($\mathbf{D}^\alpha = D^\alpha$):

$$g_i(t, z_i^0) = t^{\alpha-1} E_{\frac{1}{\alpha}}(At^\alpha; \alpha) z_i^0,$$

where:

$$z_i^0 = D^{\alpha-1} z_i(t)|_{t=0}.$$

For the Caputo regularized fractional derivative ($\mathbf{D}^\alpha = D^{(\alpha)}$), we have:

$$g_i(t, z_i^0) = E_{\frac{1}{\alpha}}(At^\alpha; 1) z_i^0,$$

where:

$$z_i^0 = z_i(0).$$

The terminal set M^* consists of the sets $M_1^*, \ldots, M_\mu^*$, $M_i^* \subset \mathbb{R}^{m_i}$, such that:

$$M_i^* = M_i^0 + M_i, \tag{1.61}$$

where M_i^0 is a linear subspace of $\mathbb{R}^{m_i}$ and M_i is a convex compact set from the orthogonal complement L_i to the subspace M_i^0 in $\mathbb{R}^{m_i}$.

The dynamic game (1.60), (1.61) is said to be terminated if for some i the inclusion $z_i \in M_i^*$ holds true.

Denote by Π_i the orthoprojector from $\mathbb{R}^{m_i}$ onto L_i. Consider the following set-valued maps:

$$W_i(t, v) = \Pi_i t^{\alpha-1} E_{\frac{1}{\alpha}}(At^\alpha; \alpha) \varphi_i(U_i, v),$$

$$W_i(t) = \bigcap_{v \in V} W_i(t, v), \quad i = 1, \ldots, \mu, \quad t \in \mathbb{R}_+, \quad v \in V.$$

Condition 3 *The maps $W_i(t)$ are nonempty-valued for all $i = 1, \ldots, \mu$, $t \in \mathbb{R}_+$.*

Due to the assumptions on the parameters of the process (1.60), the maps $W_i(t,v)$ are measurable with respect to t and closed-valued with respect to v, $v \in V$. Hence [2], the maps $W_i(t)$ are measurable and closed-valued on $\mathbb{R}_+$. Condition 3 and the measurable selection theorem [2] imply that there exists a measurable selection $\gamma_i(\cdot)$, $\gamma_i(t) \in W_i(t)$, $t \geq 0$, for each i, $i = 1, \ldots, \mu$. Let us fix these selections and set:

$$\xi_i(t, z_i) = \Pi_i g_i(t, z_i) + \int_0^t \gamma_i(\tau) d\tau. \tag{1.62}$$

Consider the set-valued maps:

$$\Re_i(t, \tau, z_i, v) = \{\rho \geq 0 : \ [W_i(t-\tau, v) - \gamma_i(t-\tau)] \bigcap \rho[M_i - \xi_i(t, z_i)] \neq \emptyset\}$$

and their support functions in the direction of $+1$:

$$\rho_i(t, \tau, z_i, v) = \sup\{\rho : \ \rho \in \Re_i(t, \tau, z_i, v)\}, \quad \rho : \Delta \times V \to \mathbb{R}_+,$$

where $\Delta = \{(t, \tau) : \ 0 \leq \tau \leq t < \infty\}$. As before, these functions are called resolving [34].

Taking into account Condition 3, the properties of the process (1.60) parameters, as well as characterization and inverse image theorems, one can show that the set-valued mappings $\Re_i(t, \tau, z_i, v)$ are $\Lambda \times \mathcal{B}$-measurable [4] with respect to τ, v, $\tau \in [0, t]$, $v \in V$, and the resolving functions $\rho_i(t, \tau, z_i, v)$ are $\Lambda \times \mathcal{B}$-measurable in τ, v, by virtue of the support function theorem [4] when $\xi_i(t, z_i) \notin M_i$.

It should be noted that for $\xi_i(t, z_i) \in M_i$ we have $\Re_i(t, \tau, z_i, v) = [0, \infty)$ and hence $\rho_i(t, \tau, z_i, v) = +\infty$ for any $\tau \in [0, t]$, $v \in V$.

Since $\rho_i(t, \tau, z_i, v)$ are $\Lambda \times \mathcal{B}$-measurable in τ, v, they are superpositionally measurable [17], i.e. $\rho_i(t, \tau, z_i, v(\tau))$ is measurable for any measurable $v(\tau)$, $v(\tau) \in V$.

Denote:

$$T_\mu(z) = \inf \left\{ t \geq 0 : \ \inf_{v(\cdot)} \max_i \int_0^t \rho_i(t, \tau, z_i, v(\tau)) d\tau \geq 1 \right\}. \tag{1.63}$$

If the inequality in braces fails for all $t \geq 0$, we set $T_\mu(z) = +\infty$. If for some i $\rho_i(t, \tau, z_i, v) = +\infty$ for $\tau \in [0, t]$, $v \in V$, we assume that $\int_0^t \rho_i(t, \tau, z_i, v(\tau)) d\tau = +\infty$ and the inequality in (1.63) is fulfilled automatically.

Theorem 4 *Suppose Condition 3 holds for the pursuit game (1.60), (1.61) and $T_\mu(z^0) < \infty$ for some initial state $z^0 = (z_1^0, \dots, z_\mu^0)$. Then for at least one i a trajectory of (1.60) can be driven from the initial state z_i^0 to the corresponding set M_i^* at the moment $T_\mu(z^0)$.*

Proof. Denote $T = T_\mu(z^0)$. Let $v(\tau)$, $v : [0, T] \to V$, be an arbitrary measurable function.

Consider the case $\xi_i(T, z_i^0) \notin M_i$ for all $i = 1, \dots, \mu$. Let us introduce the check function:

$$h_\mu(t) = 1 - \max_i \int_0^t \rho_i(T, \tau, z_i^0, v(\tau)) d\tau.$$

Resolving functions $\rho_i(T, \tau, z_i^0, v(\tau))$ are $\Lambda \times \mathcal{B}$-measurable in τ, v, hence, superpositionally measurable [14], i.e. $\rho_i(T, \tau, z_i^0, v(\tau))$ are measurable in τ. Thus, $h_\mu(t)$ is continuous, non-increasing, and $h_\mu(0) = 1$. Since $h_\mu(T) \le 0$, there exists a time instant t_*, $t_* \in (0, T]$, such that:

$$h_\mu(t_*) = 0. \tag{1.64}$$

Let us introduce the set-valued maps:

$$\begin{aligned} U_i(\tau, v) = \{u_i \in U_i :\ & (T-\tau)^{\alpha-1} \Pi_i E_{\frac{1}{\alpha}}(A(T-\tau)^\alpha; \alpha) \varphi_i(u_i, v) - \gamma_i(T-\tau) \\ & \in \rho_i(T, \tau, z_i^0, v)[M_i - \xi_i(T, z_i^0)]\}, \quad i = 1, \dots, \mu. \end{aligned} \tag{1.65}$$

By virtue of the inverse image theorem, the maps $U_i(\tau, v)$ are $\Lambda \times \mathcal{B}$-measurable, hence, according to the measurable selection theorem, in $U_i(\tau, v)$ there exists at least one $\Lambda \times \mathcal{B}$-measurable selection $u_i(\tau, v)$ that is superpositionally measurable. Let us set the pursuers' controls on the interval $[0, t_*)$ to be $u_i(\tau) = u_i(\tau, v(\tau))$.

Due to (1.64), there exists a number i_* such that:

$$1 - \int_0^{t_*} \rho_{i_*}(T, \tau, z_{i_*}^0, v(\tau)) d\tau = 0. \tag{1.66}$$

Setting in (1.65) $\rho_i(T, \tau, z_i^0, v) \equiv 0$ for $\tau \in [t_*, T]$, we obtain the following set-valued maps:

$$U_i^0(\tau, v) = \{u_i \in U_i : (T-\tau)^{\alpha-1} \Pi_i E_{\frac{1}{\alpha}}(A(T-\tau)^\alpha; \alpha) \varphi_i(u_i, v) - \gamma_i(T-\tau) = 0\}.$$

As before, by the theorem on measurable selection, in $U_i^0(\tau, v)$ there exists at least one $\Lambda \times \mathcal{B}$-measurable selection $u_i^0(\tau, v)$ that is superpositionally measurable. Let us set control of the i_*-th pursuer on the interval $[t_*, T]$ to be $u_{i_*}(\tau) = u_{i_*}^0(\tau, v(\tau))$. The controls of the other pursuers we assume to be arbitrary.

Now suppose there exists i such that $\xi_i(T, z_i^0) \in M_i$. In this case we set the control of the i-th pursuer to be:

$$u_i(\tau) = u_i^0(\tau, v(\tau)), \quad \tau \in [0, T].$$

The controls of the other pursuers are assumed to be arbitrary.

Again, consider the case $\xi_i(T, z_i^0) \notin M_i$ for all $i = 1, \ldots, \mu$. It follows from (1.66) and from the equality $\rho_{i_*}(T, \tau, z_{i_*}^0, v) \equiv 0$, $\tau \in [t_*, T]$, that:

$$\int_0^T \rho_{i_*}(T, \tau, z_{i_*}^0, v(\tau))d\tau = 1. \tag{1.67}$$

According to the Cauchy formulas (1.18), (1.22),

$$\Pi_{i_*} z_{i_*}(T) = \Pi_{i_*} g_{i_*}(t, z_{i_*}^0) + \int_0^T (T-\tau)^{\alpha-1} \Pi_{i_*} E_{\frac{1}{\alpha}}(A(T-\tau)^\alpha; \alpha) \varphi_{i_*}(u_{i_*}(\tau), v(\tau))d\tau.$$

Adding and subtracting $\int_0^T \gamma_{i_*}(T-\tau)d\tau$ from the right-hand side and taking into account the pursuers' control laws described above, we obtain:

$$\Pi_{i_*} z_{i_*}(T) \in \xi_{i_*}(T, z_{i_*}^0) \left[1 - \int_0^T \rho_{i_*}(T, \tau, z_{i_*}^0, v(\tau))d\tau\right] + \int_0^T \rho_{i_*}(T, \tau, z_{i_*}^0, v(\tau)) M_{i_*} d\tau.$$

Taking into account (1.67) and the fact that integral of a uniformly bounded compact-valued map is a convex compact set (*Aumann's theorem* [5]), we finally get the inclusion $\Pi_{i_*} z_{i_*}(T) \in M_{i_*}$.

If for some i $\xi_i(T, z_i^0) \in M_i$, then taking into account the i-th pursuer's control law and (1.62) we obtain the inclusion $\Pi_i z_i(T) \in M_i$.

1.14 Encirclement

Consider a game of pursuit involving μ pursuers:

$$\mathbf{D}^\alpha x_i = u_i, \quad \|u_i\| \le 1, \quad i = 1, \dots, \mu\,, \tag{1.68}$$

and one evader:

$$\mathbf{D}^\alpha y = v, \quad \|v\| \le 1\,. \tag{1.69}$$

Here $\mathbf{D}^\alpha$, $0 < \alpha \le 1$, as before, stands for the fractional differentiation operator in the sense of Riemann-Liouville or Caputo, $x_i, y \in \mathbb{R}^m$, $m > 1$. The game is assumed to be terminated if $x_i = y$ for some i.

Let us reduce this problem to the form (1.60), (1.61). Set $z_i = x_i - y$. Then equations (1.68), (1.69) take on the form:

$$\mathbf{D}^\alpha z_i = u_i - v, \quad z_i \in \mathbb{R}^m, \quad \|u_i\| \le 1, \quad \|v\| \le 1\,, \tag{1.70}$$

with the terminal set consisting of the sets:

$$M_i^* = M_i^0 = M_i = \{z_i : \ z_i = 0\}, \quad i = 1, \dots, \mu\,. \tag{1.71}$$

In this case $L_i = \mathbb{R}^m$, $\Pi_i = I$ are the identity operators, and A_i are zero matrices of order m. Hence:

$$E_{\frac{1}{\alpha}}(At^\alpha; 1) = 1, \quad E_{\frac{1}{\alpha}}(At^\alpha; \alpha) = \frac{1}{\Gamma(\alpha)}\,.$$

Let us check the fulfillment of Condition 3:

$$W_i(t, v) = \frac{t^{\alpha-1}}{\Gamma(\alpha)} S - \frac{t^{\alpha-1}}{\Gamma(\alpha)} v, \quad W_i(t) = \{0\} \ne \emptyset, \quad i = 1, \dots, \mu\,,$$

where S stands for the closed unit ball centered at the origin in $\mathbb{R}^m$. The equality $W_i(t) = \{0\}$ implies $\gamma_i(t) \equiv 0$ for all $i = 1, \dots, \mu$.

Consider the case when $\mathbf{D}^\alpha$ denotes fractional differentiation operator in the sense of Riemann-Liouville, i.e. $\mathbf{D}^\alpha = D^\alpha$. Then $\xi_i(t, z_i) = \frac{t^{\alpha-1}}{\Gamma(\alpha)} z_i$. The resolving functions are of the form:

$$\rho_i(t, \tau, z_i, v) = \sup\left\{\rho \ge 0 : \ -\rho \frac{t^{\alpha-1}}{\Gamma(\alpha)} z_i \in \frac{(t-\tau)^{\alpha-1}}{\Gamma(\alpha)} (S - v)\right\} =$$

$$= \sup\left\{\rho \geq 0 : \ -\rho t^{\alpha-1} z_i \in (t-\tau)^{\alpha-1}(S-v)\right\} .$$

Their values can be obtained explicitly as the greater root of the following quadratic equation in ρ:

$$\|(t-\tau)^{\alpha-1} v - \rho t^{\alpha-1} z_i\| = (t-\tau)^{\alpha-1} ,$$

whence:

$$\rho_i(t,\tau,z_i,v) = \frac{(t-\tau)^{\alpha-1}\left[(v\cdot z_i) + \sqrt{(v\cdot z_i)^2 + \|z_i\|^2(1-\|v\|^2)}\right]}{t^{\alpha-1}\|z_i\|^2} .$$

Denote $\tilde{\rho}_i(z_i,v) = \frac{v\cdot z_i + \sqrt{(v\cdot z_i)^2 + \|z_i\|^2(1-\|v\|^2)}}{\|z_i\|^2}$, then:

$$\rho_i(t,\tau,z_i,v) = \left(\frac{t-\tau}{t}\right)^{\alpha-1} \tilde{\rho}_i(z_i,v) .$$

The time of the game termination can be determined from (1.63). Denote $\Delta^{\mu-1}$ the standard $(\mu-1)$-simplex. The following inequalities hold true:

$$\inf_{v(\cdot)} \max_i \int_0^t \rho_i(t,\tau,z_i,v(\tau))d\tau = \inf_{v(\cdot)} \max_{\rho\in\Delta^{\mu-1}} \sum_{i=1}^{\mu} \rho_i \int_0^t \rho_i(t,\tau,z_i,v(\tau))d\tau$$

$$\geq \inf_{v(\cdot)} \sum_{i=1}^{\mu} \frac{1}{\mu} \int_0^t \rho_i(t,\tau,z_i,v(\tau))d\tau = \frac{1}{\mu} \inf_{v(\cdot)} \int_0^t \sum_{i=1}^{\mu} \rho_i(t,\tau,z_i,v(\tau))d\tau$$

$$\geq \frac{1}{\mu} \int_0^t \min_{\|v\|\leq 1} \max_i \rho_i(t,\tau,z_i,v)d\tau = \frac{1}{\mu} \int_0^t \min_{\|v\|\leq 1} \max_i \left(\frac{t-\tau}{t}\right)^{\alpha-1} \tilde{\rho}_i(z_i,v)d\tau$$

$$= \frac{t}{\mu\alpha} \min_{\|v\|\leq 1} \max_i \tilde{\rho}_i(z_i,v) .$$

This implies the following estimation of the game termination time:

$$T_\mu(z) \leq \frac{\alpha\mu}{\delta(z)}, \tag{1.72}$$

where:

$$\delta(z) = \min_{\|v\|\le 1} \max_i \tilde{\rho}_i(z_i, v).$$

Now suppose $\mathbf{D}^\alpha$ denotes fractional differentiation operator in the sense of Caputo, i.e. $\mathbf{D}^\alpha = D^{(\alpha)}$. Then $\xi_i(t, z_i) = z_i$. The resolving functions are of the form:

$$\rho_i(t, \tau, z_i, v) = \sup\left\{\rho \ge 0 : \ -\rho z_i \in \frac{(t-\tau)^{\alpha-1}}{\Gamma(\alpha)}(S - v)\right\}.$$

Their values can be obtained explicitly as the greater root of the following quadratic equation in ρ:

$$\left\|\frac{(t-\tau)^{\alpha-1}}{\Gamma(\alpha)}v - \rho z_i\right\| = \frac{(t-\tau)^{\alpha-1}}{\Gamma(\alpha)},$$

whence:

$$\rho_i(t, \tau, z_i, v) = \frac{(t-\tau)^{\alpha-1}\left[v \cdot z_i + \sqrt{(v \cdot z_i)^2 + \|z_i\|^2(1 - \|v\|^2)}\right]}{\Gamma(\alpha)\|z_i\|^2}$$

$$= \frac{(t-\tau)^{\alpha-1}}{\Gamma(\alpha)}\tilde{\rho}_i(z_i, v)\,.$$

The following inequalities hold true:

$$\inf_{v(\cdot)} \max_i \int_0^t \rho_i(t, \tau, z_i, v(\tau))d\tau = \inf_{v(\cdot)} \max_{\rho \in \Delta^{\mu-1}} \sum_{i=1}^{\mu} \rho_i \int_0^t \rho_i(t, \tau, z_i, v(\tau))d\tau$$

$$\ge \inf_{v(\cdot)} \sum_{i=1}^{\mu} \frac{1}{\mu} \int_0^t \rho_i(t, \tau, z_i, v(\tau))d\tau = \frac{1}{\mu} \inf_{v(\cdot)} \int_0^t \sum_{i=1}^{\mu} \rho_i(t, \tau, z_i, v(\tau))d\tau$$

$$\ge \frac{1}{\mu} \int_0^t \min_{\|v\|\le 1} \max_i \rho_i(t, \tau, z_i, v)d\tau = \frac{1}{\mu} \int_0^t \min_{\|v\|\le 1} \max_i \frac{(t-\tau)^{\alpha-1}}{\Gamma(\alpha)}\tilde{\rho}_i(z_i, v)d\tau$$

$$= \frac{t^\alpha}{\mu\Gamma(\alpha+1)} \min_{\|v\|\le 1} \max_i \tilde{\rho}_i(z_i, v)\,.$$

Combining this result with (1.63) one obtains the following estimation for the game termination time:

$$T_\mu(z) \le \left(\frac{\mu\Gamma(\alpha+1)}{\delta(z)} \right)^{1/\alpha}. \tag{1.73}$$

As $\delta(z) \ge 0$ for all $z \in \mathbb{R}^{\mu m}$, it suffices to determine the states z such that $\delta(z) > 0$, in order to ensure $T_\mu(z) < +\infty$ in both (1.72) and (1.73).

Let us set:

$$\sigma(z) = \min_{\|v\|=1} \max_i \left(\frac{z_i}{\|z_i\|} \cdot v \right).$$

One can readily see that $\sigma(z) > 0$ if and only if $\delta(z) > 0$. Indeed, if $\sigma(z) > 0$, then having treated separately the cases $\|v\| = 0$ and $\|v\| = 1$ one immediately obtains $\delta(z) > 0$. If $\delta(z) > 0$, then, in particular, $\min_{\|v\|=1} \max_i \tilde{\rho}_i(z_i, v) > 0$, which implies $\sigma(z) > 0$.

On the other hand, the following assertion is true.

Lemma 8 *The function $\sigma(z) > 0$ if and only if the origin of the space $\mathbb{R}^m$ belongs to the interior of the convex hull of the points $z_i/\|z_i\|$, $i = 1, \dots, \mu$:*

$$0 \in \operatorname{intco} \left\{ \frac{z_i}{\|z_i\|} \right\}.$$

Proof. Suppose $0 \in \operatorname{intco} \left\{ \frac{z_i}{\|z_i\|} \right\}$. This inclusion can be rewritten in terms of support functions as follows:

$$0 < \max_i \left(\frac{z_i}{\|z_i\|} \cdot v \right) \quad \forall v,\ \|v\| = 1,$$

or

$$\min_{\|v\|=1} \max_i \left(\frac{z_i}{\|z_i\|} \cdot v \right) = \sigma(z) > 0\,.$$

The sufficiency can be proved using the same reasoning in the opposite direction.

Corollary 1 $0 \notin \operatorname{intco} \left\{ \frac{z_i}{\|z_i\|} \right\}$ *if and only if* $\sigma(z) \le 0$.

Corollary 2 *The pursuit game (1.70), (1.71) can be terminated in a finite time if and only if:*

$$0 \in \operatorname{intco}\left\{\frac{z_i^0}{\|z_i^0\|}\right\}. \tag{1.74}$$

Proof. Suppose $0 \in \operatorname{intco}\left\{\frac{z_i^0}{\|z_i^0\|}\right\}$. Then by Lemma 8 $\sigma(z^0) > 0$, hence $\delta(z^0) > 0$. By virtue of the estimations (1.72), (1.73), this implies that the game termination time $T_\mu(z^0)$ is finite.

Now let $0 \notin \operatorname{intco}\left\{\frac{z_i^0}{\|z_i^0\|}\right\}$. This implies $\sigma(z^0) \le 0$ and the set:

$$V_0 = \left\{\|v\| = 1 : \max_i \left(\frac{z_i^0}{\|z_i^0\|} \cdot v\right) \le 0\right\},$$

is non-empty. Let $v_0 \in V_0$ then $\tilde{\rho}_i(z_i^0, v_0) = 0$ for all $i = 1, \ldots, \mu$.

First consider the case when $\mathbf{D}^\alpha$ is the Riemann-Liouville fractional differentiation operator. Then:

$$\rho_i(t, \tau, z_i^0, v_0) = \left(\frac{t-\tau}{t}\right)^{\alpha-1} \tilde{\rho}_i(z_i^0, v_0) = 0, \quad i = 1, \ldots, \mu, \quad 0 < \tau < t.$$

This implies, by definition of $\rho_i(t, \tau, z_i^0, v_0)$, that:

$$\left\{-\rho_i \frac{t^{\alpha-1}}{\Gamma(\alpha)} z_i^0\right\} \bigcap \frac{(t-\tau)^{\alpha-1}}{\Gamma(\alpha)}(S - v_0) = \emptyset \quad \forall \rho_i > 0,\ 0 < \tau < t,\ i = 1, \ldots, \mu,$$

or

$$\rho_i \frac{t^{\alpha-1}}{\Gamma(\alpha)} z_i^0 + \frac{(t-\tau)^{\alpha-1}}{\Gamma(\alpha)}(S - v_0) \bigcap \{0\} = \emptyset.$$

By setting $\rho_i = \frac{\alpha(t-\tau)^{\alpha-1}}{t^\alpha}$ in the latter expression, we derive:

$$0 \notin \frac{t^{\alpha-1}}{\Gamma(\alpha)} z_i^0 + \frac{t^\alpha}{\Gamma(\alpha+1)}(S - v_0).$$

Taking into account (1.18) and:

$$\begin{aligned} \tfrac{t^\alpha}{\Gamma(\alpha+1)}(S - v_0) &= \textstyle\int_0^t \frac{(t-\tau)^{\alpha-1}}{\Gamma(\alpha)}(S - v_0) d\tau \\ &= \left\{\textstyle\int_0^t \frac{(t-\tau)^{\alpha-1}}{\Gamma(\alpha)}(u(\tau) - v_0) d\tau : \ u(\tau) \in S\right\}, \end{aligned} \tag{1.75}$$

we obtain $z_i(t) \neq 0$ for all $i = 1, \ldots, \mu$, $t > 0$, i.e. the game cannot be terminated in finite time.

Now suppose $\mathbf{D}^\alpha$ stands for the Caputo fractional differentiation operator. In this case:

$$\rho_i(t, \tau, z_i^0, v_0) = \frac{(t-\tau)^{\alpha-1}}{\Gamma(\alpha)} \tilde{\rho}_i(z_i^0, v_0) = 0, \quad i = 1, \ldots, \mu, \quad 0 < \tau < t.$$

This implies, by definition of $\rho_i(t, \tau, z_i^0, v_0)$, that:

$$-\rho_i z_i^0 \notin \frac{(t-\tau)^{\alpha-1}}{\Gamma(\alpha)}(S - v_0) \quad \forall \rho_i > 0,\ 0 < \tau < t,\ i = 1, \ldots, \mu,$$

or

$$0 \notin \rho_i z_i^0 + \frac{(t-\tau)^{\alpha-1}}{\Gamma(\alpha)}(S - v_0).$$

By setting $\rho_i = \frac{\alpha(t-\tau)^{\alpha-1}}{t^\alpha}$ in the latter expression, we get:

$$0 \notin z_i^0 + \frac{t^\alpha}{\Gamma(\alpha+1)}(S - v_0).$$

Combining this result with (1.22) and (1.75), we get $z_i(t) \neq 0$ for all $i = 1, \ldots, \mu$, $t > 0$, i.e. the game cannot be terminated in finite time.

Remark 2 *In the case of the Caputo fractional derivative, i.e. when* $\mathbf{D}^\alpha = D^{(\alpha)}$*, the condition (1.74) implies that the initial position of the evader lies within the interior of the convex hull of the pursuers' initial positions. This 'encirclement' condition coincides with the one for the integer-order pursuit game* [34].

1.15 Bagley-Torvik Equation

From the practical point of view, of considerable interest is the equation of the form:

$$az'' + bD^{\alpha,\mu}z + cz = f, \quad 0 < \nu < 2,$$

describing oscillations with fractional damping. Equations of such kind arise in describing vibrations of a plane wing in supersonic gas flow [28] resulting

in the flutter-type phenomena, nano-dimensional sensors [20], etc. At $\alpha = \frac{3}{2}$ this equation is called the Bagley-Torvik equation and describes the motion of a rigid plate in a Newtonian fluid [32].

In describing physical phenomena and processes, as a rule, the Caputo derivative corresponding to the type $\mu = 1$ is used, since in this case the initial conditions have clear physical interpretation. Consider the Bagley-Torvik equation involving the Caputo derivative:

$$ay''(t) + bD^{(3/2)}y(t) + cy(t) = f(t) \tag{1.76}$$

under the initial conditions:

$$y(0) = y_0, \quad y'(0) = y'_0. \tag{1.77}$$

Denote $z_1(t) = y(t)$, $z_2(t) = D^{(1/2)}y(t)$, $z_3(t) = y'(t)$, $z_4(t) = D^{(3/2)}y(t)$. Then, as shown in [15], $D^{(1/2)}z_1 = z_2$, $D^{(1/2)}z_2 = z_3$, $D^{(1/2)}z_3 = z_4$, $D^{(1/2)}z_4 = y''$. Therefore, the equation (1.76) under the initial conditions (1.77) is equivalent to the system:

$$\begin{cases} D^{(1/2)}z_1 & = z_2 \\ D^{(1/2)}z_2 & = z_3 \\ D^{(1/2)}z_3 & = z_4 \\ D^{(1/2)}z_4 & = \frac{1}{a}(-cz_1 - bz_4 + f) \end{cases}$$

or, in matrix form:

$$D^{(1/2)}z = Az + Bf\,, \tag{1.78}$$

where:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -c/a & 0 & 0 & -b/a \end{bmatrix}$$

$$, B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1/a \end{bmatrix}, z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}$$ under the initial conditions:

$$z(0) = z^0 = \mathrm{col}(y_0, 0, y'_0, 0)\,.$$

By virtue of (1.22), the solution to this system is given by the formula:

$$z(t) = E_2(A\sqrt{t}; 1)z^0 + \int_0^t E_2\left(A\sqrt{t-\tau}; \frac{1}{2}\right) B\frac{f(\tau)d\tau}{\sqrt{t-\tau}}\,.$$

Now suppose that in the equation (1.76) $f(t) = u(t) - v(t)$, where u, v are controls of the first and the second players, respectively, and $|u| \le r$, $r > 1$, $|v| \le 1$. We assume that the first player strives to bring the system into the state $z_1 = 0$, while the goal of the second player is the opposite. In such case the system (1.78) takes on the form:

$$D^{(1/2)}z = Az + \bar{u} - \bar{v},$$

where $\bar{u} = Bu$, $\bar{v} = Bv$, $\bar{u} \in rBS$, $\bar{v} \in BS$, $S = [-1, 1]$.

The terminal set is $M^* = \{z \in \mathbb{R}^4 : \ z_1 = 0\}$. Then $M_0 = \{z \in \mathbb{R}^4 : \ z_1 = 0\}$, $L = \{z \in \mathbb{R}^4 : \ z_2 = z_3 = z_4 = 0\}$, $M = \{0\}$, $\Pi = \{\pi_{ij}\}$, where:

$$\Pi = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Then $W(t, v) = \frac{1}{\sqrt{t}}\Pi E_2\left(A\sqrt{t}; \frac{1}{2}\right) B(rS - v)$, $W(t) = \frac{r-1}{\sqrt{t}}\Pi E_2\left(A\sqrt{t}; \frac{1}{2}\right) BS \neq \emptyset$ and the condition (1.32) is straightforwardly satisfied. Set $\gamma(t) \equiv 0$, then $\xi(t) = \Pi E_2(A\sqrt{t}; 1)z^0$. The resolving function is defined as follows:

$$\rho(t, \tau, v) = \max\{\rho \ge 0 : \ -\rho\Pi E_2(A\sqrt{t}; 1)z^0 \in \tfrac{1}{\sqrt{t-\tau}}\Pi E_2\left(A\sqrt{t-\tau}; \tfrac{1}{2}\right) B(rS - v)\}.$$

To find the guaranteed time of the game termination, it is necessary to study the resolving function. In general case this is a complicated problem, as it depends on specific form of the generalized Mittag-Leffler matrix functions $E_2(A\sqrt{t}; 1)$ and $E_2\left(A\sqrt{t}; \frac{1}{2}\right)$. However, in some particular cases the problem can be simplified. For example, let us set $c = 0$ in (1.76) and denote $p = -\frac{b}{a}$. Then the matrix A takes on the form $A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & p \end{bmatrix}$. Then, it can be easily verified that:

$$A^2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & p \\ 0 & 0 & 0 & p^2 \end{bmatrix}, \quad A^k = \begin{bmatrix} 0 & 0 & 0 & p^{k-3} \\ 0 & 0 & 0 & p^{k-2} \\ 0 & 0 & 0 & p^{k-1} \\ 0 & 0 & 0 & p^k \end{bmatrix}, \quad k = 3, 4, \ldots,$$

Hence,

$$E_2(A\sqrt{t};1)=\sum_{k=0}^{\infty}\frac{A^k\sqrt{t^k}}{\Gamma(k/2+1)}=$$

$$=\begin{bmatrix}1 & \frac{\sqrt{t}}{\Gamma(3/2)} & t & p^{-3}E_2(p\sqrt{t};1)-p^{-3}-\frac{\sqrt{t}}{p^2\Gamma(3/2)}-\frac{t}{p}\\ 0 & 1 & \frac{\sqrt{t}}{\Gamma(3/2)} & p^{-2}E_2(p\sqrt{t};1)-p^{-2}-\frac{\sqrt{t}}{p\Gamma(3/2)}\\ 0 & 0 & 1 & p^{-1}E_2(p\sqrt{t};1)-p^{-1}\\ 0 & 0 & 0 & E_2(p\sqrt{t};1)\end{bmatrix},$$

$$E_2\left(A\sqrt{t};\frac{1}{2}\right)=\sum_{k=0}^{\infty}\frac{A^k\sqrt{t^k}}{\Gamma((k+1)/2)}=$$

$$=\begin{bmatrix}\frac{1}{\Gamma(1/2)} & \sqrt{t} & \frac{t}{\Gamma(3/2)} & p^{-3}E_2(p\sqrt{t};1/2)-\frac{1}{p^3\Gamma(1/2)}-\frac{\sqrt{t}}{p^2}-\frac{t}{p\Gamma(3/2)}\\ 0 & \frac{1}{\Gamma(1/2)} & \sqrt{t} & p^{-2}E_2(p\sqrt{t};1/2)-\frac{1}{p^2\Gamma(1/2)}-\frac{\sqrt{t}}{p}\\ 0 & 0 & \frac{1}{\Gamma(1/2)} & p^{-1}E_2(p\sqrt{t};1/2)-\frac{1}{p\Gamma(1/2)}\\ 0 & 0 & 0 & E_2(p\sqrt{t};1/2)\end{bmatrix}.$$

Which in combination with the equalities:

$$\Gamma\left(\frac{1}{2}\right)=\sqrt{\pi},\quad \Gamma\left(\frac{3}{2}\right)=\frac{1}{2}\Gamma\left(\frac{1}{2}\right)=\frac{\sqrt{\pi}}{2},$$

$$E_2(z;1)=e^{z^2}\mathrm{erfc}(-z),$$

$$E_2\left(z;\frac{1}{2}\right)=zE_2(z;1)+\frac{1}{\Gamma(1/2)}=ze^{z^2}\mathrm{erfc}(-z)+\frac{1}{\sqrt{\pi}},$$

where $\mathrm{erfc}(z)=\frac{2}{\sqrt{\pi}}\int_z^{\infty}e^{-t^2}dt$ is the complementary error function, yields:

$$E_2(A\sqrt{t};1)=\begin{bmatrix}1 & 2\sqrt{\frac{t}{\pi}} & t & \frac{e^{p^2t}}{p^3}\mathrm{erfc}(-p\sqrt{t})-\frac{1}{p^3}-\frac{2}{p^2}\sqrt{\frac{t}{\pi}}-\frac{t}{p}\\ 0 & 1 & 2\sqrt{\frac{t}{\pi}} & \frac{e^{p^2t}}{p^2}\mathrm{erfc}(-p\sqrt{t})-\frac{1}{p^2}-\frac{2}{p}\sqrt{\frac{t}{\pi}}\\ 0 & 0 & 1 & \frac{e^{p^2t}}{p}\mathrm{erfc}(-p\sqrt{t})-\frac{1}{p}\\ 0 & 0 & 0 & e^{p^2t}\mathrm{erfc}(-p\sqrt{t})\end{bmatrix},$$

$$E_2\left(A\sqrt{t};\frac{1}{2}\right)=\begin{bmatrix} \frac{1}{\sqrt{\pi}} & \sqrt{t} & \frac{2t}{\sqrt{\pi}} & \frac{\sqrt{t}e^{p^2t}}{p^2}\mathrm{erfc}(-p\sqrt{t})-\frac{\sqrt{t}}{p^2}-\frac{2t}{p\sqrt{\pi}} \\ 0 & \frac{1}{\sqrt{\pi}} & \sqrt{t} & \frac{\sqrt{t}e^{p^2t}}{p}\mathrm{erfc}(-p\sqrt{t})-\frac{\sqrt{t}}{p} \\ 0 & 0 & \frac{1}{\sqrt{\pi}} & \sqrt{t}e^{p^2t}\mathrm{erfc}(-p\sqrt{t}) \\ 0 & 0 & 0 & p\sqrt{t}e^{p^2t}\mathrm{erfc}(-p\sqrt{t})+\frac{1}{\sqrt{\pi}} \end{bmatrix}.$$

Denote:

$$\begin{aligned} \xi_1(t) &= \left\{\Pi E_2(A\sqrt{t};1)z^0\right\}_1 = y_0+ty_0', \\ \omega(t) &= \left\{\tfrac{1}{\sqrt{t}}\Pi E_2\left(A\sqrt{t};\tfrac{1}{2}\right)B\right\}_1 = \\ &= \tfrac{1}{a}\left[\tfrac{e^{p^2t}}{p^2}\mathrm{erfc}(-p\sqrt{t})-\tfrac{1}{p^2}-\tfrac{2\sqrt{t}}{p\sqrt{\pi}}\right], \end{aligned}$$

where $\{x\}_1$ stands for the first component of the vector x. Then the resolving function can be found as the greatest positive root of the following equation in ρ:

$$|\omega(t-\tau)v-\rho\xi_1(t)|=|\omega(t-\tau)|r.$$

Upon solving this equation we find that:

$$\rho(t,\tau,v)=\frac{\omega(t-\tau)}{\xi_1(t)}(v\pm r),$$

we write '+' or '−' in the parentheses depending on whether the expression $\omega(t-\tau)/\xi_1(t)$ is positive or negative. In view of physical meaning of the Bagley-Torvik equation, we have $a>0$, $b>0$, hence $p<0$. Note also that since $\omega(0)=0$ and $\omega'(t)=\frac{1}{a}(2-\mathrm{erfc}(p\sqrt{t}))e^{p^2t}>0$, it follows that $\omega(t)>0$ for all $t>0$. Thus:

$$\inf_v\rho(t,\tau,v)=\frac{(r-1)\omega(t-\tau)}{|\xi_1(t)|}.$$

The guaranteed time of the game termination, therefore, can be found from the relationship:

$$\int_0^t\inf_{v\in V}\rho(t,\tau,v)d\tau=\int_0^t\frac{(r-1)\omega(t-\tau)}{|\xi_1(t)|}d\tau=1.$$

Consider the integral:

$$\int_0^t\frac{(r-1)\omega(t-\tau)}{|\xi_1(t)|}d\tau=\frac{r-1}{|\xi_1(t)|}\int_0^t\omega(\tau)d\tau=$$

$$= \frac{r-1}{a|\xi_1(t)|}\int_0^t \left[\frac{e^{p^2\tau}}{p^2}\mathrm{erfc}(-p\sqrt{\tau}) - \frac{1}{p^2} - \frac{2\sqrt{\tau}}{p\sqrt{\pi}}\right] d\tau =$$

$$= \frac{r-1}{a|\xi_1(t)|}\left[\frac{1}{p^2}\int_0^t e^{p^2\tau}\mathrm{erfc}(-p\sqrt{\tau})d\tau - \frac{t}{p^2} - \frac{4\sqrt{t^3}}{3p\sqrt{\pi}}\right].$$

Let us integrate by parts the integral in brackets. We set $u = \mathrm{erfc}(-p\sqrt{\tau})$, $dv = e^{p^2\tau}$. Then:

$$du = \frac{pe^{-p^2\tau}d\tau}{\sqrt{\pi\tau}}, \quad v = \frac{e^{p^2\tau}}{p^2},$$

and:

$$\int_0^t e^{p^2\tau}\mathrm{erfc}(-p\sqrt{\tau})d\tau = \frac{e^{p^2t}}{p^2}\mathrm{erfc}(-p\sqrt{t}) - \frac{1}{p^2} - \frac{1}{p\sqrt{\pi}}\int_0^t \frac{d\tau}{\sqrt{\tau}} =$$

$$= \frac{e^{p^2t}}{p^2}\mathrm{erfc}(-p\sqrt{t}) - \frac{1}{p^2} - \frac{2\sqrt{t}}{p\sqrt{\pi}}.$$

Finally, we see that the time of the game termination can be found from the equation:

$$\frac{e^{p^2t}}{p^4}\mathrm{erfc}(-p\sqrt{t}) - \frac{1}{p^4} - \frac{2\sqrt{t}}{p^3\sqrt{\pi}} - \frac{t}{p^2} - \frac{4\sqrt{t^3}}{3p\sqrt{\pi}} = \frac{a|\xi_1(t)|}{r-1}.$$

This equation always has a solution since its left-hand side vanishes at $t = 0$ and has the growth rate of $O(t^{3/2})$, while the right-hand side is positive at the initial instant and growing linearly in t.

Bibliography

[1] J.P. Aubin, H. Frankowska: Set-valued analysis of Systems & Control: Foundations & Applications. Birkhauser, Boston, 1990.

[2] R.J. Aumann: Integrals of set-valued functions. J. Math. Anal. Appl., 12:1-12, 1965.

[3] R.L. Bagley, P.J. Torvik: On the appearance of the fractional derivative in the behavior of real materials. J. Appl. Mech., 51, pp. 294-298, 1984.

[4] M. Caputo: Linear model of dissipation whose Q is almost frequency independent – II. Geophys. J. R. Astr. Soc., 13, pp. 529-539, 1967.

[5] A.A. Chikrii: Conflict-Controlled Processes. Kluwer Acad. Publ., Boston, London, Dordrecht, 1997.

[6] A.A. Chikrii: Differential games with several pursuers. 14, pp. 81-107, 1985.

[7] A.A. Chikrii: Optimization of game interaction of fractional-order controlled systems. Optimization Methods and Software, 23(1), pp. 39-72, 2008.

[8] A.A. Chikrii, S.D. Eidelman: Generalized Mittag-Leffler matrix functions in game problems for evolutionary equations of fractional order. Cybernetics and Systems Analysis, 36(3), pp. 315-338, 2000.

[9] A.A. Chikrii, I.S. Rappoport, K.A. Chikrii: Multivalued mappings and their selectors in the theory of conflict-controlled processes. Cybern. Syst. Anal., 43(5), pp. 719-730, 2007.

[10] A.A. Chikrii: Game dynamic problems for systems with fractional derivatives, Vol. 17 of Pareto Optimality, Game Theory and Equilibria, pp. 349-386. Springer, 2008.

[11] A.A. Chikrii, I. Matychyn: Game Problems for Fractional-Order Systems, Vol. XI of New Trends in Nanotechnology and Fractional Calculus Applications, pp. 233-241. Springer, 2010.

[12] F. Clarke: Optimization and Nonsmooth Analysis. Wiley-Interscience, New York, 1983.

[13] K. Diethelm, J. Ford: Numerical solution of the Bagley-Torvik equation. BIT Numerical Mathematics, 42(3), pp. 490-507, 2002.

[14] Gh.E. Draganescu, N. Cofan, D.L. Rujan: Nonlinear vibrations of a nano-sized sensor with fractional damping. J. Optoelectron. Adv. Mater., 7(2), pp. 877-884, 2005.

[15] N.A. Grigorenko: Mathematical Methods of Control of Multiple Dynamic Processes, MSU Publ., Moscow, 1990, (in Russian).

[16] O. Hajek: Pursuit Games, Vol. 120 of Mathematics in Science and Engineering. Acad. Press, New York, 1975.

[17] R. Hilfer: Fractional time evolution of Fractional Calculus, Applications in Physics, pp. 87-130. World Scientific, Singapore, 2000.

[18] Shouchuan Hu, N.S. Papageorgiou: Handbook of Multivalued Analysis. Vol. 1: Theory, Vol. 419 of Mathematics and its Applications. Kluwer Acad. Publ., Dordrecht, 1997.

[19] A.D. Ioffe, V.M. Tikhomirov: Theory of Extremal Problems, Vol. 6 of Studies in mathematics and its applications. North-Holland, Amsterdam, 1979.

[20] A.A. Kilbas, H.M. Srivastava, J.J. Trujillo: Theory and Applications of Fractional Differential Equations, Vol. 204 of North-Holland Mathematics Studies. Elsevier, Amsterdam, 2006.

[21] V.V. Kobelev: Linear non-conservative systems with fractional damping and the derivatives of critical load parameter. GAMM-Mitt., 30(2), pp. 287-299, 2007.

[22] A.N. Kolmogorov, S.V. Fomin: Elements of the Theory of Functions and Functional Analysis. Dover Publ., Mineola; New York, 1999.

[23] N.N. Krasovskii, A.I. Subbotin: Game-Theoretical Control Problems of Springer series in Soviet mathematics. Springer, New York; Berlin, 1988.

[24] Yu.G. Krivonos, I.I. Matychyn, A.A. Chikrii: Dynamic Games with Discontinuous Trajectories. Naukova Dumka, Kyiv, 2005, (in Russian).

[25] K.S. Miller, B. Ross: An introduction to the fractional calculus and fractional differential equations. Wiley & Sons, 1993.

[26] B.S. Mordukhovich: Variational analysis and generalized differentiation, I: Basic Theory; II: Applications, pp. 330-331 of Grundlehren der mathematischen Wissenschaften. Springer, New York, 2006.

[27] I. Podlubny: Fractional Differential Equations, Vol. 198 of Mathematics in science and engineering. Acad. Press, San Diego, 1999.

[28] L.S. Pontryagin: Selected works, v. 1. Selected research papers, volume 1 of Classics of Soviet mathematics. Gordon & Breach Science Publ., New York, 1986.

[29] B.N. Pshenichnyi: Simple pursuit by several objects. Cybernetics and Systems Analysis, 12(3), pp. 484-485, 1976.

[30] S.G. Samko, A.A. Kilbas, O.I. Marichev: Fractional Integrals and Derivatives. Gordon & Breach, Amsterdam, 1993.

[31] Yu.S. Osipov, A.V. Kryazhimskii: Inverse problems for ordinary differential equations: dynamical solutions. Gordon and Breach Publ., Basel, 1995.

Chapter 2

Algorithms of parallel computations

O. Khimich, K. Gromaszek, A. Kotyra

2.1 Introduction

Increasing demands imposed on the quality of project solutions and using of new constructive materials generate a need for the solving of radically new scientific and engineering problems as well as for the carrying out of unique calculations. Need is growing for new methods and approaches related to the construction and investigation of correct computer models which adequately reflect realistic functioning of objects being investigated. These factors result the considerable increase in the volume of information being processed which, in turn, causes the growth of requirements to the computational resources, including requirements to run times of problems.

Run times of problems are determined by four factors: element base, computer's structure and architecture, problems' solving algorithms and quality of composed programs.

Since an increase in the clock rate for the available processors is reaching its

limiting value, the paralleling of computations is the main reserve for increase in computers' performance.

High performance indicators of computers are gained at the expense of integrating of the great number of processors and arrangement of massive multiprocessing. Such computers are designed on the basis of open system ideology. The openness of systems involves the standardization of interfaces, scalability of equipment as well as portability of the software.

The present results of investigations on the development of parallel algorithms for the solving of basic classes of problems of the computational mathematics: linear algebraic systems, algebraic eigenvalue problem, initial-value problems for systems of ordinary differential equations, non-linear equations and systems as well as software described in monographs [6, 7].

At present a great deal of works (for example, [11, 9, 8, 10, 12]) are devoted to the problem of paralleling of algorithms for the solving of basic classes of problems of the computational mathematics. Performed analysis has demonstrated that creation of efficient algorithms requires taking into account parallel computers' architecture, while investigation of them should be carried out with taking into account available computing resources (the limited number of processors on which parallel processes could be carried out; account of interaction times, synchronization and volumes of memory). Moreover, even within one class of parallel computers the program implementation should take into account peculiarities of computers' architecture and their inter-processor communications.

Algorithms and programs implementing them proposed in the monograph ensure the satisfying of basic requirements to parallel algorithms and programs for MIMD-computers parallelism, scalability and loyalty. *Parallelism* implies ability to perform a lot of actions simultaneously that is very important for programs implemented on several processes. *Scalability* enables to perform programs by using different number of processes. *Loyalty* characterizes a need for accessing local data more often than accessing remote data. Significance of these characteristics is determined by 'cost of remote access to memory/cost of local access to memory' ratio. This ratio is a key to the increasing of programs' efficiency on the distributed memory architectures to which the intelligent MIMD-computer belongs.

Algorithms under consideration also investigate and solve problems directly related to the arrangement of parallel computations, including:

1. efficient paralleling of computations,

2. determining of efficient scheme of the initial information decomposition,

3. determining both of the topology and amount of processors required for the efficient solving of problem,

4. the ensuring of uniform loading of all processors being used for the solving of problem (balancing of processes),

5. synchronization of exchanges between processes,

6. minimization of exchanges between processes.

From the algorithmic point of view, all known sequential algorithms (methods) are prototypes of most known parallel algorithms (with some exception, for example, in case of banded matrices with narrow bandwidth during the solving of linear algebraic systems). In fact, under such an approach each processor simultaneously implements sequential computational schemes over local data block obtained as a result of the initial data decomposition by means of employing some or other method. It is fundamentally important that with such an adaptive approach to parallel algorithms, the succession of results obtained in the theory of computational methods is preserved (accuracy, convergence, complexity, etc.)

From this point of view, the typical phenomenon is evolution undergone by parallel algorithms (in problems of linear algebra) based on elementary one- and two-sided Jacoby transformations (known to be the best representatives of the sequential algorithms with respect to the economy) to methods based on Givens' and Householders transformations – primary methods in the hierarchical list among orthogonal sequential algorithms with respect to the efficiency. On the one hand, this fact is associated with natural parallelism of algorithms based on the Jacobi rotations, while, on the other hand, it is associated with Haydn's effect which lowers efficiency of Givens' and Householder's methods for the linear algebra problems with common technique for the storing of matrices (by means of one-dimensional block column distribution between processes). In this case each process works only with one block of matrix columns. The k-th column is distributed to the process numbered k/tc, where $tc{=}n/p$ is the maximum number of columns distributed to the process. This scheme doesn't ensure sufficient balancing of the processes' loading since as soon as first tc columns be processed the zero process will

stop its work. Similarly one-dimensional block column method of distributing doesn't ensure good balancing of processes.

An idea of cyclic technique for the storing and processing of matrices is a notable step towards improvement of the processes' balancing and hence improvement of performance of computational algorithms and leads to the balancing scheme for the matrix triangulation and tri-diagonalization algorithms. The idea of column (row) – cyclic way of the matrix storing and processing was independently proposed in the number of works (see, for example, [8]), including [10]. Thus, according to row-column scheme the matrix is distributed between p processors as follows: the i-th process works with rows numbered i, $i+p$, $i+2p$. This scheme by maintaining the same order of rows' (columns') processing as in sequential triangulation and tri-diagonalization algorithms provides the changing by 1 of the process's number when passing from some row to the next. Thus, in the process of algorithms' implementation similarly the same amount of computations is gained in each process, i.e. the influence of Haidn's effect is almost excluded.

Such situation is typical not only for algorithms based on orthogonal transformations, but also for Gauss, Cholesky and some other algorithms used for the matrix reduction to one standard forms characteristic feature of them is gradual (from step to step) decrease in dimensions of the matrix being processed.

The third scheme (column block-cyclic) is a compromise between schemes described above. By choosing the block size equal to n, columns are broken by blocks sized nb and distributed in a cyclic manner. This implies that column k will be processed in the process with logical number $[(k\text{-}1)/nb]$ mod p. In fact this scheme involves previous two schemes for the data storing with $nb = tc = n/p$ and $nb = 1$. For nb greater than 1 this results in the somewhat worse balancing of processes compared to the cyclic distribution of columns, but in so doing the total time of system's latency decreases since the amount of data exchanges between processes decreases.

The fourth scheme (two-dimensional block-cyclic distribution) involves the precious schemes as particular cases. Along with good balancing and latency indicators, one of the significant indicators of advisability of such matrix distributing and processing is the possibility to apply block procedures from standard library programs of BLAS and BLACS [12].

Thus, the efficiency of algorithms to a large extent is determined by the fol-

lowing: scheme of the initial data distribution between processors; agreement between problem's solution algorithm and computer architecture; topology of inter-processor communication.

The quality of parallel algorithms can be estimated by such criteria as acceleration coefficient S_p and efficiency coefficient E_p:

$$S_p = T_1/T_p, \quad E_p = S_p/p\,,$$

where p is the number of processors involved in the solving of problem on MIMD-computer; T_p is the time required for the solving of problem on the MIMD-computer with p processes; T_1 is the time required for the solving of the same problem on hypothetical mono-processor computer possessing performance inherent in mono-processor computer and operating memory equal to the total memory used by p processes.

In addition to notations introduced above further we are going to make use of the following notations:

t – average time required for the performing of one arithmetic operation,

t_o – time required for exchange by one machine word between two processes,

t_c – time required for the establishing of connections between two processes:

$$\tau_{\rm o} = t_{\rm o}/t, \quad \tau_{\rm c} = t_{\rm c}/t\,.$$

Construction of parallel algorithms assumes that information required for the implementation of the computational algorithm is stored and processed either in the operating memory of the hypothetical mono-processor computer or in the overall memory of the MIMD-computer on which p processes are carried out, i.e. computational process is carried out without using the external memory.

Parallel algorithms intended for the solving of problems are presented in terms of processes rather than processors. Process is an independent 'thread' of management possessing its own (not interesting with any other) memory. Each process is implemented on one core (or physical on-core processor). In

the general case, several processes can be carried out on one core (or physical one-core processor). In this case the control system is assumed to manage the scheduling of execution of processes. Parallel algorithms implemented in program and presented here have been developed and tested for the case 'one processor per one core' by using interaction in pairs and collective exchanges.

Processes may communicate in pairs by means of either 'point-to-point' – type exchanges or collective communications. In parallel algorithms the following collective communications are used most often:

1. ***broadcasting*** – one of processes from the communicating group sends data to all processes of this group,

2. ***multi-gathering*** of the number (array of numbers) – all processes from the communicating group sends equal portions of data to all processes of this group; in so doing operations, for example, addition or choosing of the maximum value, etc, are performed over corresponding components of data obtained from different processes,

3. ***multi-gathering*** of vector – all processes of the communicating group sends equal portions of data to one process of this group, and a vector if formed from the obtained data in the process.

These collective communications can be verified by means of functions from the library of parallel programming environment MPI (see for example [13]): MPI – Bcast – for broadcasting; MPI – Reduce – for multi-gathering of array of numbers; MPI – Gather – for multi-gathering of vector; MPI – Allreduce, MPI – Allgather – for multi-gathering and subsequent broadcasting of array or vector, respectively. As a rule, tree-algorithm is used in the implementation of the functions which for one operation of multi-gathering or multi-casting produces the minimum number of communications (synchronizations) equal to $\log_2 p$ (p – the number of processes in group) between pairs of process from the group.

Parallel algorithms have been implemented by means of programs written in the programming language C for the MPI-standard parallel programming environment. Parallel algorithms and programs described here were tested on intelligent workstation from the Inparcom-family both in the operating environment Linux and in the inter-processor communication environment MVAPICH optimized for Infiniband.

2.2 Linear algebraic systems

2.2.1 Methods for the solving of linear algebraic systems

The solving of LAS with real matrices can be carried out by either direct or iterative methods. Direct methods for the solving of problems:

$$Ax = b\,, \tag{2.1}$$

with the following real matrices are dealt with:

- LAS with dense square non-singular general matrix,
- LAS with symmetric positive definite matrix.

Symmetric positive definite matrix is a matrix all eigenvalues of which are positive, while positive semi-definite matrix is a matrix with some eigenvalues equal to zero.

Consider some direct methods for solving LAS on the basis of which parallel algorithms for MIMD-computers were created. Most direct methods are based on the idea of sequential equivalent transformations of particular system for the sake of excluding unknowns from the part of equations. As a result, the original system (2.1) of order n is transformed into equivalent system:

$$A^{n-1}x = b^{n-1}\,, \tag{2.2}$$

with matrix A^{n-1} of simpler form, for example, triangular or diagonal. Thus, it will be not difficult to solve the equivalent system (2.2).

A process of equivalent transformations may be presented as sequential premul- tiplication of matrix A and right-hand side b by matrices M_i $(i = 1, 2, \ldots, n-1)$, in so doing one multiplication results in the annulments of, at least, one element of the matrix being transformed. Then:

$$A^{n-1} = MA,\;\; b^{n-1} = Mb\,, \tag{2.3}$$

where $M = M_{n-1}, M_{n-2}, \ldots, M_1$. Matrices M_i may be triangular, orthogonal.

Various modifications of elimination methods are in essence methods for decomposition of the matrix A into product of triangular matrices, triangular or orthogonal matrices and so on. Indeed, from (2.3) it follows that $A = LU$ if $L = M^{-1}$ and $U = A^{n-1}$.

Gauss method *is one of the most efficient methods for solving of linear systems with non-singular general matrices.* Algebraic basis for the Gauss elimination is a statement that for square non-linear matrix A of order n there exists unique lower triangular matrix L with units on the principal diagonal and unique upper triangular matrix U such that $LU = A$ and $\det(A) = u_1, u_{22}, \ldots, u_{nn}$.

During the solving of system with non-singular matrix by Gauss method the following three sub-problems can be separated:

1. LU-decomposition of system's matrix:

$$A = LU\,, \tag{2.4}$$

2. solving of LAS with lower triangular matrix (this problem is often referred to as a forward substitution):

$$Ly = b\,, \tag{2.5}$$

3. solving of LAS with upper triangular matrix (backward substitution):

$$Ux = y\,. \tag{2.6}$$

LU-decomposition of system's matrix (2.5) consists of $(n-1)$ steps. At the s-th (s = 1, 2, ..., $n-1$) step a diagonal block of order $n-s+1$ located in lower right-hand corner of the matrix $A^{(s-1)}$ is transformed. Element $a_{ss}^{(s-1)}$ is referred as to a pivotal element of the s-th step. For the successful implementation of Gauss method all pivotal elements should be non-zero. Besides, the proximity of pivotal elements to zero can result in large error in the computed solution. In order to ensure the fulfillment of conditions $\left|a_{ss}^{(s-1)}\right| \geq \varepsilon > 0$ for all s = 1, 2, ..., $n-1$ the largest modulus (pivotal) elements should be chosen at each step either in the entire block of matrix $A^{(s-1)}$ being transformed or in the first row of this block or in its first column. By permutation of rows

and/or columns of matrix $A^{(s-1)}$ the pivotal element is placed on the position of the element $a_{ss}^{(s-1)}$.

As a rule, ***for solving of LAS with symmetric positive definite matrix*** various versions of ***Cholesky method*** are used based on the following theorem. If A is a symmetric positive definite then there exists real non-singular lower triangular matrix L such that $LL^T = A$.

In the solving of system with symmetric positive definite matrix by LL^T-version of Cholesky method, similarly as in Gauss method, the following three sub-problems may be separated:

1. LLT-decomposition of system's matrix,
2. solving of LAS lower triangular matrix:

$$Ly = b\,,$$

3. solving of LAS with upper triangular matrix:

$$L^T x = y\,.$$

During the LL^T-decomposition of symmetric positive definite matrix n square roots are to be evaluated. To avoid this LDL^T-version of Cholesky can be employed. In this case:

$$A = LDL^T\,, \tag{2.7}$$

where L is a lower triangular matrix with units at the principal diagonal, while D is positive definite diagonal matrix. This decomposition can easily be transformed into LL^T-decomposition: $A = (LD^{1/2})(LD^{1/2})^T$. In the LDL^T-version of Cholesky method the following three sub-problems may be separated:

1. LDL^T-decomposition of system's matrix,
2. the solving of LAS with upper triangular matrix:

$$Ly = b\,, \tag{2.8}$$

3. the solving of LAS with upper triangular matrix:

$$L^T x = D^{-1} y\,. \tag{2.9}$$

2.2.2 Parallel algorithms for the solving of linear systems

Algorithms with parallel arrangements of computations will be further referred to as parallel algorithms. For the implementation on MIMD-computers of methods described in paragraph 2.2.1 the following parallel algorithms are required:

1. LU-decomposition of non-singular general matrix (2.4),
2. the solving of LAS with lower (2.8) and upper (2.9) triangular matrices.

Row-cyclic parallel algorithms for triangular decomposition of dense non-singular matrix.

For solving LAS (2.1) with square non-singular matrix the LU-decomposition (2.4) of the Gauss method is employed while for system with symmetric positive definite matrix the LDLT-decomposition (2.7) of Cholesky method is used.

In the version of LU-decomposition algorithm being used U is an upper triangular matrix, while, lower triangular matrix L with units on principal diagonal isn't formed explicitly. Thus, LU-decomposition is carried out for $s = 1, 2, \ldots, n-1$ by following formulas:

$$\begin{gathered} u_{sj} = a_{sj}^{(s-1)}, j = s, s+1, \ldots, n; m_{ss} = (u_{ss})^{-1}), \\ m_{is} = a_{is}^{(s-1)} m_{ss}, a_{ij}^{(s)} = a_{ij}^{(s-1)} - u_{sj} m_{is}, i, j = s+1, \ldots, n), \\ u_{nn} = a_{nn}^{(n-1)}, m_{nn} = (u_{nn})^{-1} . \end{gathered} \tag{2.10}$$

The LDL^T-decomposition (2.7) of symmetric matrix is carried out for $s = 1, 2, \ldots, n-1$ by following formulas ($d_1 = a_{11}$):

$$\begin{gathered} t_{is} = a_{is} - \sum_{k=1}^{s-1} t_{ik} l_{sk} \ \ (i = s+1, \ldots, n), \\ l_{s+1,k} = t_{s+1,k}/d_k (k = 1, \ldots, s), \\ d_{s+1} = a_{s+1,s+1} - \sum_{k=1}^{s} t_{s+1,k}\, l_{s+1,k} . \end{gathered} \tag{2.11}$$

Distribution of data and results. Efficient paralleling of triangular decomposition algorithms for Gauss and Cholesky method is based both on row-cyclic scheme for the distribution of matrix elements between processes [8, 10] and on natural parallelism essential for matrix-vector operations.

Row-cyclic distribution scheme consists in the following: elements of rows numbered k, $k+p$, $k+2p$, ..., ($k = 1, 2, \ldots, p, p$ – the number of processes being used) are distributed in succession to process with logical number $k-1$. This scheme is used for distributing elements of the original matrix A of system (2.1), triangular matrices L and U, diagonal matrix D, right-hand side of system b and results of solving systems with triangular matrices y and x.

In the case of symmetric matrix of LAS all matrix elements are to be predetermined. In so doing computations are carried out so that only elements of lower matrix triangle be modified.

Row-cyclic parallel algorithm for LU-decomposition (2.4) of dense nonsingular matrix (***Gauss method***) ***with column pivoting*** for $s = 1, 2, \ldots, n-1$ consists in following sequence of operations:

1. each process simultaneously and independently of other processes chooses maximum elements among elements $a_{is}^{(s-1)}$ $(i \geq s)$ of the s-th column distributed to this process,
2. each process searches for pivotal element of the s-th column (maximum element among maximum elements found by each process) by means of broadcasting and multi-gathering operations in the introduction to this monograph),
3. permutation both of row in which the pivotal element is located and the s-th row,
4. modification of permutation vector for right-hand sides,
5. broadcasting of the s-th row (containing the pivotal element) to all processes,
6. each process according to the row-cyclic distribution scheme modifies matrix $A^{(s-1)}$ by formulas (2.10).

As a result of performing LU – factorization each process contains local parts of matrices L and U that can be stored on the place of corresponding

local parts of the original matrix A. Besides, vector of row's permutations is formed which is used in the calculation of solution of system (2.5).

Row-cyclic parallel algorithm. LDL^T-decomposition (2.7) of dense symmetric matrix (**Cholesky method**) for $s = 1, 2, \ldots, n-1$ consists in the following sequence of operations:

1. by means of broadcasting operation the array of elements l_{sk} $(k = 1, \ldots, s-1)$ and d_s of the s-th row matrices L and D are broadcasted to all processes,
2. each process by virtue of (2.11) evaluates values $t_{is}(i = s+1, \ldots, n)$ according to scheme of distributing elements of the original matrix A,
3. values of elements $l_{s+1,k}(k = 1, \ldots, s-1)$ of the $(s+1)$-th row of the matrix L and d_{s+1} are evaluated by formulas (2.41) by process to which $(s+1)$-th row of matrices is distributed.

Efficiency of algorithms. In order to estimate efficiency, i.e. coefficients of acceleration and efficiency (see introduction) of the parallel algorithm it's necessary to evaluate expenses of computing resources for p parallel processes and compare them with expenses required for the sequential version of the algorithm under consideration.

As it is known the total number of arithmetic operations required for the LU-decomposition (2.4) of square matrix by Gauss method is estimated by value $O_1 = 2n^3/3 + O(n^2)$.

The number of arithmetic operations performed by one process at the s-th step of algorithm $(s = 1, 2, \ldots, n-1)$ is approximately equal to:

$$\frac{n-s}{p}(2n - 2s + 1) + 1\,.$$

Then the number of arithmetic operations performed by each of p processes for $n-1$ steps is estimated by value $O_p = \frac{2n^3}{3p} + O\left(\frac{n^2}{p}\right)$.

Generally, the s -th step of parallel algorithm requires the performing of the following operations: for the column pivoting – one multi-gathering and one broadcasting of one double machine word; for the permutation of rows: one exchange by array consisting of n elements between two process one broadcasting of array consisting of $n-s+1$ elements. Hence, the total number

of synchronization is evaluated by value $O_c \approx n(3 + 3\log_2 p)$, while the total amount of data by which the chosen processes exchanges approximately amounts to $O_o \approx 0,5(n+1)n\log_2 p + n^2$ double words in case of employing 'tree'-algorithm for broadcasting and multi-gathering [13].

Thus, execution time of parallel algorithm for the LU – decomposition (2.4) under consideration is estimated by value $T_p = (O_p + O_c\tau_c + O_o\tau_o)t$, where t is time required for the performing of one arithmetic operation, while τ_c and τ_o are determined in the introduction. Time required for the performing of corresponding sequential algorithm is estimated by value $T_1 = O_1 t$. Then for $n >> p$ coefficients of acceleration and efficiency of parallel row-cyclic algorithm of the LU – decomposition (2.4) can be estimated as follows:

$$S_p \equiv \frac{T_1}{T_p} \approx p\left(1 + \frac{3p\,(\log_2 p + 2)}{4n}\tau_1\right)^{-1}, \quad E_p = \frac{S_p}{p},$$

where $\tau_1 = \tau_o + \frac{6\log_2 p+2}{n(\log_2 p+2)}\tau_c$. If $\frac{3p(\log_2 p+2)}{4n}\tau_1 << 1$, then $E_p \approx 1 - \frac{3p(\log_2 p+2)}{4n}\tau_1$.

Similarly efficiency of parallel row-cyclic algorithm for the LDL^T – decomposition of Cholesky method for dense symmetric matrix can be estimated.

As it is known, the total number of arithmetic operations required to the LDL^T – decomposition (2.7) of Cholesky method for dense symmetric matrix is estimated by value: $O_1 = n^3/3 + O(n^2)$, while the number of arithmetic operations performed by each of p processes with taking into account non-uniform loading of processes data exchange by value:

$$O_p = \frac{n^2\,(n/3 + 1,5p - 1)}{p} + O\,(n)\ .$$

At the s-th step of parallel algorithm one operation of broadcasting of the array consisting of s elements is performed by each process. Hence, the total number of exchanges is estimated by value $O_c \approx n\log_2 p$, and at that the total number of data by which processes exchange approximately amounts to $O_o \approx 0,5n^2\log_2 p$ double words.

Then with $n >> p$ coefficients of acceleration and efficiency of parallel row-cyclic algorithm of the LDL^T – decomposition (2.7) can be estimated as follows:

$$S_p \approx p\left(1 + \frac{4,5p - 3}{n} + \frac{3p\log_2 p}{2n}\tau_2\right)^{-1}, \quad E_p = \frac{S_p}{p},$$

where $\tau_2 = \tau_> + \frac{2}{n}\tau_A$. If $\frac{4,5p-3}{n} + \frac{3p\log_2 p}{2n}\tau_2 << 1$, then:

$$E_p \approx 1 - \frac{4,5p-3}{n} - \frac{3p\log_2 p}{2n}\tau_2.$$

Row-cyclic parallel algorithms for the solving of linear algebraic systems with triangular matrices.

After the performing of LU – decomposition (2.4) or LDL^T – decomposition (2.7) the solving of LAS (2.1) reduces to the solving of systems with lower and upper triangular matrices (2.5), (2.6) or (2.8), (2.9), respectively.

It should be noted once more that during the LU – decomposition the lower triangular matrix L is not formed explicitly. With taking into account this fact the solving of linear algebraic systems (2.5) with lower triangular matrix is carried out for $s = 1, 2, \ldots, n-1$ according to the following formulas (notations from (2.10)):

$$y_s = b_s^{(s-1)}, \quad b_i^{(s)} = b_i^{(s-1)} - m_{is}y_s \quad (i = s+1, \ldots, n), \tag{2.12}$$

with $y_n = b_n^{(n-1)}$.

LAS (2.6) with upper triangular matrix for $s = n, n-1, ..., 2$ is solved as follows (notations from (2.10), (2.12)):

$$x_s = y_s^{(n-s)}, \quad y_i^{(n-s+1)} = y_i^{(n-s)} - u_{is}x_s \quad (i = 1, \ldots, s-1), \tag{2.13}$$

where $y_i^{(0)} = y_i/u_{ii}$ $(i = 1, 2, \ldots, n)$ and $x_1 = y_1^{(n-1)}$.

The solving of LAS (2.8) with lower triangular matrix obtained as a result of LDL^T – decomposition (2.7) of the symmetric matrix is carried out for $s = 1, 2, \ldots, n-1$ by formulas analogous to (2.12):

$$y_s = b_s^{(s-1)}, \quad b_i^{(s)} = b_i^{(s-1)} - l_{is}y_s \quad (i = s+1, \ldots, n), \tag{2.14}$$

where $y_n = b_n^{(n-1)}$, while the solving of LAS (2.9) with upper triangular matrix is evaluated as follows $\left(x_n = \frac{y_n}{d_n}\right)$:

$$x_i = \frac{y_i}{d_i} - \sum_{k=i+1}^{n} l_{ki}x_k \quad (i = n-1, \ldots, 2, 1). \tag{2.15}$$

Formulas (2.11)−(2.15) are given for the case of one right-hand side. If the number of right-hand sides of system (2.1) $q > 1$, then in this case components of vectors in these formulas should be replaced by elements of corresponding rows both of matrices of right-hand sides and solutions of systems.

Distribution of data and results. As stated in previous paragraph, the row-cyclic scheme is employed for the inter-processor distribution elements of triangular matrices L and U, diagonal matrix D, as well as right-hand side b and results y and x of solving systems (2.5), (2.6) or (2.8) with triangular matrices.

Elements of matrix LT of system (2.9) are distributed between processes by row-cyclic scheme.

Row-cyclic parallel algorithm. For the solving of system (2.5) $Ly = b$ the so-called column algorithm is employed which for $s = 1, 2, \ldots, n-1$ consists of the following sequence of operations:

1. by means of broadcasting operation the array q of elements of the s-the row of matrix of the solution y is broadcasted to all processes (for the definition of the broadcasting operation see introduction),
2. each process performs modification (2.12) of values $b_i^{(s)}$ $(i = s+1, \ldots, n)$ according to scheme for the distribution of elements of the right-hand side b.

The same algorithm is employed for the solving of system (2.8) with formula (2.12) replaced by (2.14), while LAS (2.6) with upper triangular matrix is solved by the same algorithm in which $s = n, n-1, \ldots, 2$, and modifications are carried out according to (2.13).

Column-cyclic parallel algorithm. For the solving of system (2.9) $LTx = D^{-1}y$ the inner products algorithm is employed which for $i = n-1, n-2, \ldots, 1$ consists of the following sequence of operations (first put $x_n = y_n/d_n$):

1. each process according to scheme for distribution both of elements of matrix L and solution x by formula (2.15) evaluates partial sums required for finding i-th row of solutions $x_{i,k}$ $(k = 1, \ldots, q)$;

2. a process to which i-th row of solutions $x_{i,k}$ ($k = 1, \ldots, q$) has been distributed carries out multi-gathering of partial sums evaluated by all process (for definition of multi-gathering see introduction).

Efficiency of algorithms. The total number of arithmetic operations required for the solving of LAS with triangular matrix and q right-hand sides is estimated by value $O_1 = qn^2 + O(qn)$. The number of arithmetic operations performed by each of p processes during the solving of systems (2.5) or (2.8) $Ly = b$ is estimated by value $O_p \approx \frac{qn|n-1|}{p}$, while during the solving of systems (2.6) $Ux = y$ or (2.9) $L^T x = D^{-1}y$ – by value $O_p \approx \frac{qn(n+p-1)}{p}$.

Each step of both parallel algorithms involves the performing of either one broadcasting operation or one multi-gathering array consisting of q elements. Hence, the total number of data by which the process exchanges approximately amounts to $O_c \approx n \log_2 p$ double words.

Then with $n >> p$ and $n >> q$ acceleration and efficiency coefficients of parallel algorithms under consideration for the solving of LAS with triangular matrices may be estimated as follows:

$$S_p \approx p\left(1 + \frac{p\log_2 p}{n}\tau_3\right)^{-1} \quad \text{or} \quad S_p \approx p\left(1 + \frac{p-1}{n} + \frac{p\log_2 p}{n}\tau_3\right)^{-1},$$

where $\tau_3 = \tau_o + \frac{1}{q}\tau_c$. $E_p = \frac{S_p}{p}$; if $\frac{p\log_2 p}{n}\tau_3 << 1$ then:

$$E_p \approx 1 - \frac{p\log_2 p}{n}\tau_3 \quad \text{or} \quad E_p \approx 1 - \frac{p-1}{n} - \frac{p\log_2 p}{n}\tau_3.$$

Two-dimensional block-cyclic parallel algorithms for the solving of LAS with dense non-singular matrices.

Block algorithm for the solving of LAS enable to reduce implementation of algorithms to the performing of operations in processes over separate blocks of matrices which they are broken into instead of performing operations over separate elements of matrices. As a rule, dimensions of blocks are determined with taking into account the volume of computers' cache memory [12]. This provides high speed of matrix-vector operations.

Consider first how the following block algorithms for solving LAS with dense matrices can be implemented on computer with sequential arrangement of

computations: LU – decomposition of non-singular matrix and LL^T – decomposition of positive definite matrix.

Block algorithm for LU-decomposition. During the solving of LAS $Ax{=}b$ (2.1) algorithm for LU – factorization reduces matrix $A$to the form $A{=}PLU$, where P – is a matrix of permutations, L is a lower triangular matrix (with units on principal diagonal) while U is an upper triangular matrix. This enables to replace the solving of one system (2.1) with general matrix by the solving of two LAS $Ly{=}b$ and $Ux{=}y$ with triangular matrices.

Let us assume that n-th order matrices A, L and U are to be divided into square blocks of order s.

At the k-th step of algorithm ($k = 1, 2, \dots$) let us represent a sub-matrix $A^{(k)}$ (diagonal block of matrix A) of order $r = n{-}(k{-}1)s$ which contains the last r rows and r columns of the matrix A in the form:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = P \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} = P \begin{pmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{pmatrix},$$

where block A_{11} is of size $s{\times}s$, block A_{12} – $s{\times}(r{-}s)$, block $A_{21}{-}$ $(r{-}s){\times}s$ and block $A_{22}{-}$ $(r{-}s){\times}(r{-}s)$.

Let us carry out a sequence of Gauss transformations (see paragraph 2.2.1) on the part of matrix (Fig. 2.1) consisting of blocks A_{11} and A_{21}:

$$l_{im} = \frac{a_{im}}{a_{mm}}, \quad l_{ij} = a_{ij} - l_{im}a_{mj}\,, \tag{2.16}$$

where $i = \overline{1,s}$, $m = \overline{1,s}$, $j = \overline{m+1,r}$.

As a result, we get matrices L_{11}, L_{21}. Further let us find from them unknown matrices U_{11} and $\tilde{A}_{22}$ (the latter is formed on the place of A_{22}):

$$U_{12} \leftarrow (L_{11})^{-1}A_{12}\,, \tag{2.17}$$

$$\tilde{A}_{22} \leftarrow A_{22} - L_{21}U_{12} = L_{22}U_{22}\,. \tag{2.18}$$

Further, the value of k is increased by 1 and computations are repeated for the new value of k, the matrix $\tilde{A}_{22}$ being considered as a sub-matrix $A^{(k)}$.

It is easily seen that total number of arithmetic operations required for the block LU – decomposition (2.4) of square matrix by Gauss method is estimated, as in the case of non-block version, by value $O_1 \approx 2n^3/3 + O(n^2)$.

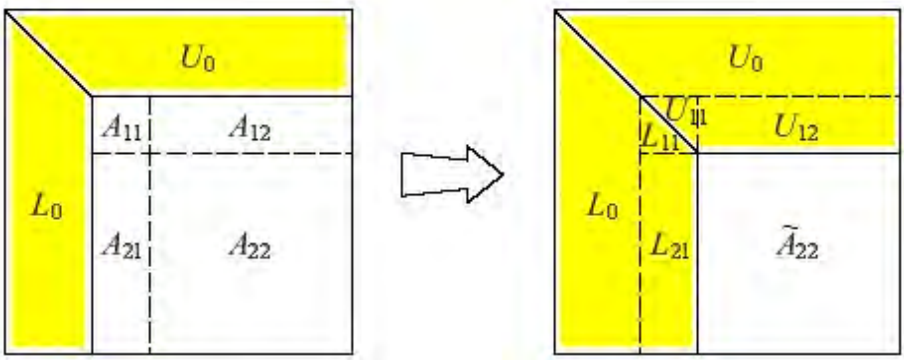


Figure 2.1: Scheme of implementation of the block algorithm for LU – decomposition

Block algorithm for the LL^T-factorization. In the solving of LAS $Ax{=}b$ (2.1) with symmetric positive definite matrix the LL^T – factorization algorithm reduces matrix $A$to the form $A = LL^T$, where L is a lower and L^T – an upper triangular matrices.

Similarly as in the previous case we'll consider n-th order matrices A and L which are broken into s-th order square blocks.

At the k-th step of block algorithm for the LL^T – decomposition under consideration ($k = 1, 2, \dots$) the sub-matrix $A^{(k)}$ of order $r = n{-}(k{-}1)s$ (diagonal block of matrix A) which contains last r rows and r columns of the matrix A may be presented in the form:

$$\begin{pmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22} \end{pmatrix} = \begin{pmatrix} L_{11}L_{11}^T & L_{11}L_{21}^T \\ L_{21}L_{11}^T & L_{21}L_{21}^T + L_{22}L_{22}^T \end{pmatrix},$$

where block A_{11} is of size $s{\times}s$, $A_{12}{-}$ $s{\times}(r{-}s)$, block $A_{21}{-}$ $(r{-}s){\times}s$, block $A_{22}{-}$ $(r{-}s){\times}(r{-}s)$.

For the implementation of algorithm the following operations are to be performed at each step:

1. perform LL^T – decomposition A_{11} (see paragraph 2.2.1) and get the matrix L_{11},
2. modify matrix L_{21} by formula:

$$\tilde{L}_{21} = L_{21} \cdot \left(L_{11}^T\right)^{-1},$$

3. modify matrix A_{22} by formula:

$$\tilde{A}_{22} = A_{22} - \tilde{L}_{21} \cdot \tilde{L}_{21}^T .$$

Hence, at the k-th step a transformed part of the matrix L is obtained. At the next step the matrix $\tilde{A}_{22}$ is transformed (Fig. 2.2).

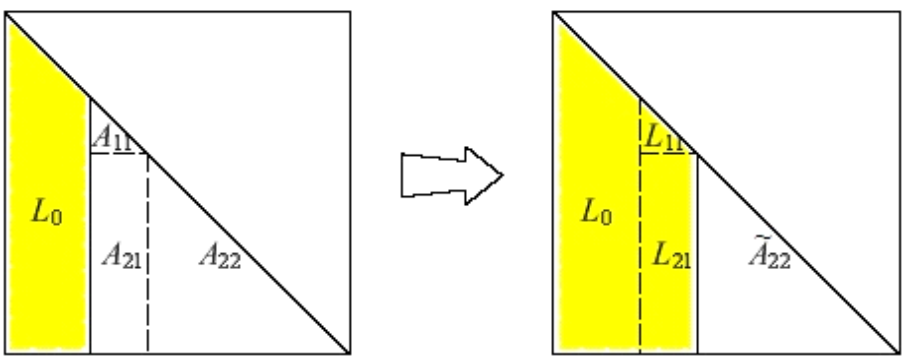


Figure 2.2: Scheme of implementation of the block algorithm for LL^T −−factorization

In this case the total number of arithmetic operations required for the block LL^T – decomposition (2.7) is estimated by the value $O_1 = n^3/3 + O(n^2)$ similarly as in the case of non-block version.

Distribution of data and results. Two-dimensional block-cyclic distribution of matrix elements between processes is used in the solving of LAS with dense non-singular matrices by parallel block algorithms.

For the distribution of original n-th order matrix A between p processes it should be represented in the form:

$$A = \begin{pmatrix} A_{11} & A_{12} & \ldots & A_{1p} \\ A_{21} & A_{22} & \ldots & A_{2p} \\ \ldots & \ldots & \ldots & \ldots \\ A_{q1} & A_{q2} & \ldots & A_{qp} \end{pmatrix},$$

where:

$$A_{ij} = \begin{pmatrix} A_{ij} & A_{ij+p} & \dots & A_{ij+(p-1)p} \\ A_{i+qj} & A_{i+qj+p} & \dots & A_{i+qj+(p-1)p} \\ \dots & \dots & \dots & \dots \\ A_{i+(q-1)qj} & A_{i+(q-1)qj+p} & \dots & A_{i+(q-1)qj+(p-1)p} \end{pmatrix}.$$

Here A_{ij} is a block of elements of the matrix A of order s.

Matrix of the system is distributed between processes so that $A_{ij} \in P_{ij}$, where i and j are Cartesian coordinates of process on the two-dimensional $p \times q$ grid. Let us illustrate such distribution by the following example.

Let n=8, s=2, p=2, q=2, then for the matrix:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{18} \\ a_{21} & a_{22} & \dots & a_{28} \\ \dots & \dots & \dots & \dots \\ a_{81} & a_{82} & \dots & a_{88} \end{pmatrix},$$

its block representation has a form: $A = \begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{pmatrix}$,

where $A_{ij} = \begin{pmatrix} a_{2i-1,2j-1} & a_{2i-1,2j} \\ a_{2i,2j-1} & a_{2i,2j} \end{pmatrix}$.

In this case block-cyclic distribution for the matrix A takes the form:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix},$$

where

$$A_{11} = \begin{pmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{pmatrix},\ A_{12} = \begin{pmatrix} a_{12} & a_{14} \\ a_{32} & a_{34} \end{pmatrix},\ A_{21} = \begin{pmatrix} a_{21} & a_{23} \\ a_{41} & a_{43} \end{pmatrix},\ A_{22} = \begin{pmatrix} a_{22} & a_{24} \\ a_{42} & a_{44} \end{pmatrix}.$$

Thus, each process P_{ij} contains elements of the matrix A^*_{ij} as well as corresponding blocks of matrix of the right-hand sides.

Each vector of right-hand sides is distributed cyclically by $p \times q$ blocks between processes of the first column of processes' grid. Hence, process P_{k1} contains elements numbered ks, $ks+1$, $ks+2$, ..., $ks+s-1$, $(k+p)s$, $(k+p)s+1$, $(k+p)s+2$, ..., $(k+p)s+s-1$, ..., where s is the number of blocks equal to its counterpart in the matrix distribution. Without loss of generality, let us assume that $\frac{n}{pqs}$ is an integer.

Two-dimensional block-cyclic parallel algorithm for LU-facto- rization. Consider the implementation of this algorithm at the k-th step. Taking into account the fact that algorithm is determined by formulas (2.16), (2.17), let us draw reader's attention to the data exchanges between processes. A process which contains block A_{11} will be called a leading process of the k-th step. Then algorithm is implemented as follows:

1. in the s-step cycle the Gauss transformation over blocks A_{11} and A_{12} is performed, i.e. maximum element in sought in the leading column; exchange by elements of the leading row and row containing maximum elements is carried out; the leading and all subsequent rows are transformed according to (2.2); as a result of transformation, blocks A_{11} and A_{12} are located on the place of blocks L_{11} and L_{12} , respectively,

2. the matrix $(L_{11})^{-1}$ is evaluated (since L_{11} is completely contained in the leading process, $(L_{11})^{-1}$ is evaluated within the limits of one process); then $(L_{11})^{-1}$ is broadcasted to all processes of the corresponding row of the process' grid, i.e. if P_{ij} is the leading process then $(L_{11})^{-1}$ is sent to processes P_{il}, $l = 1, p$,

3. processes independently on each other evaluate a product of the obtained block $(L_{11})^{-1}$ by blocks A_{12} distributed to them according to (2.3); as a result elements of A_{12} are replaced by elements of U_{12},

4. blocks L_{21} are broadcasted horizontally over the rows of the processes' grid, while U_{12} are broadcasted vertically over columns. After this a transformation of the corresponding blocks of matrix A_{22} according to (2.4) takes place in every process.

Two-dimensional block-cyclic parallel algorithm for LL^T-factorization. At each step of algorithm the following sequence of operations is to be performed:

1. process P_{ij} containing block A_{11} carries out decomposition of this block thus evaluating L_{11} (see 2.2.1),

2. process P_{ij} broadcasts the block L_{11} to processes P_{mj}, $m = 1, q$, i.e. carries out vertical broadcasting over grid of processes,

3. processes P_{mj}, $m = 1, q$ independently on each other modify corresponding parts of sub-matrix L_{21} according to $\tilde{L}_{21} = L_{21} \cdot \left(L_{11}^T\right)^{-1}$,

4. processes P_{mj}, $m = 1, q$ broadcast corresponding blocks of matrix $\tilde{L}_{21}$ to all processes in horizontal direction over grid of processes and perform their transposing so that all process receive all parts of sub-matrices $\tilde{L}_{21}$, $\tilde{L}_{21}^T$, required for the forming of product $\tilde{L}_{21}L_{21}^T$,

5. all processes simultaneously perform modification of their parts of sub-matrix A_{21}: $\tilde{A}_{21} = A_{21} - \tilde{L}_{21}\tilde{L}_{21}^T$.

Efficiency of two-dimensional block-cyclic parallel algorithms. Coefficients of acceleration and efficiency of algorithms will be determined according to formulas $S_{pq} = \frac{T_1}{T_{pq}}$, $E_{pq} = \frac{S}{pq}$,where pq is the number of processes; T_1 is the time required for the performing of algorithm on one process; T_{pq} is the time required for the performing of algorithm on the grid $(p \times q)$ of processes. To determine T_{pq} let us make use of formula $T_{pq} = Nt + Mt_o + Qt_c$, where N is the number of arithmetic operations (addition and multiplication); M is the number of exchanges; Q is the number of inter-processes synchronizations.

For two-dimensional block-cyclic parallel algorithm for LU – factorization the number of arithmetic operations is mainly determined by stages 3 and 4 of the algorithm and is estimated by value $N \approx 2n^3/(3pq)$.

To evaluate coefficients of acceleration and efficiency for this algorithm it's necessary to determine basic components for the amount of processes performed at the k-th step of algorithm. The broadcasting of matrices $(L_{11})^{-1}$, L_{21}, U_{12} requires $(p-1)s^2$ and $((p-1)(n-ks)/q+(q-1)(\ n-ks)/p)s$ transfers of elements. Then after k steps we get $M \approx n^2(p+q)(p+q-1)/(2pq\ )$. In so doing at each step $Q = 4n(p+q-2)/s$ synchronizations is performed. Thus, we have:

$$T_{pq} \approx \frac{2n^3 t}{3pq} + \frac{n^2(p+q)(p+q-1)}{2pq} t_{\mathrm{o}} + \frac{4n(p+q-2)}{s} t_{\mathrm{c}}\,.$$

Hence, for coefficients of acceleration and efficiency we get:

$$S_{pq} = \frac{T_1}{T_{pq}} \approx pq\left(1 + \frac{3(p+q)(p+q-1)}{4n}\tau_{\text{o}} + \frac{6pq(p+q-2)}{sn^2}\tau_{\text{c}}\right)^{-1},$$
$$E_{pq} = \frac{S}{pq} \approx \left(1 + \frac{3(p+q)(p+q-1)}{4n}\tau_{\text{o}} + \frac{6pq(p+q-2)}{sn^2}\tau_{\text{c}}\right)^{-1}.$$

Let us analyze coefficients of acceleration and efficiency for two-dimensional block-cyclic parallel algorithm for LL^T – factorization. Principal term in the expression of the number of arithmetic operations for this algorithm is determined mainly by implementation of the last stage and with taking into account the symmetry of the matrix, it can be written as follows:

$$N \approx \sum_{k=1}^{n/s}\left(\frac{(n-ks)^2}{pq}s\right) \approx \frac{n^3}{3pq}.$$

A considerable volume of exchanges is performed at the stage 4:

$$M \approx \frac{1}{2}\sum_{k=1}^{n/s}\left((p-1)\frac{(n-ks)}{q} + (q-1)\frac{(n-ks)}{p}\right)s \approx \frac{n^2(p^2+q^2)}{4pq}.$$

The number of synchronizations is determined by value $Q \approx 2n(p+2q-2)/s$.

Thus, we have:

$$T_{pq} = \frac{n^3 t}{3pq} + \frac{n^2(p^2+q^2)}{4pq}t_{\text{o}} + \frac{2n(p+2q-2)}{s}t_{\text{c}}.$$

Hence, for coefficients of acceleration and efficiency we get estimates:

$$S_{pq} = \frac{T_1}{T_{pq}} = pq\left(1 + \frac{3(p^2+q^2)}{4n}\tau_{\text{o}} + \frac{6pq(p+2q-2)}{sn^2}\tau_{\text{c}}\right)^{-1},$$

$$E_{pq} = \frac{S}{pq} = \left(1 + \frac{3(p^2+q^2)}{4n}\tau_{\text{o}} + \frac{6pq(p+2q-2)}{sn^2}\tau_{\text{c}}\right)^{-1}.$$

Two-dimensional block-cyclic algorithms presented here possess both well balancing of processes' loading and possibility of the efficient control over

the using of various levels memory. Considerable improvement in time required for the performing of algorithms takes place with such choice of blocks' dimension that could be completely contained in the computer's cache memory.

As the number of processes is increases, acceleration of algorithms grows. The obtained dependencies of coefficients of acceleration and efficiency both on the matrix dimension and parameters of the process grid indicate that not always the increase in the number of processes leads to decrease in problems' execution times. So, in case of relatively small dimension of matrix the inter-process data exchanges take much of time required for the execution of algorithm.

The obtained formulas make it possible to draw a conclusion on the dependence of algorithms' efficiency on the dimension of process grid. So, with the same total number of processes the sum $p + q$ may take different value. Obviously for the sake of greater efficiency it is advisable to choose such dimension of grid for which this sum is minimal.

2.3 Algebraic eigenvalue problem

2.3.1 Methods for the solving of algebraic eigenvalue problem (AEVP) with symmetric matrices

This paragraph deals with methods for the solving of problem:

$$Ax = \lambda x\,, \tag{2.19}$$

where A – is n –th order real square matrices,

– full standard AEVP with tri-diagonal symmetric matrix A,

– full standard AEVP with dense symmetric matrix A.

To evaluate all eigenvalues and eigenvectors of ***real tri-diagonal symmetric matrix*** the QL-method with implicit shift is employed which for $T_1 \equiv A$ and $s = 1, 2, \ldots$ is determined by following relations [2]:

$$Q_s(T_s - k_s I) = L_s, \quad T_{s+1} = L_s Q_s^T + k_s I = Q_s T_s Q_s^T\,, \tag{2.20}$$

where Q_s is an orthogonal and L_s is a lower triangular matrices; k_s is a shift corresponding to the least in module eigenvalue of the 2×2 leading block of the matrix T_s. Matrix Q_s is usually a product of matrices of elementary plane rotations:

$$Q_s^T = P_1^{(s)} P_2^{(s)} \cdots P_{n-1}^{(s)} \quad (s = 1, 2, \ldots) \; . \tag{2.21}$$

The iterative process continues until absolute values of all off-diagonal elements become less than the given quantity. Then diagonal elements will be approximate eigenvalues. Columns of the products of transposed matrices Q_s of (2.22):

$$Z = Q \cdot Q_1^T Q_2^T \cdots Q_s^T \cdots \; , \tag{2.22}$$

will be approximate eigenvectors, here $Q \equiv I$, if $T_1 \equiv A$ or is determined from $A = QT_1Q^T$.

During the evaluating of all eigenvalues and eigenvectors of ***real dense symmetric matrix*** the following three items could be mentioned:

1. reduction of the original symmetric matrix to tri-diagonal symmetric form,

2. evaluation of all eigenvalues (coinciding with eigenvalues of the original matrix) and eigenvectors of tri-diagonal symmetric matrix by the above described QL-method with implicit shift,

3. evaluation of eigenvectors of the original matrix.

The reduction of the original symmetric matrix $A^{(0)} \equiv A$ to tri-diagonal symmetric matrix T_1 enables to replace the evaluation of eigenvalues of arbitrary symmetric matrix by evaluation of eigenvalues of tri-diagonal matrix. There exists the sufficient number of practicable procedures for the reduction of matrix to tri-diagonal form.

For reduction of the original symmetric matrix to tri-diagonal symmetric form the Householder's transformations may be used and $n-2$ two-sided elementary transformation of reflection are required:

$$A^{(i)} = P^{(i)} A^{(i-1)} P^{(i)} \qquad (i = 1, 2, ..., n-2) \, , \tag{2.23}$$

where orthogonal matices $P(i) = I + s_i\, u_i\, u_i\, T$ while vectors u_i and factors s_i are determined so that for each $i = 1, 2,..., n-2$ the following conditions hold:

$$a^{(i)}_{n-i+1,j} = a^{(i)}_{j,n-i+1} = 0 \qquad (j = 1, \ldots, n-i-1)\,, \tag{2.24}$$

or (by analogy with reduction of rectangular matrix to two-diagonal form):

$$a^{(i)}_{i,j} = a^{(i)}_{j,i} = 0 \qquad (j = i+2, \ldots, n)\,. \tag{2.25}$$

Then $T_1 \equiv A^{(n-2)}$.

The evaluation of all eigenvectors of the original matrix (dense symmetric) is carried out by accumulating of elementary transformations of reflection:

$$Q = P^{(3)} P^{(2)} \ldots P^{(n-2)}\,, \tag{2.26}$$

and rotation (2.22). Then columns of the matrix $X = QZ$ are approximate eigenvectors of the original matrix.

2.3.2 Parallel algorithms for solving of algebraic eigenvalue problem

Analysis of methods intended for the solving of problems presented in paragraph 2.3.1 has shown that the following parallel algorithms are required for the solving of these problems on parallel computers:

1. reduction of dense symmetric matrix to tri-diagonal symmetric matrix by means of two-sided elementary transformations of reflection (2.23),

2. accumulation of elementary transformations of rotation (2.26),

3. QL – method with implicit shift (2.20) and accumulation of elementary transformations of rotation (2.21), (2.22) for the solving of full AEVP with tri-diagonal symmetric matrix.

Row-cyclic parallel algorithm for the reduction of dense symmetric matrix to the tri-diagonal symmetric matrix.

This algorithm of the Householder's method is similar to row-cyclic algorithm for the reduction of rectangular matrix to two-diagonal form described in section 2.2. Here we are going to deal with algorithm implementing another version of reduction starting from the lower right-hand corner. In such a situation orthogonal matrices $P^{(i)}$ of elementary transformations of reflection (2.23) are formed so that conditions (2.24) hold.

Two-sided transformations (2.23) together with conditions (2.24) can be written as follows:

$$A^{(i)} = A^{(i-1)} + u_i v_i^T + v_i u_i^T \qquad (i \;=\; 1,\; 2, \ldots,\; n-2)\,, \tag{2.27}$$

where

$$\begin{aligned}
u_i &= \left(a_{k,1}^{(i-1)}, \ldots, a_{k,k-2}^{(i-1)}, a_{k,k-1}^{(i-1)} - e_k, 0, \ldots, 0\right)^T, \quad v_i = w_i + c_i u_i\,, \\
e_k &= -\mathrm{sign}(a_{k,k-1}^{(i-1)})\sigma_i, \qquad w_i = s_i A^{(i-1)} u_i, \qquad c_i = 0,5 s_i w_i^T u_i, \\
s_i &= \left(e_k a_{k,k-1}^{(i-1)} - \sigma_i^2\right)^{-1}, \quad \sigma_i^2 = \sum_{j=1}^{k-1} \left(a_{k,j}^{(i-1)}\right)^2, \quad (k = n - i + 1)\,.
\end{aligned} \tag{2.28}$$

Thus, on each step (for each i) a square sub-matrix $A^{(i-1)}$ of order $n-i$ is modified; this matrix includes element $a_{1,1}^{(i-1)}$ and this enables condition (2.24) to hold, while elements $a_{n-i,n-i}^{(i)}$ and $a_{n-i+1,n-i}^{(i)} = a_{n-i,n-i+1}^{(i)}$ are, respectively, diagonal and off-diagonal elements of tri-diagonal matrix $T_1 \equiv A^{(n-2)}$. As transformations are carried out, the symmetry of the original matrix A and all matrices $A^{(i)}$ is taken into consideration.

Distribution of data and results. Despite the fact that original matrix A is symmetric and for the performing of transformations (2.27), (2.28) suffice it to have elements both of principal diagonal of the matrix and, for example, of its lower triangle, but it is advisable to specify all matrix elements. In so doing computations are arranged so that only elements of lower triangle of the matrix are modified and the rest of elements may be used, for example, for the evaluation of residual.

For the arrangement of parallel computations matrices $A^{(i)}$, including the original matrix $A^{(0)}$, are distributed between processes by row-cyclic scheme:

elements of matrix rows numbered $k,\ k+p,\ k+2p,\ ...$, are located in process with logical number $k-1$, where $k = 1, 2, \ldots, p$, p is the number of processes. The result of reduction – diagonal $a_{j,j}^{(n-2)}$ $(j = 1, 2, \ldots, n)$ and off-diagonal $a_{j,j+1}^{(n-2)} \equiv a_{j+1,j}^{(n-2)}$ $(j = 1, 2, \ldots, n-1)$ elements of tri-diagonal matrix $T_1 \equiv A^{(n-2)}$ – are evaluated by each process. Besides, non-zero elements u_i are located on the position of elements of lower triangle of the original matrix according to their inter-process distribution. To perform both intermediate computations according to formulas (2.28) and inter-process data exchanges three arrays for the locating of vectors u_i, v_i and w_i of (2.27) are required.

Algorithm. Row-cyclic parallel Householder's algorithm for the reduction of dense symmetric matrix to tri-diagonal symmetric matrix for $i = 1, 2, \ldots, n-2$ consists of the following sequence of operations:

1. (a) by means of broadcasting operation one-dimensional array of elements $a_{k,1}^{(i-1)}, \ \ldots\ , a_{k,k-1}^{(i-1)}, \ a_{k,k}^{(i-1)}$ of the k-th matrix row is sent to all processes ($k = n - i + 1$, definition of broadcasting operator see in Introduction),

   (b) all processes simultaneously evaluate values σ_i^2 and $e_k \equiv a_{k,k-1}^{(i)}$ of (2.28), form the reflection vector u_i and then evaluate value s_i of (2.28),

   (c) each process evaluates the array of the first $n-i$ elements of vector of partial sums of the product $A^{(i-1)}u_i$ of symmetric matrix given both by lower triangle and elements of principal diagonal by row-cyclic scheme by vector; simultaneously each process performs multi-gathering of array of the first $n-I$ components of the vector $A^{(i-1)}u_i$ (definition of multi-gathering operation for the array of numbers is given in Introduction),

2. all process simultaneously form the first $n-i$ elements of the vector w_i according to (2.26),

3. all processes simultaneously evaluate the value c_i and form the first $n-i$ elements of the vector v_i according to (2.28),

4. each process according to row-cyclic scheme of distribution of elements of the matrix $A^{(i-1)}$ performs modification (2.27) of the lower triangle of its sub-matrix of order $n-I$ which contains element $a_{1,1}^{(i-1)}$.

After performing of all these operations for all $i = 1, 2, \ldots, n-2$ elements $a_{2,1}^{(n-2)}$, $a_{2,2}^{(n-2)}$ and $a_{1,1}^{(n-2)}$are to be sent to all processes. Thus, after performing of all operations involved in algorithm under consideration each process will form two n-dimensional vectors of diagonal and off-diagonal elements of tri-diagonal symmetric matrix $A^{(i-2)}$.

Efficiency of algorithm. The total number of arithmetic operations required for the reduction of dense symmetric matrix to tri-diagonal symmetric matrix by Householder's method is estimated by value:

$$O_1 \approx 4n^3/3\,, \tag{2.29}$$

while the number of arithmetic operations performed by each of p processes – by value:

$$O_p \approx \frac{4n^3 + 12n^2 p}{3p}.$$

On each step of i – cycle two operations of broadcasting of one-dimensional arrays of $n-i$ elements each, and one operation of multi-gathering of $(n-i)$-dimensional vector are performed. Hence, the total number of exchanges is estimated by value $O_c \approx 3n \log_2 p$ and at that the amount of data by which processes exchange constitutes approximately $O_o \approx 1.5n^2 \log_2 p$ double words.

Then coefficients of acceleration and efficiency of row-cyclic parallel algorithm for the reduction of dense symmetric matrix to tri-diagonal symmetric matrix are estimated as follows:

$$S_p \approx p\left(1 + \frac{3p}{n} + \frac{1.125p\log_2 p}{n}\tau_1\right)^{-1}, \qquad E_p = \frac{S_p}{p}, \tag{2.30}$$

where $\tau_1 = \tau_o + \frac{2}{n}\tau_c$. If $\frac{3p}{n} + \frac{1{,}125p\log_2 p}{n}\,\tau_1 >> 1$ then:

$$E_p \approx 1 - \frac{3p}{n} - \frac{1,125p\log_2 p}{n}\,\tau_1\,. \tag{2.31}$$

Block cyclic parallel algorithm for the reduction of dense symmetric matrix to tri-diagonal symmetric matrix.

Besides factors taken into consideration in (2.31), the efficiency of algorithm for the reduction of dense symmetric matrix to tri-diagonal symmetric ma-

trix is considerably affected by arrangement of work with operating memory. In up-to-date processors this memory is of the complicated architecture and rate of addressing to the various-levels operating memory (reading or writing) considerably differs. At the same time during the modification of matrix $A^{(i-1)}$ (2.27) very often we have to address the slowest basic operating memory that results in considerable increase in the execution times. This problem can be resolved by modification of algorithm – the using of block performing of transformations (2.27).

Consider a version of block algorithm where orthogonal matrices $P^{(i)}$ of elementary transformations of reflection (2.23) are formed so that conditions (2.25) hold. Then (in contrast to (2.28)) the reflection vector u_i and scalar values σ_i, s_i are formed differently. Thus, instead of (2.28) we have:

$$u_i = \left(0, \ldots, 0, a_{i,i+1}^{(i-1)} - e_{i+1}, a_{i,i+2}^{(i-1)}, \ldots, a_{i,n}^{(i-1)}\right)^T, \quad v_i = w_i + c_i u_i\,, \quad (2.32)$$

$$e_{i+1} = -\mathrm{sign}(a_{i,i+1}^{(i-1)})\sigma_i, \qquad w_i = s_i A^{(i-1)} u_i, \qquad c_i = 0.5 s_i w_i^T u_i\,, \quad (2.33)$$

$$s_i = \left(e_{i+1} a_{i,i+1}^{(i-1)} - \sigma_i^2\right)^{-1}, \quad \sigma_i^2 = \sum_{j=i+1}^{n} \left(a_{i,j}^{(i-1)}\right)^2. \quad (2.34)$$

In the block version of Householder's method for the reduction of dense symmetric matrix to tri-diagonal form according to [2] instead of using both of the reflection vector u_i and vector v_i the rectangular matrices both of vectors u_i and vectors v_i are used in (2.27). These rectangular matrices are formed as follows:

$$\begin{aligned} U_I^{(1)} &= u_{I-s+1}, \quad U_I^{(r)} = \left(U_I^{(r-1)}, u_{I-s+r}\right), \\ V_I^{(1)} &= v_{I-s+1}, \quad V_I^{(r)} = \left(V_I^{(r-1)}, v_{I-s+r}\right), \end{aligned} \quad (2.35)$$

where $r = 2, \ldots, s$, s – size of block. Now the matrix transformations are carried out as follows:

$$\begin{aligned} A^I &= A^{(I-s)} + U_I^{(s)} V_I^{(s)^T} + V_I^{(s)} U_I^{(s)^T}, \\ A^{(n-2)} &= A^{(N)} + U_{N+s}^{(n-N-2)} V_{N+s}^{(n-N-2)^T} + V_{N+s}^{(n-N-2)} U_{N+s}^{(n-N-2)^T}, \end{aligned} \quad (2.36)$$

where $I = s, 2s, \ldots, N, n-2, N = Ts, T = (n-3)/s$ (here the value of a is equal to integer part of quantity a). In this case, similarly as in row-cyclic version, in fact only $n-I$-th order of the matrix $A^{(I-s)}$ is transformed which contains matrix element $a_{n,n}^{(I-s)}$. During the evaluating of reflection vectors u_i and vectors w_i the following representation of the matrix $A^{(i)}$ $(I-s < i \leq I)$ is used:

$$A^{(i)} = A^{(I-s)} + U_I^{(r)} V_I^{(r)^T} + V_I^{(r)} U_I^{(r)^T} \qquad (r = i + s - I)\,. \tag{2.37}$$

In so doing the direct transformation of the entire matrix is not carried out, and only necessary elements are evaluated for example vector $A^{(i-1)}u_i$ is evaluated as follows:

$$A^{(i-1)}u_i = A^{(I-s)}u_i + U_I^{(r-1)} \left(V_I^{(r-1)^T} u_i \right) + V_I^{(r-1)} \left(U_I^{(r-1)^T} u_i \right)\,. \tag{2.38}$$

where: $(r > 1)$.

Distribution of data and results. Despite the fact that, similarly as in the case or row-cyclic algorithm, suffice it to have elements of principal diagonal of the matrix and, for example, elements of its upper triangle for the performing of transformations (2.33)-(2.38) still it is advisable to preassign all elements of the matrix but only elements of upper triangle are to be modified. The row-cyclic scheme of distribution matrices A(I) between processes including the matrix A(0) is also employed here for the arrangement of parallel computations. Similarly as in non-block algorithm the results of reduction are diagonal $a_{j,j}^{(n-2)}$ (j = 1, 2,. . . , n) and off-diagonal $a_{j,j+1}^{(n-2)} \equiv a_{j+1,j}^{(n-2)}$ (j = 1, 2,. . . , n−1) elements of the tri-diagonal matrix T1≡A(n−2) – are computed by each process, while non-zero elements of vectors u_i are stored on the places of elements of upper triangle of the original matrix according to their distribution between processes.

For the performing of the intermediate computations by formulas (2.33), forming both of the rectangular matrix of reflection $U_I^{(s)}$ and rectangular matrix $V_I^{(s)}$, (2.35) as well As data exchanges between processes, each process requires two arrays to locate vectors u_i, w_i of (2.33) and two arrays for the storing of rectangular matrices $U_I^{(s)}$, $V_I^{(s)}$.

Algorithm. During the reduction of dense symmetric matrix to tri-diagonal symmetric matrix by means of block-cyclic parallel Householder's algorithm for each $I = s, 2s, \ldots, N, n-2$:

1. first, each process forms rectangular matrices $U_I^{(s)}$ and $V_I^{(s)}$ (2.35),
2. further each process according to distribution scheme of elements of the matrix $A^{(I-s)}$ performs modification (2.36) of the upper triangle of its $n-I$ order sub-matrix which contains element $a_{n,n}^{(I-s)}$.

During the forming of rectangular matrices $U_I^{(s)}$ and $V_I^{(s)}$ (2.35) for each $i = I-s+1, \ldots, I$ the following sequence of operations ($I = s, 2s, \ldots, N$, $n-2$) is performed enabling to form vectors u_i and v_i:

1. by means of broadcastings operation the one-dimensional array of elements $a_{i,i}^{(i-1)}$, $a_{i,i+1}^{(i-1)}$, $\ldots$, $a_{i,n}^{(i-1)}$ of the i-th matrix row is sent to all process (the definition of the broadcasting operation is given in the Introduction),
2. all processes simultaneously compute values σ_i^2 and $e_i \equiv a_{i,i+1}^{(i)}$ of (2.33), form the reflection vector u_i and then compute the value s_i of (2.33),
   (a) each process computes an array of last $n-i$ elements of the vector of partial sums of the product $A^{(i-1)}u_i$ of symmetric matrix given both by upper triangle and elements of principal diagonal according to row-cyclic scheme by vector according to formula (2.38),
3. all processes simultaneously perform the multi-gathering of array containing last $n-i$ components of vector $A^{(i-1)}u_i$ (definition of multi-gathering operation for the array of numbers is given in the Introduction),
4. simultaneously all processes compute the last $n-i$ elements of vector w_i according to (2.33),
5. all processes simultaneously compute the value c_i and form last $n-i$ elements of the vector v_i according to (2.33),
6. a process containing elements of the $(i+1)$-th matrix row compute elements $a_{i+1,i+1}^{(i)}$, $\ldots$, $a_{i+1,n}^{(i)}$ if $i < I$ with taking into account (2.37).

Similarly as in row-cyclic algorithm, elements $a_{n-1,n-1}^{(n-2)}$, $a_{n-1,n}^{(n-2)}$ and $a_{n,n}^{(n-2)}$ computed according to the distribution scheme should be broadcast to all processes.

Efficiency of algorithm. As noted above, the total number of arithmetic operations required for the reduction of dense symmetric matrix to tridiagonal symmetric matrix by Householder's method is estimated by value $O_1 \approx 4n^3/3$. The number of arithmetic operation of block cyclic algorithm performed by each of p processes is estimated by quantity:

$$O_p \approx \frac{4n^3 + 6n^2 ps}{3p}\,.$$

As to algorithm dealt with, similarly as for the row-cyclic algorithm, the total number of exchanges is estimated by $O_c \approx 3n \log_2 p$, while total amount of data by which processes involved in computations exchange constitutes approximately $O_o \approx 1.5n^2 \log_2 p$ double words.

Then coefficients of acceleration and efficiency of block cyclic parallel algorithm for the reduction of dense symmetric matrix to tri-diagonal symmetric matrix are estimated as follows:

$$S_p \approx p\left(1 + \frac{1,5ps}{n} + \frac{1,125p\log_2 p}{n}\,\tau_1\right)^{-1}, \qquad E_p = \frac{S_p}{p}\,, \tag{2.39}$$

where $\tau_1 = \tau_> + \frac{2}{n}\,\tau_{\rm A}$. If $\frac{1,5ps}{n} + \frac{1,125p\log_2 p}{n}\,\tau_1 >> 1$, then:

$$E_p \approx 1 - \frac{1,5ps}{n} - \frac{1,125p\log_2 p}{n}\,\tau_1\,. \tag{2.40}$$

The using of one-dimensional block cyclic scheme, i.e. distribution of q rows in succession to each process instead of using the row-cyclic data distribution scheme decreases the number of exchanges but increases non-balancing of processors' loading.

Row-cyclic parallel algorithm for the accumulation of elementary reflection transformations.

The forming of matrix of elementary reflection transformations Q (2.26) can be carried out as follows:

$$Q_{(i)} = P^{(k)} Q_{(i-1)} \equiv Q_{(i-1)} + u_k w_k^T \qquad (i = 1,\ 2, \ldots,\ n-2)\,, \tag{2.41}$$

where $Q_{(0)} \equiv I, \qquad Q_{(0)} \equiv Q_{(n-2)},$

$$w_k = s_k Q^T_{(i-1)} u_k, \qquad s_k = \left(e_{i+2} u_{k,i+1}\right)^{-1} . \tag{2.42}$$

The reflection vector u_k is determined in (2.28), $e_{i+1} \equiv t_{i+1,i}$ is an off-diagonal element of the tri-diagonal matrix $T_1 \equiv A^{(n-2)}$, $k = n-i-1$. Thus, a process of forming of the matrix of elementary refection transformations (2.26), (2.41), (2.42) is analogues to the process of forming of matrices of the left- or right-hand elementary reflection transformations which is described in section 2.3.1, but the process starts from the left-hand upper corner of the matrix.

Distribution of data and results. Array of off-diagonal elements of matrix T_1 and reflection vectors u_k formed during the reduction of symmetric matrix to tri-diagonal form are employed as initial data in the forming of matrix Q (2.37), (2.38). Non-zero elements of these vectors are stored on the position of elements of lower triangle of the original matrix according to their row-cyclic distribution between processes. In so doing elements of vector u_k are placed on the position of elements of $(i+2)$-th row of the original matrix.

The result of computations – square matrix Q – is distributed by between processes by row-cyclic scheme. It can be located either the position of the original matrix or separately if the original matrix is required for further computations.

For the carrying out intermediate computations by formulas (2.37), (2.38) and data exchanges between processes three arrays for storing vectors are required: one array for u_k and two arrays for w_k.

Algorithm. Row-cyclic algorithm for the accumulating of elementary reflection transformations for $i = 1, 2, \ldots, n-2$ consists of the following sequence of operations:

1. by means of the broadcasting operation one-dimensional array consisting of $i+1$ non-zero elements of the reflection vector u_k is sent to all processes ($k = n-i-1$, definition of the broadcasting operation is given in the Introduction),

2. all processes simultaneously compute the value s_k of (2.38),

3. each process computes the array of the first $(i+1)$ elements of vector of partial sums of the product $Q^T_{(i-1)}u_k$ of the square matrix given by column cyclic scheme by vector,

4. simultaneously each process performs multi-gathering of array of the first $i+1$ components of the vector $Q^T_{(i-1)}u_k$ (definition of multi-gathering operation of the array of numbers is given in the Introduction),

5. all processes simultaneously form the first $i+1$ elements of vector w_k according to (2.39),

6. each process according to the row-cyclic scheme of distribution of elements of the matrix $Q_{(i-1)}$ performs modification (2.37) of its square sub-matrix of order $i+1$ which contains element $q^{(i-1)}_{1,1}$.

Efficiency of algorithm. The total number of arithmetic operations required for the forming of matrix by elementary reflection transformations Q (2.36) is estimated by value:

$$O_1 \approx 4n^3/3\,, \tag{2.43}$$

while the number of arithmetic operations performed by each of p processes is estimated by value:

$$O_p \approx \frac{4n^3 + 1.5n^2p}{3p}\,. \tag{2.44}$$

On each step of the i-cycle the following operations are performed*:* two operations of broadcasting of one-dimensional arrays, each consisting of $i+1$ elements and one multi-gathering operation of the $(i+1)$-dimensional array. Hence, the total number of exchanges is estimated by value $O_c \approx 3n\log_2 p$, and at that the total amount of data by which processes exchange constitutes approximately $O_o \approx 1.5n^2\log_2 p$ double words.

Then coefficients of acceleration and efficiency of row-cyclic parallel algorithm for the forming of matrix of elementary reflection transformations Q (2.36) are estimated as follows:

$$S_p \approx p\left(1 + \frac{0.375p}{n} + \frac{1.125p\log_2 p}{n}\,\tau_1\right)^{-1}, \qquad E_p = \frac{S_p}{p}\,, \tag{2.45}$$

where $\tau_1 = \tau_o + \frac{2}{n}\tau_c$. If $\frac{0.375p}{n} + \frac{1.125p\log_2 p}{n}\,\tau_1 >> 1$ then:

$$E_p \approx 1 - \frac{0.375p}{n} - \frac{1.125p\log_2 p}{n}\,\tau_1\,. \tag{2.46}$$

Block cyclic parallel algorithm for the accumulating of elementary reflection transformations.

During the forming of matrix Q of elementary reflection transformations (2.49) dealt with in paragraph 2.3.2 computations are carried out by formulas:

$$\begin{gathered} Q_{(t)} = I + U_{n-2}^{(t)} {W_{n-2}^{(t)}}^T, \\ Q_{(I)} = Q_{(I-s)} + U_K^{(s)} {W_K^{(s)}}^T \quad (I = t+s,\ t+2s,\ \ldots,\ n-2)\,, \end{gathered} \tag{2.47}$$

here $Q \equiv Q_{(n-2)}$, $K = n-I+s-1$, $t = n-Ts-1$, $T = (n-3)/s$ (the value of a is equal to the integer part of number a),

$$\begin{aligned} U_K^{(1)} &= u_K, \quad U_K^{(r)} = \left(u_{K-r+1}, U_K^{(r-1)}\right), \\ W_K^{(1)} &= w_K, \quad W_K^{(r)} = \left(w_{K-r+1}, W_K^{(r-1)}\right), \end{aligned} \tag{2.48}$$

$$w_k = s_k Q_{(i-1)}^T u_k, \qquad s_k = \left(e_{k+2} u_{k,k+1}\right)^{-1}, \tag{2.49}$$

$r = 2, \ldots, s$, the reflection vector u_k is determined in (2.33), $e_{i+1} \equiv t_{i+1,i}$ is an off-diagonal element of tri-diagonal matrix $T_1 \equiv A^{(n-2)}$, $k = n-i-1$. In this case, similarly as in the block version of algorithm for the reduction of symmetric matrix to tri-diagonal form the following product $Q_{(i-1)}^T u_k$ is used:

$${Q_{(i-1)}}^T u_k = {Q_{(I-s)}}^T u_k + W_K^{(r-1)} \left({U_K^{(r-1)}}^T u_k\right). \tag{2.50}$$

Distribution of data and results. In the algorithm under consideration the distribution of data and results is similar to that described in the previous paragraph 2.3.2 for row-cyclic algorithm. Difference consists in the following: non-zero elements of the reflection vectors u_k are stored on the place of elements of upper triangle of the original matrix according to their row-cyclic distribution between processes. In so doing elements of vector u_k are located on the place of elements of the k-th row of the original matrix. Another difference is that for the carrying out both of intermediate computations by formulas (2.48)-(2.50) and inter-process data exchanges each process requires two arrays for storing vectors u_i of (2.33) and w_i of (2.49) as well as two arrays for storing rectangular matrices $U_K^{(s)}$, $W_K^{(s)}$.

Algorithm. Initially, in processes, where next to last and last rows of matrix are located, the assignations $q_{n-1,n-1} = q_{n,n} = 1$, $q_{n-1,n} = q_{n,n-1} = 0$ are performed. Further, for the forming of matrix Q (2.26) by block cyclic parallel algorithm for $I = t, t+s, t+2s, \ldots, n-2$:

1. each process forms the rectangular matrices $U_K^{(s)}$ and $W_K^{(s)}$ (2.48),

2. whereupon each process according to distribution scheme for elements of the matrix $Q_{(I)}$ performs modification of its lower right-hand square block of order $I+1$.

Forming (2.48) of rectangular matrices $U_K^{(s)}$ and $W_K^{(s)}$ for each $i = I-s+1, \ldots, I$ (or $i = 1, \ldots, t$, if $I = t$) is performed by means of the following sequence of operating which enables to form vector w_k, ($I = t, t+s, t+2s, \ldots, n-2$):

1. by means of the broadcasting operation an one-dimensional array containing $i+1$ non-zero elements of the reflection vector u_k is sent to all processes ($k = n-i-1$, definition of the broadcasting operation is given in the Introduction),

2. all processes simultaneously compute the value s_k of (2.49),

3. each process computes an array of the last $i+1$ elements of vector of partial sums of the product $Q_{(i-1)}^T u_k$ (2.50) of the square matrix given by column cyclic scheme by vector,

4. each process simultaneously with other processes performs the multi-gathering of the array containing $i+1$ components of the vector $Q_{(i-1)}^T u_k$ (definition of multi-gathering operation of the array of numbers is given in the Introduction),

5. all processes simultaneously form the last $i+1$ elements of vector w_k according to (2.46),

6. according to the distribution scheme of matrix elements the following assignations are performed $q_{k,k} = 1$, $q_{k,j} = q_{j,k} = 0$ where $j = k+1, \ldots, n$.

Efficiency of algorithm. As noted in the previous paragraph, the total number of arithmetic operations required for the forming of matrix Q of elementary reflection transformations (2.26) is estimated by value $O_1 \approx 4n^3/3$. The number of arithmetic operations required for the performing of block cyclic algorithm under consideration by each of p processes is estimated by value:

$$O_p \approx \frac{4n^3 + 1.5n^2 s\,(p+11)}{3p}\,.$$

Similarly, as for row-cyclic algorithm, the total number of exchanges is estimated by value $O_c \approx 3n \log_2 p$, and at that the total amount of data by which processes exchange constitutes approximately $O_o \approx 1.5n^2 \log_2 p$ double words.

Then for coefficients of acceleration and efficiency of the block cyclic parallel algorithm for the forming of matrix of elementary reflection transformations Q (2.51) the following estimates are valid:

$$S_p \approx p\left(1 + \frac{0.375s\,(p+11)}{n} + \frac{1.125p\log_2 p}{n}\,\tau_1\right)^{-1}, \qquad E_p = \frac{S_p}{p}\,, \quad (2.51)$$

where $\tau_1 = \tau_o + \frac{2}{n}\tau_c$. If $\frac{0{,}375s(p+11)}{n} + \frac{1{,}125p\log_2 p}{n}\,\tau_1 >> 1$ then:

$$E_p \approx 1 - \frac{0.375s\,(p+11)}{n} - \frac{1.125p\log_2 p}{n}\,\tau_1\,. \quad (2.52)$$

2.3.3 Parallel QL-algorithm for tri-diagonal real symmetric matrices

QL-method with implicit shift for the evaluating of all eigenvalues of tri-diagonal symmetric matrix is similar to QR-method with implicit shift for evaluating of singular values of upper two-diagonal matrix; this method is dealt with in section 2.3.1. Similarly as QR-algorithm for two-diagonal matrix it is not advisable to parallel QR-algorithm for tri-diagonal matrix due to relations of efficiency; it is advisable that each process completely evaluate all approximate eigenvalues. It is also advisable to form in parallel elements of matrix of eigenvectors according to (2.21), (2.22).

Distribution of data and results. According to the foregoing all diagonal and off-diagonal elements of tri-diagonal symmetric matrix are stored in each

process. Moreover, all evaluated approximate eigenvalues are also stored in each process.

Since each separately taken (elementary) right-hand plane rotation used in the evaluating of matrix of eigenvectors by formulas (2.21), (2.22) consists of in-pair transformations of only two elements belonging to columns being modified) of each row, and parameters of plane rotation are evaluated by each process, elements of matrix of eigenvectors being formed are distributed between processes by row-cyclic scheme. In this case there is no need in inter-process data exchange during the forming of matrices of eigenvectors.

Algorithm. At each iteration $s = 1, 2$, of parallel QR-algorithm the following operations are performed:

1. simultaneously without data exchanges each process computes the value of the shift k_s; in succession form matrices of plane rotations $P_j^{(s)}$ ($j = 1, 2, \ldots, m < n$) and modify tri-diagonal matrix T_i according to (2.20),

2. simultaneously according to the distribution scheme and without data exchanges each process modifies elements of eigenvectors' matrix of eigenvectors being formed,

3. simultaneously without data exchanges each process verifies a criterion for the attaining of the given accuracy in the evaluating of the next approximate eigenvalue.

Modification of matrix of eigenvectors being formed can be carried out immediately after the forming of every matrix of plane rotations $P_j^{(s)}$ or at the end of every iteration. After performing of the next approximate eigenvalue the order of matrix being processed is decreased by one.

Efficiency of algorithm. Parallel algorithm is design so that all computations be carried out without inter-process data exchanges. The number of arithmetic operations performed by each of p processes is approximately p times less than that in computations in mono-process mode. Hence, coefficient of acceleration of parallel QL-algorithm, similarly as in the case of parallel algorithm described in section 2.3.1, is close to p, while coefficient of efficiency – to unity.

2.4 Non-linear equations and systems

2.4.1 Statements of problems with approximate initial data

Problem on finding of real roots of non-linear equation. Let $f(x)$ be continuous in the interval $[a, b]$ function of one real variable x. Numerical problem of finding in the interval $[a, b]$ of real roots of the equation:

$$f\,(x) = 0, \qquad a \le x \le b\,, \tag{2.53}$$

with approximate initial data consists in the separation of roots, their approximate evaluation within the given accuracy and estimating the reliability of results (i. e. estimating of module of deviation $|x - \bar{x}|$ of exact value of root of the equation with approximate initial data from exact value of root of the equation with accurate initial data) under condition $|\bar{f}\,(\bar{x}) - f\,(x)| < \Delta$, where $\bar{f}\,(\bar{x}) = 0$ is a non-linear equation with accurate initial data; Δ is an estimate for error in the specification of function $f\,(x)$.

Problem on finding solutions of SNE. Problems on finding solutions of systems of non-linear equations with approximate initial data are posed as follows: it is required to find a solution of system of n equations:

$$f(x) = 0\,, \tag{2.54}$$

in the given region $G = \{a_i \le x_i \le b_i \ (i{=}1,\, 2,\ldots,n)\}$, where:

$x = (x_1, x_2, \ldots, x_n)^T$, $\quad f\,(x) = (f_1\,(x)\,, f_2\,(x)\,, \ldots, f_n\,(x))^T$; x is an n-dimensional vector being sought, and estimate its reliability under assumption that vector-function $f\ (x)$ satisfies all necessary conditions for the existence of unique solution in the region G and for this region the inequality $\|\,\varphi\ (\nu) - \ f\,(\nu)\,\| \le \Delta$ holds, where $\varphi\,(y) = 0$ is an accurate system of non-linear equations; ν is an arbitrary vector from the region G; and Δ is an estimate for error in formulas for the evaluation of vector-function $f(x)$.

2.4.2 Methods for the solving of SNE

In most cases non-linear equations and systems are solved by iterative methods. In so doing a region is to be specified in which it is required to find the following: a solution, the required value ε which is used in the chosen termination criterion for iterative processes and restriction from above for the number of iterations being performed. The lather is related to the fact that no solution of the system may be in the region under consideration and at that the cycling of the iterative process may happen.

To estimate both an accuracy of problem with approximately given initial data and reliability of the obtained results it is necessary to determine characteristics of equation or system of equations in the neighborhood of solution.

Characteristics of equation in the neighborhood of a root are determined by the value of module of the derivative $\left|f'\left(x^{(k)}\right)\right|$, while characteristics of system of equations in the neighborhood of solution – by norm $\left\|H^{-1(k)}\right\|$ of the matrix inverse to Jacobi matrix which is evaluated by formula $H^{(k)} = H\left(x^{(k)}\right) = \left\{\frac{\partial f_i\left(x^{(k)}\right)}{\partial x_j}\right\}_{i,\,j=1}^{n}$, k is the number of iteration. Tending of the mentioned values to zero indicates that solution may turn out to be non-unique in this neighborhood (in particular, a root of the function may be multiple).

To terminate iterative process for the solving of SNE one should proceed as follows: at first, the holding of condition $\left\|f\left(x^{(k)}\right)\right\| \leq \varepsilon$ is verified and only if at some iteration it holds then a change to the verification of condition:

$$\left\|f\left(x^{(k)}\right)\right\| \leq \frac{\varepsilon}{\left\|H^{-1(k)}\right\|}, \tag{2.55}$$

for systems of equations occurs.

This condition ensures the holding of the inequality $\left\|x^{(k)} - x\right\| \leq \varepsilon$ for systems of equations where x denotes an exact solution of problem (2.54) with approximate data. For one equation it is necessary to verify the condition:

$$\left|f\left(x^{(k)}\right)\right| \leq \varepsilon \left|f'\left(x^{(k)}\right)\right|, \tag{2.56}$$

which ensures the given accuracy of the obtaining of the value of root $\left|x^{(k)} - x\right| \leq$

ε of the equation (2.53) with approximate data.

For approximations obtained in such a manner the following estimates are valid:

$$\left| x^{(k)} - \bar{x} \right| \leq \varepsilon + \frac{\Delta}{\left| f'(x) \right|}. \tag{2.57}$$

For one equation ($\bar{x}$ is an exact root of the accurate equation) and:

$$\left\| x^{(k)} - \bar{x} \right\| \leq \varepsilon + \left\| H^{-1(k)} \right\| \Delta, \tag{2.58}$$

for system of equations ($\bar{x}$ is an exact solution of accurate system of equations). Within estimates (2.57), (2.58) the approximate nature of initial data of the problem being solved and approximate character of the evaluation of solutions by iterative methods are taken into consideration.

For the evaluation of roots of non-linear equation a method for roots' separation with their subsequent finding by bisection method is employed.

For the evaluation of solutions of SNEs the iterative methods are employed based to some extent on classic Newton's method possessing the quadratic rate of convergence.

Iterative process is said to converge if the following estimate:

$$\left\| x^{(k+1)} - \bar{x} \right\| \leq c \left\| x^{(k)} - \bar{x} \right\|^{\alpha},$$

holds, where c is some quantity bounded from above; α is an order of convergence of the method. If $\alpha = 2$ the quadratic rate of convergence of the iterative process is attained, if $1 < \alpha < 2$ then iterative process converges over-linearly.

Parallel algorithms for the solving of SNE by Newton's method as well as its modifications referred to as quasi-Newton's methods are to be dealt with below:

1. Burdakov's method [2] which with special choice of the iterative parameter ensures the global convergence to one of system's solution on the basis of given initial approximation,

2. Dennis-Moore's method [3] in which the Jacobi matrix and its inverse are evaluated on the basis of the initial approximation and then this inverse matrix is corrected by iterative formula within the iterative process,

3. Broyden's method [1] in which the Jacobi matrix is evaluated on the basis initial approximation which is further corrected by iterative formula within the iterative process,

4. Powell's method [4] is used for the solving of SNE with symmetric Jacobi matrix; in this method the Jacobi matrix is evaluated on the basis of initial approximation and further this method is corrected within the iterative process by iterative formula.

Quasi-Newton's methods are known to possess the over-linear rate of convergence. Formulas for the implementation of these methods will be given during the consideration of their corresponding parallel algorithms.

2.4.3 Parallel algorithms for the solving of non-linear equations and systems

Evaluation of roots of one equation. For the numerical solving of equation (2.53) the computer algorithm, which combines steadiness (failure-free work) of the bisection method with asymptotic rate of convergence of secants' method for the case of smooth functions.

Prior to the starting of computations the following data should be specified: the interval [a, b] in which the roots are to be evaluated, function $f(x)$ and error in its specification Δ.

Thereupon the interval [a, b] is to be broken into p parts, where p is the number of processors chosen for the evaluation of roots of the equation; a, b are final points of the given interval. Coordinates of intervals [a_i, b_i] where i = 1, ..., p of the length equal to $h = \frac{b-a}{p}$ are evaluated in the corresponding processes. Then each process simultaneously with others:

1. chooses size of the initial step h_0^i by formula:
   $h_0^i = \frac{0{,}01}{|f(a_i)|}$, $(i = 1,..., p)$,

2. evaluates step size at interval by formula:
   $h_i = \frac{0{,}01 \cdot h_0^i}{|f(b_i) - f(a_i)|} \leq 0{,}001 \ (i = 1,..., p)$,
3. evaluates values of function $f\ (x)$ at intervals' subdivision points $x = a_i + j\, h_i$, where $j = 1,\ldots k$, $k = \left[\frac{b_i - a_i}{h_i}\right]$.

Doing so, it should be taken into account that if at any point x of the interval's $[a_j, b_j]$ subdivision $(j = 1,\ldots,k)$ a value of function $f\ (x)$ is equal to zero, then thus evaluated roots of the equation are written into array of the result.

If, after all, $f(a_j) \cdot f(b_j) < 0 \quad (j = 1, \ldots, k)$ then this interval contains at least one root of the equation.

Further each of p processes independently of other processes performs the following actions in intervals containing at least one root of the equation:

1. computes roots of equations by the same algorithm,
2. writes the computed values of roots into array of results.

Accuracy of the obtained solutions $x^{(k)}$ with respect to exact solutions of accurate equations is estimated by formula (2.57). The evaluation of root in the interval $[a_j, b_j]$ containing odd number roots of the equations is carried out by the following algorithm:

1. the initial approximation is chosen by formula:
$$x^{(0)} = \frac{a_j + b_j}{2}\,,$$
2. then either interval $[a_j, x^{(0)}]$ or $[x^{(0)}, b_j]$ at the ends of which the function $f(x)$ takes values of different signs is chosen and midpoint of this interval is taken as $x^{(0)}$ and so on.

Midpoint of the last interval is taken as an approximate solution of the approximate equation. It should be noted that for such implementation of algorithm for finding roots of equation the coefficient of its efficiency in most cases will be close to 1.

2.4.4 Solving of systems of non-linear equations

During the solving of SNE by any method (mainly the iterative one) the bulk of arithmetic operations falls within the evaluation of values of the vector-function $f(x)$. Therefore during the arrangement of computations on parallel computers the arithmetic operations involved in the evaluation of vector-function $f(x)$ should be paralleled first of all that will enable to parallel the evaluation of the approximation to Jacobi matrix and the solving of the corresponding LAS.

The automatic partitioning of the evaluation of values of components of the system's vector-function by p blocks (p is the number of processes in use) is carried out as follows: the relation $\frac{n}{p} = q$ where a is an integer part of the number a; n is an order of the system; whereupon the quantity $s = p\,(q+1) - n$ is evaluated; then the last s processes will process blocks each containing q equations, while the first $p-s$ processes will process blocks of $(q+1)$ equations each.

Doing so it should be taken into account that in some very seldom cases rules for the evaluation of components of system's vector function differ very much. In these cases systems of equations cannot possess high order, and therefore it is not reasonable to solve them on MIMD-computers. In most cases evaluation of components of the vector-function is subordinated to certain constructional regulations and such systems may be of the sufficiently high order.

Let us describe, as an example, one of possible ways of vector-function's specification in C language.

For example, in order to solve the system:

$$\sum_{j=1}^{n} x_j - 0.5\,(3n+1) + 2x_i^2 - 2\left(1 + 2\frac{i}{n} + \left(\frac{i}{n}\right)^2\right) = 0 \qquad (i = 1,\, 2,\, \ldots,\, n)$$

a program for the evaluation of function values has a form:

```
void f (int n, int l, int m, double *x, double *y)
  { double ss, sss, fv;
  int i, j;
    for( i=l; i<m; i++)
      { ss=0.;
      sss=i+1;
      for( j=0; j<n; j++)   ss+=x[j];
        fv=ss-0.5*(3.*n+1.)+2.*x[i]*x[i]-2.*(3.+2.*sss/n +
        (sss/n)*(sss/n));
        y[i-l] = fv;
      }
  }
```

Here:

1. n denotes an order of system of non-linear algebraic equations,
2. l denotes the initial number of equation in each separately taken process used for the evaluation both of values of the vector-function and corresponding rows of matrix approximating the Jacobi matrix,
3. $m-1$ is a final number of equation in each separately taken process used for the evaluation both of values of the vector-function and corresponding rows of matrix approximating the Jacobi matrix,
4. x denotes a vector of variables,
5. y is a value of vector-function at the point x.

Values l and m for each process are evaluated prior to call of program for the evaluation of values of the vector-function.

If system of non-linear equations is of non-regular structure, then each equation is to be marked by label 'case k:' (k is an integer) and function is to be called according to the above-described rule by means of 'switch (i)' operator. Some other techniques for the specification of vector-function are also possible, but calls of programs for their evaluation should correspond to descriptions of functions as demonstrated in the above example.

After simultaneous computation of values of the corresponding components of vector x processes should exchange the computed values since, as a rule,

all components of vector x are required for the evaluation of values of the vector-function at the next iteration.

Such way for the specification of vector-function possesses the following advantages:

1. as a rule, SNE of high order possess regular structure and it is necessary only to write a loop header and equation depending on the loop parameter,
2. during the implementation of hidden parallelism principle the evaluation of values of the vector-function is automatically distributed between chosen processes,
3. approximation to the Jacobi matrix is also evaluated by blocks and therefore for the solving of LAS of the form (2.54) one of versions of block Gauss method (see section 2.3.1) is employed.

For the solving of SNE by methods to be dealt with below the following information is to be specified:

1. the number of processes,
2. order of system of non-linear equations,
3. $x^{(0)}$ – initial approximation to the solution,
4. lower and upper boundaries of the region where the solution is to be sought,
5. ε – quantity used in termination criterions for the iterative process and determining an accuracy of the obtained approximation,
6. maximal number of iterations to be performed,
7. a subroutine for the evaluation of values of the vector-function.

Newton's method. Generally, the Newton's method, provided that value $x^{(k)}$ is known, is implemented by formulas:

$$x^{(k+1)} = x^{(k)} + w^{(k)} \quad (k = 0,\, 1, \ldots) \;, \tag{2.59}$$

where the correction $w^{(k)} = x^{(k+1)} - x^{(k)}$ is evaluated as a solution of LAS:

$$H^{(k)} w^{(k)} = -\alpha_k f\left(x^{(k)}\right) , \tag{2.60}$$

and $H^{(k)} = H\left(x^{(k)}\right)$ is the Jacobi matrix of SNE; k is the number of iteration. At the beginning the parameter αk is taken to be unity.

Further we'll distinguish between systems of two kinds: system of small order that can be solved in one process and systems of high order – otherwise. However it is advisable to solve systems of small order by Newton's method simultaneously on all processes starting from different initial approximations in order to obtain some set of solutions.

In this case, provided that iterative process starts from the same initial approximations on mono-process and parallel computers, the coefficient of efficiency in most cases will be approximately equal to the unity. For the solving of systems of high order the first $p-s$ processes will compute

$(q + 1)$-th rows of the Jacobi matrix and the same number of components of the vector-function $f\left(x^{(k)}\right)$, while the rest s processes will compute q rows of the Jacobi matrix and the same number of components of the vector-function $f\left(x^{(k)}\right)$.

Then a parallel algorithm for the Newton's method is implemented on the MIMD-computer with the use of p processes according to the following computational scheme:

1. the following data are sent to each process: pre-specified initial approximation $x^{(0)}$, quantity $\varepsilon > 0$ used within the termination criterion for the iterative process; boundaries of region in which the solution is to be sought; and the limiting number of iterations,

2. all processes simultaneously evaluate components of the vector-function $f\left(x^{(0)}\right)$ distributed to them and corresponding rows of the matrix $H\left(x^{(0)}\right)$,

3. at each iteration:

   (a) LAS (2.56) is solved by parallel algorithms intended for its solving (see, for example, section 2.3.1),

(b) the next approximation to the solution $x^{(k+1)}$ $(k = 0, 1,\ldots)$ is evaluated on the basis of solution of LAS,

(c) the gathering of computed components of the next approximation is carried out in order to form a full vector in all processes with simultaneous synchronization of computations,

(d) corresponding components of vector-function $f\left(x^{(k+1)}\right)$ and rows of the matrix $H\left(x^{(k+1)}\right)$ $(k = 0, 1...)$ are evaluated by means of using either analytical form of derivatives (if it is not difficult to evaluate them) or their finite-difference analogs,

(e) the quantities:

$$\varphi_{k+1}^{(j)} = \sum_{i=l}^{m-1} \left(f_i^{(k+1)}\right)^2 \quad (j = 0, 1, \ldots, p-1)\,, \tag{2.61}$$

are evaluated, where j is the logical number of process; l is the initial number of equation in each separately taken process which is used for the evaluation both of values of the vector-function and corresponding rows of matrix approximating the Jacobi matrix; $m-1$ is the final number of equation in each separately taken process which is used for the evaluation of values of the vector-function and corresponding rows of matrix approximating the Jacobi matrix,

(f) for the obtaining of the value of $\varphi_{k+1} = \left\| f\left(x^{(k+1)}\right) \right\|$ all $\varphi_{k+1}^{(j)}$ are summed in some process (for example, in zero one, j is the logical number of process):

$$\varphi_{k+1} = \sqrt{\sum_{j=0}^{p-1} \varphi_{k+1}^{(j)}}\,, \tag{2.62}$$

(g) if $\varphi_{k+1} \geq \varphi_k$ the parameter α_k is halved, values of all components of vector of corrections $w^{(k)}$ are reduced in half, the new value of $x^{(k+1)}$ is evaluated and iterative process is continued from the item 'a',

(h) if $\varphi_{k+1} < \varphi_k$ then the termination criterion for the iterative process $\varphi_{k+1} \leq \varepsilon$ is verified on the starting iterations; if on the certain iteration this condition is fulfilled then formula (2.55) is further employed by each process for the verification of termination criterion for the iterative process; it is this iteration where the criterion is verified for the attaining of the limiting number of iterations the exceeding of which results in termination of the iterative process,

(i) if condition (2.55) is fulfilled the iterative process ends; otherwise it is continued from item 'a'.

Note that verifications (2.55) require the evaluation of matrix inverse to Jacobi matrix. However, since the Jacobi matrix varies only slightly within the neighborhood of solution it is not reasonable to evaluate it on each iteration, suffice it to evaluate this matrix once every several iterations.

After successful termination of the iterative process an error in the obtained solution of the problem with approximate initial data with respect to exact solution of system with accurate initial data is evaluated by formula (2.58).

Let us estimate efficiency of this algorithm. Let it be necessary to perform N_G arithmetic operations for the evaluation of one element of the matrix $H(x)$ and N_f arithmetic operations for the evaluation of the component of the vector-function $f\ (x)$, while the solving of LAS on mono-processor and MIMD-computers requires times η_1 and η_p respectively.

Determine times required for the performing of one iteration on mono-processor and MIMD-computers:

$$T_1 \approx cqt + \eta_1, \qquad T_p \approx cqt + c_1(t_o + t_c) + \eta_p\,. \tag{2.63}$$

Here $c = N_G n + N_f$, $\quad c_1 = [\log_2 p]\,, \quad c_1(t_o + t_c)$ is time required for the in-pairs communication of processes and data exchange between them. Then the coefficient of efficiency for the above implementation of Newton' method is evaluated by formula:

$$E_p \approx \frac{1 + \eta_1/cnt}{Q}\,, \tag{2.64}$$

where $Q = 1 + c_1/cn(t_o + t_c) + \eta_p p/cnt$. Denote by t, t_o, t_c times for: the performing of one arithmetic operation, exchange by one number and linkage of communication between processes for performing of the required data exchanges, respectively.

Quasi-Newton's methods:

Burdakov's method. Burdakov's method is globally convergent i.e. a choice of the iteration parameter within this method is carried out so as to

ensure convergence of the iterative process starting from any initial approximation to one of system's solutions. Algorithm for Burdakov's method is implemented according to the following scheme.

Each of p processes computes the required number of values of components both of the vector-function and rows of the Jacobi matrix as well as norm of vector-function used in termination criterion of the iterative processes. Distribution of data between processes is carried out automatically in similarly the same manner as described in Newton's method.

Iterative formula of Burdakov's method is the following:

$$x^{(k+1)} = x^{(k)} - \alpha_k \left(H^{(k)}\right)^{-1} f\left(x^{(k)}\right),$$

where α_k is an iteration parameter specially evaluated according to [2] and the Jacobi matrix $H^{(k)}$is evaluated by means of finite differences.

Iterative process in Burdakov's method for the most part coincides with implementation of Newton's method and therefore, coefficient of its efficiency is the same than that in the Newton's method. Error in the computed solution of SNE with respect to exact solution of accurate system of equations is estimated by formula (2.58).

Dennis-Moore method. With pre-specified initial data and on the basis of starting approximation to solution the Dennis-Moore method evaluates the Jacobi matrix and determines its inverse. Further, in course of the iterative process the evaluating of approximations to system's solution this inverse matrix is corrected, so instead of solving linear algebraic system only multiplication of matrix by vector is carried out at each iteration.

The Dennis-Moore method is implemented for $k = 0, 1, \ldots$ by formula:

$$\begin{aligned} x^{(k+1)} &= x^{(k)} - B^{(k)} f\left(x^{(k)}\right), \\ B^{(k+1)} &= B^{(k)} + \frac{\left(w^{(k)} - B^{(k)} y^{(k)}\right) w^{(k)T} B^{(k)}}{w^{(k)T} B^{(k)} y^{(k)}}, \end{aligned} \tag{2.65}$$

where $B^{(k)} = \left(H^{(k)}\right)^{(-1)}$, $w^{(k)} = x^{(k+1)} - x^{(k)}$, $y^{(k)} = f\left(x^{(k+1)}\right) - f\left(x^{(k)}\right)$.

Parallel algorithm for this method is implemented by the following computational scheme.

1. The following initial data are pre-assigned in each of p processes: starting approximation to the solution of system, quantity ϵ to be used in termination criterion for the iterative process for estimating the accuracy of the obtained result and boundaries of region where the solution is to be sought. First, the appropriate number of values both the vector-function and rows of the Jacobi matrix are computed. Computations are automatically distributed between processes in similar the same manner as described for the Newton's method. Then a matrix inverse to Jacobi matrix is evaluated and at that computations in processes are distributed by blocks containing the same number of rows as in original matrix. Processes exchange values of components of the vector-function for the forming of full vector in each of them.

2. For $k = 0, 1, \ldots$ each process:

   (a) computes values of corresponding components of the vector $x^{(k+1)}$ and vector is gathered in all processes,

   (b) computes values of components of the vector-function $f\left(x^{(k+1)}\right)$ and this vector is gathered in all processes,

   (c) verifies the fulfillment of termination criterion for the iterative process:

$$\left\| f\left(x^{(k+1)}\right) \right\| = \sqrt{\sum_{i=1}^{n} f_i^2\ (x^{(k+1)})},$$

   if the condition is fulfilled the iterative process comes to end, otherwise computations are continued from item 'd'; the condition of the attaining of the limiting number of iterations is also verified here, and in so doing the iterative process is terminated if this limiting number of iterations is exceeded,

   (d) computes values of vectors $w^{(k)}$, $y^{(k)}$ as well as values of components of vectors $B^{(k)}y^{(k)}$ and $w^{(k)} - B^{(k)}y^{(k)}$,

   (e) computes vector-rows ${w^{(k)}}^T B^{(k)}$ to be added up and carries out the gathering of vector-row with the summing up,

   (f) computes products ${w^{(k)}}^T B^{(k)} y^{(k)}$ to be added up and carries out the gathering of the product in all processes with summing up,

   (g) computes block of corresponding rows of the matrix $B^{(k+1)}$ by formula (2.65) and comes back to 'a' for the carrying out computations for the next values of k.

After completion of the iterative process an accuracy of the obtained solution with respect to exact solution of accurate system of equations is estimated by formula (2.58).

Let us determine the time required for the performing of one iteration on the mono-processor and MIMD-computers: $T_1 = (9n^2 + 7n + n\,N_f)\,t$, where n is an order of the system; t is time required for the performing of one arithmetic operation and N_f is the number of arithmetic operations required for the evaluation of value of one component of the vector-function. As to multi-processor computer, time required for the performing of one iteration is equal to:

$$T_p = (9n + 1 + N_f)c_1 t + (4 + 2p)nt + (3t_o c_1 + nt_o + 2t_c)c_2\,, \qquad (2.66)$$

where p is the number of processes, $c_1 = \frac{n}{p}$ is an integer part of the relation $c_2 = \log_2 p$; t_o is time of the inter-process exchange by one number; t_c is the time for the communication linkage between two processes.

Therefore the coefficient of efficiency for parallel algorithm for this method is:

$$E_p \approx \left(1 + \frac{(4 + 2p)\,np - 6n + p\,(2\tau_{\rm o} c_1 + n\tau_{\rm o} + 2\tau_{\rm c})\;c_2}{9n^2 + 7n + nN_f}\right)^{-1},$$

where τ_o and τ_c are relations of the exchange time and communication linkage time between two processes to the time required for the performing of one arithmetic operation.

Broyden's method. On the basis of starting approximation to the solution the Broyden's method the Jacobi matrix is evaluated. Further, in course of the iterative process for the evaluating of approximations to system's solution this matrix is corrected by means of the iterative process and used for the solving of linear algebraic system on each iteration.

Broyden's method for $k = 0, 1, \ldots$ is implemented by formulas:

$$\begin{aligned} B^{(k)} w^{(k)} &= -f\left(x^{(k)}\right), \\ x^{(k+1)} &= x^{(k)} + w^{(k)}, \\ y^{(k)} &= f\left(x^{(k+1)}\right) - f\left(x^{(k)}\right), \\ B^{(k+1)} &= B^{(k)} + \frac{\left(y^{(k)} - B^{(k)} w^{(k)}\right) w^{(k)T}}{w^{(k)T} w^{(k)}}, \end{aligned} \qquad (2.67)$$

where $w^{(k)} = x^{(k+1)} - x^{(k)}, \quad y^{(k)} = f\left(x^{(k+1)}\right) - f\left(x^{(k)}\right)$.

Parallel algorithm for this method is implemented by the following computational scheme:

1. Each of p processes on the basis of initial data computes the corresponding number of values of components both of the vector-function and of the Jacobi matrix,

2. Further, for $k = 0, 1, \ldots$ each process:

   (a) by means of any parallel algorithm solves LAS for the evaluating of $w^{(k)}$ and this vector is gathered in all processes,

   (b) computes $x^{(k+1)}$ simultaneously with other processes,

   (c) computes a value of the vector-function $f\left(x^{(k+1)}\right)$ distributed between processes,

   (d) verifies the termination criterion for the iterative process $\left\|f\left(x^{(k+1)}\right)\right\| \le \frac{\varepsilon}{\left\|B^{(k)^{-1}}\right\|}$, where $\left\|f\left(x^{(k+1)}\right)\right\| = \sqrt{\sum_{i=1}^{n} f_i^2\left(x^{(k+1)}\right)}$.
   If this condition is fulfilled the iterative process comes to end, while otherwise it returns to the item 'e'; the condition of the attaining of the limiting number of iterations is also verified here, and in so doing the iterative process is terminated if this limiting number of iterations is exceeded,

   (e) computes a value of the vector $y^{(k)}$ distributed between processes,

   (f) computes a block of corresponding rows of the matrix $B^{(k+1)}$ by formula (2.67) and performs passage to the item 'a' with the next value of k.

After completion of the iterative process a declination of the obtained solution from exact solution of accurate system of equations is estimated by formula (2.58).

Let us determine time required for the performing of one iteration on monoprocessor computer: $T_1 = (N_s + 11n^2 + 5n + n\,N_f)\,t$, where n is an order of system; t is time for the performing of on arithmetic operation; N_f is the number of arithmetic operations required for the computing of value of one component of the vector-function, while N_s is the number of arithmetic

operations required for the solving of LAS. As to multi-processor computer, time required for the performing of one iteration constitutes:

$$T_p = \left(\frac{N_s}{p\,q} + 3n + c_1(4n + 2 + N_f)\right) t + (t_c + c_1 t_o)c_2\,, \tag{2.68}$$

where p is the number of processors; $c_1 = \frac{n}{p}$ is an integer part of the relation $c_2 = \log_2 p$ t_o is time for the inter-process exchange by one number; t_c is the time for the communication linkage between two processes; N_s is the number of arithmetic operations required for the solving of LAS; coefficient of efficiency of algorithm fort he solving of LAS is denoted by q.

Therefore coefficient of efficiency of parallel algorithm implementing this method can be written as follows:

$$E_p \approx \frac{N_s + 4n^2 + 5n + nN_f}{\frac{N_s}{q} + 4n^2 + 3pn + 2n + nN_f + c_2\,(p\tau_c + n\tau_o)}\,,$$

where τ_o and τ_c are relations of the exchange time and communication linkage between two processes respectively, to the time required for the performing of one arithmetic operation.

2.5 Initial-value problems for systems of ordinary differential equations

2.5.1 Statements of problems with approximate initial data

During the computer modeling of real-life phenomena and processes by means of systems of ordinary differential equations (SODE) a need is often generated for the solving of initial-value problems the initial data for which (formulas for the evaluating of right-hand sides of SODE and initial conditions) are given approximately. Approximate nature of these data can be caused by the following reasons:

1. errors in the initial conditions' specification since they as results of various measurements are not accurate,

2. errors in formulas for the evaluation of right-hands sides; these errors are caused by the fact that right-hand sides are some approximations to right-hand sides of realistic systems of differential equations; besides, very often right-hand sides are approximated by simpler functions for the sake of economy of amount of arithmetic operations during their evaluation on each step of integration,

3. in some cases the solution is evaluated from equivalent equations non-resolvable explicitly with respect to the solution being sought,

4. employment of numerical (discrete) integration method and rounding-off numbers during computations,

5. introduction of errors during discretization of various type dynamic problems by means of finite-element method in spatial variables.

The above-mentioned errors present some difficulties in investigating and solving of this type of problems. An answer to the question concerning the effect of errors on the solution is given by theory on the stability of solutions to SODEs.

The initial-value problem for the n-th order SODE in the interval $[t_0, T]$ will be considered in the form:

$$\begin{aligned} \frac{du}{dt} &= f(u), \\ u(t_0) &= u^{(0)}, \end{aligned} \tag{2.69}$$

where $u = (u_1, u_2, \ldots, u_n)^T$ is a vector being sought, while the right-hand side is an n-dimensional function $f(u) = (f_1(u), f_2(u), \ldots, f_n(u))^T$.

The right-hand sides of the system are considered to be continuous. System of the form (2.69) is considered to be a system written in general form because with the explicit dependence of the right-hand on t it's possible to introduce extra component of the vector u (for example u_{n+1}, if order of system (2.1) is equal to n), extra equation $du_{n+1}/dt = 1$ and extra initial condition $u_{n+1}(t_0) = t_0$ and thus reduce the available problem to that written in form (2.69). Problem (2.69) is a problem with approximate data.

Along with this problem let us consider problem with accurate initial in the interval $t \in [t_0, T]$:

$$\begin{aligned} \frac{d\,v}{d\,t} &= \varphi\,(v)\,, \\ v\,(t_0) &= v^{(0)}\,, \end{aligned} \tag{2.70}$$

where v is an n-dimensional vector and $\varphi\,(v)$ – n-dimensional vector-function.

Generally, for problems (2.69)-(2.70) the additional inequalities:

$$\|\,\varphi\,(w) - f\,(w)\,\| \leq\, \Delta, \quad \left\|\,u^{(0)} - v^{(0)}\right\| \,\leq\, \delta\,, \tag{2.71}$$

should be given for arbitrary values $w(t)$ determining accuracy of the problem's initial-data pre-assignation.

2.5.2 Method for the SODE solution

One of the important characteristics of differential problems presenting a severe difficulty during the numerical solving of problems of the form (2.69) is stiffness.

By 'stiff' problem we mean such a problem components of whose solution vector are stable (i.e. there is no eigenvalue of the Jacobi matrix having large positive real part in some interval where solution is being sought) and, at least, some components of the solution vector are strongly stable (i.e. at least one eigenvalue of the Jacobi matrix possesses large in modulus negative real part in some interval where the solution is sought). Fundamental tools for solving of initial-value problems for SODE are numerical methods.

The following methods are employed for the solving of common ('non-stiff') SODE:

(I) 4-th order Runge-Kutta method as most wide-spread and extensively used method for practical numerical calculations,

(II) 5(6)-th order Runge-Kutta method [4] which can be used for some control calculations and comparison of the obtained solution with solutions obtained by 4-th order Runge-Kutta method,

(III) Euler-Cauchy method which is stable for purely implicit eigenvalues of the Jacobi matrix,

(IV) Adams methods of order up to 12-th [5]: almost always ensuring the required accuracy of the solution and somewhat minimizing the solution time at the expense of automatic choice both of length of the integration step size and method's order.

The following methods will be used for the solving of 'stiff' SODE:

(V) Gear's methods of orders up to 5-th possessing the same positive characteristics as Adams' methods.

Computational formulas for the above-mentioned methods will be given during the implementation of parallel algorithms in form of computational schemes and estimating the efficiency of algorithms' implementation on parallel computers.

During the constructing of numerical methods for the solving of initial-value problems the original differential problem is replaced (approximated) by discrete problem which contains additional discretization parameter, an algorithm and convenient for the computer implementation computational scheme for the solving of discrete problem are constructed and questions related to the numerical stability of solution method and convergence of discrete problem's solution to the solution of original differential problem are investigated.

It is shown that error in the numerical solution of problem (2.69) at any point of integration interval is estimated by formula:

$$\left\| z^{(k+1)} \right\| \;\leq\; \rho_k \left\| z^{(k)} \right\| + \psi^{(k)}, \tag{2.72}$$

where $z^{(k)} = y^{(k)} - u(t_k)$, $y^{(k)}$ is the solution of difference problem obtained by numerical method at the point t_k; $u(t_k)$ is the solution of problem (2.69) at the same point; $\psi^{(k)}$ is a norm of the error in the approximation of differential equation by difference equation, while ρ_k is an estimate of norm of step-to-step passage operator.

The condition $0 < \rho_k \leq 1$ ensures stability of the numerical method, while condition $\psi^{(k)} \leq \varepsilon\,(1 - \rho_k)$ provides given accuracy of the solution, i.e. ful-

fillment of inequality $\left\|y^{(k)} - u(t_{k+1})\right\| \leq \varepsilon$ at each point of the integration interval.

Quantities ρ_k and $\psi^{(k)}$ depend both on the length of integration step h_k and the numerical method in use. For example, for the 1st order explicit method:

$$\rho_k = \left\|I + h_k H(y^{(k)})\right\|, \qquad \psi^{(k)} = \frac{h_k^2}{2} \max_{1\leq \mathrm{j} \leq \mathrm{n}} \left| \frac{f_j\left(y^{(k+1)}\right) - f_j\left(y^{(k)}\right)}{h_k} \right|,$$

where I is an identity matrix; $H(y^{(k)})$ is the Jacobi matrix for system (2.69); this matrix is negatively determined for problems possessing asymptotically stable solutions.

The length of the integration step for each numerical method is evaluated both from stability conditions and conditions for the attaining of solution's accuracy.

If integration step is chosen from these conditions then $\left\|y^{(k+1)} - u\left(t_{k+1}\right)\right\| \leq \varepsilon$ condition holds at each point t_{k+1} of the integration interval. Then the declination of the numerical solution of problem (2.69) with approximate data from exact solution of problem (2.70) with accurate data is estimated by formula:

$$\left\|y^{(k+1)} - v\left(t_{k+1}\right)\right\| \leq \varepsilon + \delta + \left(t_{k+1} - t_0\right)\Delta. \tag{2.73}$$

It should be taken into account that actually two last addends have a form $\prod_{i=0}^{k+1} \rho_i \delta + \sum_{i=0}^{k} h_i \prod_{j=i+1}^{k+1} \rho_j \Delta$. And since for stable integration methods ρ_i is always less than 1, the performing of large number of integration steps results in that fact that product ρ_i tends to zero. In this formula this product is replaced by 1, i.e. in (2.73) the estimate from above for the error is given. Practical error is considerably lesser.

The following techniques for the determining both of Lipschitz constant at any k-th integration step and approximation to evaluates of Jacobi matrix is employed:

(VI) a vector:

$$w^{(k)} = \begin{cases} \varepsilon\left(1 + f\left(y^{(k)}\right)\right) & \text{when } \varepsilon\left(1 + f\left(y^{(k)}\right)\right) \neq 0, \\ \varepsilon f\left(y^{(k)}\right), & \text{otherwise} \end{cases}$$

is constructed,

(VII) a value $H\left(y^{(k)}\right) w^{(k)} = f\left(y^{(k)} + w^{(k)}\right) - f\left(y^{(k)}\right)$ is evaluated,

(VIII) from the identity $B^{(k)(0)} w^{(k)} \equiv H\left(y^{(k)}\right) w^{(k)}$ the diagonal matrix (vector) $B^{(k)(0)}$ is evaluated,

(IX) a vector is constructed by the rule $x_i^{(k)} = (-1)^{i-1}\, w_i^{(k)} \quad (i = 1, 2, \ldots, n)$,

(X) the diagonal matrix $B^{(k)(3)}$ is evaluated by formula:
$\frac{f_i\left(y^{(k)}+x^{(k)}\right)-f_i\left(y^{(k)}\right)}{x_i^{(k)}} = B_i^{(1)} \quad (i = 1, 2, \ldots, n)$,

(XI) the diagonal matrix $B^{(k)}$ is evaluated by formula:

$$B_i^{(k)} = \begin{cases} B_i^{(k)(0)} & \text{when} \quad B_i^{(k)(0)} \times B_i^{(k)(1)} < 0, \\ \min\left(B_i^{(k)(0)},\, B_i^{(k)(1)}\right) & \text{otherwise} \end{cases}$$

$(i = 1, 2, \ldots, n)$; components of the diagonal matrix approximate eigenvalues of the Jacobi matrix,

(XII) the Lipschitz constant $L_k = \max\limits_{1 \le i \le n} \left|B_i^{(k)}\right|$ is evaluated; if it is bounded the solution exists and is unique,

(XIII) a length of the integration step $h_k = 2/L_k$ is evaluated; whereupon one step of integration is performed; all subsequent lengths of integration steps are evaluated by means of formulas similar to (2.7).

In doing so the information obtained during the evaluation of Lipschitz constant is used. For example, for the 1-st explicit method the quantity ρ_k (with taking into account the obtained values) is evaluated by relevant formula.

2.5.3 Parallel algorithms for the solving of initial-value problems for SODE

For most numerical methods the number of arithmetic operations required on one step of integration during the solving of problem (2.69) is mainly determined by the number of operations required for the evaluation of right-hand sides of system of equations. Therefore, for most methods the evaluation of

right-hand sides is to be parallelized first of all. Based on this parallelization the evaluation of Jacobi matrix is automatically parallelized. However, if the number of arithmetic operations required for evaluation of vector of right-hand sides is insignificant, it is advisable to carry out the computation of solution of initial-value problems for SODE either on one or two processes.

The number of components of vector-function to be evaluated on one process is determined as follows: the ratio $\frac{n}{p} = g$ is evaluated, where a is an integer part of the number a; n is an order of system of equations; p is the number of processes performed on MIMD-computer; further the value $s = p\,(g + 1) - n$ is valuated; then in the last s processes the right-hand sides will be computed by blocks, each block containing $q = g$ equations, while in the first $p - s$ processes the right-hand sides will be computed by blocks of $q = g+1$ equation each. This enables to perform automatic distribution of computation of components of the SODE's vector-function on p processes.

The following function may serve as a template of program on C for the evaluation of components of vector-function of SODE's right-hand sides:

```
void diffun(int n, int l, int m, double t, double *y, double *f)
  {.....
  for ( i = l; i < m; i++) {
    .....
    }
  }.
```

Each process computes l-th trough $(m - 1)$-st components of the vector-function. In so doing $l = kq$, $m = (k+1)\ q-1$, where k is the logical number of process, $k = 0, 1, ..., p-1$; t is an independent variable; y is dependent variable (solution); f is vector of the values of those components of the vector-function which were computed by the k-th process at the point t.

For all numerical methods intended for the integration of SODE the following initial data are to be pre-assigned in order indicated below:

1. order of system of ordinary differential equations,
2. the number of output points,
3. starting point of the integration interval,

4. final point of the integration interval,
5. required accuracy of computations,
6. error in the initial data pre-specification,
7. error in formulas for the evaluation of right-hand sides,
8. initial values of the solution vector,
9. array containing coordinates of solution's output points.

4-th order Runge-Kutta method. On the i-th step of integration (i.e. for the evaluation of solution at point $t_i = \sum_{j=0}^{i-1} h_j$) the classical 4-th order Runge-Kutta method is implemented by formulas:

$$y^{(i+1)} = y^{(i)} + (k_1 + 2k_2 + 2k_3 + k_4)/6\,, \tag{2.74}$$

where:

$$\begin{aligned} k_1 &= h_i f(t_i, y^{(i)}), \\ k_2 &= h_i f(t_i + h_i/2, y^{(i)} + 0.5k_1), \\ k_3 &= h_i f(t_i + h_i/2, y^{(i)} + 0.5k_2), \\ k_4 &= h_i f(t_i + h_i, y^{(i)} + k_3) \quad (i = 0, 1, 2, ...)\,. \end{aligned} \tag{2.75}$$

On each step of integration this method requires four-fold computation of the vector-function $f(t,\ y)$. The 4-th order Runge-Kutta method is the most widespread method for the numerical solving of initial-value problems for SODE.

Parallel algorithm of the 4-th order Runge-Kutta method on p processes is implemented according to the following computational scheme:

1. initial data are sent to each of p processes,
2. the following computations (in the indicated order) are carried out in the interval of integration starting from the initial value of independent variable to the coordinate of output point for each value of $t_i = \sum_{j=0}^{i-1} h_j (i{=}0,\ 1,\ 2,...)$:

(a) diagonal matrix $\left\{ B_j^{(0)} \right\}$ $(j = 1, 2, \ldots, n)$ is evaluated by algorithm described in the previous paragraph; system's right-hand sides are always computed by blocks in each process and immediately gathered in all processes in order to form full vector in each process,

(b) a predicted length of the integration step is evaluated by formula: $h_i = 2/ \max\limits_{1 \le i \le n} \left| B_j^{(i)} \right|$,

(c) $\bar{\bar{y}}^{(i+1)}$ is evaluated by double integration with step length equal to $h_i/2$ and $\bar{y}^{(i+1)}$ is evaluated by integration with step length equal to h_i according to formulas (2.74) and (2.75); during computations by formulas (2.75) right-hand sides of the system are always evaluated by blocks and immediately gathered in all processes in order to form a full vector in each processes,

(d) an error caused by the approximation of system of differential equations by numerical integration formulas (2.74), (2.75) is evaluated:

$$\psi^{(i+1)} = \max_{1 \le j \le n} \left| \bar{y}_j^{(i+1)} - \bar{\bar{y}}_j^{(i+1)} \right| ,$$

(e) two values $\rho_i^{(1)}$ and $\rho_i^{(2)}$ are evaluated by formula:

$$\rho_i = \left| 1 + h_i B_j^{(i)} + \frac{h_i^2}{2} B_j^{(i)\,2} \frac{h_i^3}{6} B_j^{(i)\,3} + \frac{h_i^4}{24} B_j^{(i)\,4} \right| ,$$

for minimum and maximum values of $B_j^{(i)}$; $\max\left(\rho_i^{(1)},\ \rho_i^{(2)}\right)$ is chosen as value of ρ_i ,

(f) an improved length of the integration step is evaluated by formula:

$$h_{new} = h_i \sqrt[5]{\frac{\varepsilon\ |1 - \rho_i|}{\psi^{(i+1)}}} ,$$

proceeding from the satisfying of criterion for the attaining of required accuracy,

(g) the following value:

$$h_i = \begin{cases} h_{new}, & \text{if } h_{new} \times \left| B^{(i)} \right| \le 2.78529, \\ 2.78529/ \left| B^{(i)} \right| & \text{otherwise,} \end{cases} ,$$

is chosen as the length of the integration step, where $\mid B(i) \mid = \max\limits_{1\le i\le n} \left| B_j^{(i)} \right|$,

(h) the condition $t_i + h_i > t_p$; is verified if it is fulfilled then $h_i = t_p - t_i$ is chosen as length of the integration step, where t_p is a coordinate of the solution's output points otherwise length of the integration step is not corrected,

(i) for the obtaining of solution $y^{(i+1)}$ at the point $t_{i+1} = t_i + h_i$ the integration is carried out by formulas (2.74), (2.75) with already computed length of the integration,

(j) the condition for the termination of the process of solving of the initial-value problem for SODE $t_{i+1} > T$ is verified, if it is fulfilled the process of solving the problem is over, otherwise the process of solving is continued beginning from the item 'a'.

On the evaluation calculation of solution at output the point the following information is saved: a solution vector, Lipschitz constant and estimate for the solution's error evaluated by formula (2.73).

Let us estimate the efficiency coefficient for this implementation of parallel algorithm for the 4-th order Runge-Kutta method.

After the computation of vector-function it is necessary to exchange information so that the computed vector be available in every process, i.e. Multi-gathering of vector's components should be carried out. Let us denote by T_2 time required for multi-gathering:

$$T_2 = \log_2 p \; t_c + \left(\frac{s\,(s+1) + p^2 - 1}{2} \right) q \; t_o, \qquad s = \left[\frac{p+1}{2} \right].$$

Time required for the evaluation of vector-function on the MIMD-computer is equal to $T_3 = qt_f$ where t_f is average time required for the computation of one component of the vector-function on mono- and multi-processor computers. A choice of the next integration step requires time equal to $T_6 = 2s(t_c + t_o)$. Then efficiency coefficient of parallel algorithm for the 4-th order Runge-Kutta method is determined as follows:

$$E_p = \frac{45\,t + 12\,t_f + 9\,t/n}{45\,t + 12\,t_f + 9\,t/n + 12\,T_2 + T_6}.$$

Here the following notations are introduced: t_o – time for exchange by one number between processes; t_c – communication linkage time between two processes; t – time required for the performing of one arithmetic operation.

Algorithm described above for the solving of initial-value problem for SODE enables to obtain a solution of the problem within a priori given accuracy in all points of the integration interval, i.e. it ensures the obtaining of solution of the problem with global accuracy.

The 5(6)-th order Runge-Kutta method. This method belongs to method of Runge-Kutta type. It enables to carry out a control over the error arising during the evaluation of solution of the initial-value problem for SODE on every step and thus to use this information for the evaluation of such length of the integration step enables to attain the given local error in the solution.

For this method right-hand sides will be considered in the form $f(t,\, y)$ since computational formulas involve coefficients of parameter t different from coefficients of the parameter y.

Formulas for the evaluation of solution at the point t_{i+1} proceeding from known solution at the point t_i are the following:

1. for the 5-th order method:

$$y^{(i+1)} = y^{(i)} \,+\, h_i \sum_{j=0}^{5} \gamma_j \, k_j \,, \tag{2.76}$$

2. for the 6-th order method:

$$\bar{y}^{(i+1)} \,=\, y^{(i)} \,+\, h_i \sum_{j=0}^{7} \bar{\gamma}_j \, k_j \,, \tag{2.77}$$

where $k_0 = f(t_i, y^{(i)})$, $k_j = f\Big(t_i + \alpha_j h_i,\, y^{(i)} + h_i \sum_{\nu=0}^{j-1} \beta_{j\nu} k_\nu\Big)$ $(j = 1,$ $2, \ldots, 7)$.

Let us give values of coefficients:

$\alpha_1 = 1/6$, $\alpha_2 = 4/15$, $\alpha_3 = 2/3$, $\alpha_4 = 4/5$, $\alpha_5 = 1$, $\alpha_6 = 0$, $\alpha_7 = 1$,

$\gamma_0 = 31/384$, $\gamma_1 = 0$, $\gamma_2 = 1125/2816$, $\gamma_3 = 9/32$, $\gamma_4 = 125/768$,

$\gamma_5 = 5/66$, $\bar{\gamma}_0 = 7/1408$, $\bar{\gamma}_1 = 0$, $\bar{\gamma}_2 = 1125/2816$, $\bar{\gamma}_3 = 9/32$,

$\bar{\gamma}_4 = 125/768$, $\bar{\gamma}_5 = 0$, $\bar{\gamma}_6 = 5/66$, $\bar{\gamma}_7 = 5/66$,

and non-zero values of the following coefficients:

$\beta_{10} = 1/6$, $\beta_{20} = 4/75$, $\beta_{30} = 5/6$, $\beta_{40} = -8/5$, $\beta_{50} = 361/320$,

$\beta_{60} = 11/640$, $\beta_{70} = 93/640$, $\beta_{21} = 16/15$, $\beta_{32} = 5/2$, $\beta_{42} = -4$,

$\beta_{52} = 407/128$, $\beta_{62} = 11/256$, $\beta_{72} = 803/256$, $\beta_{43} = 16/25$,

$\beta_{53} = 11/80$, $\beta_{63} = 11/160$, $\beta_{73} = -11/160$, $\beta_{54} = 55/128$,

$\beta_{64} = 11/256$, $\beta_{74} = 99/256$, $\beta_{76} = 1$.

Principal term of the local error on each step of integration is evaluated by formula:

$$M_i = 5h_i(k_0 + k_5 - k_6 - k_7)/66\,. \tag{2.78}$$

Parallel algorithm for the 5(6)-th order Runge-Kutta method on p processes is implemented by the following computational scheme.

1. The initial data are sent to each of p processes.
2. Diagonal matrix $\left\{B_j^{(0)}\right\}$ $(j = 1, 2,\ldots, n)$ is evaluated by algorithm described in the previous paragraph; in each process the right-hand sides of system are always evaluated by blocks and immediately gathered in all processes in order to form a full vector in each process.
3. Length of the integration step is predicted by formula $h_0 = 2/\max\limits_{1\le i\le n}\left|B_j^{(0)}\right|$ and $g = 1/375h_0$.
4. Further the following computations are carried out in the indicated order over the integration interval starting from the initial value of independent variable to the co-ordinate of output point for each value of $t_i = \sum_{j=0}^{i-1} h_j$ $(i{=}0, 1, 2,...)$:

(a) $\bar{\bar{y}}^{(i+1)}$ computed according to formulas (2.10) and $\bar{y}^{(i+1)}$ is evaluated according to formulas (2.11) with length of the integration step h_i; during computations according to these formulas the right-hand sides of system are always computed by blocks and immediately gathered in all processes in order to form a full vector in each process,

(b) $\psi^{(i+1)} = \max\limits_{1 \le i \le n} \left| \bar{y}_j^{(i+1)} - \bar{\bar{y}}_j^{(i+1)} \right|$ is evaluated,

(c) if $\psi^{(i+1)} < 0.5\varepsilon$, then $\bar{y}^{(i+1)}$ is chosen as a solution and co-ordinate t_i is increased by the length of the integration step,

(d) if $\psi^{(i+1)} < 1 \times 10^{-12}$ then $h_{i+1} = g$ is chosen as length of the next integration step, otherwise $s = h_i \sqrt[6]{\frac{\varepsilon}{\psi^{(i+1)}}}$ is evaluated and if $s > 2\ h_i$ then $h_{i+1} = 2h_i$ is chosen else $h_{i+1} = s$; if at that $h_{i+1} > g$ then length of the integration step is chosen as follows: $h_{i+1} = g$,

(e) if $\psi^{(i+1)} \ge 0.5\varepsilon$ then $h_{i+1} = \sqrt[6]{\frac{\varepsilon}{2\psi^{(i+1)}}}$,

(f) the condition for running over the integration interval until the output point is attained $h_{i+1} \ge t_p$ is verified; if this condition is fulfilled the results are saved and integration is continued on the next interval until the next output point is attained,

(g) if $t_{i+1} + h_{i+1} > T$ a length of the integration step is corrected $h_{i+1} = T - t_{i+1}$ following which the solving of problem is continued from the item 'a'.

Upon evaluation of solution the following information is printed at the output point: solution, Lipschitz constant and solution's error estimate evaluated by formula (2.73).

Let us estimate coefficient of efficiency for this implementation of parallel algorithm for the 5(6)-th order Runge-Kutta method. After computation of vector-function processes should exchange information so that the computed vector be available in each process, i.e. multi-gathering of vector's components should be carried out

Coefficient of efficiency of this parallel algorithm on one step of integration is determined by formula:

$$E_p = \left(1 + \frac{(p-1)\ n + 0.5p\ \ (p+1)\ \tau_{\mathrm{c}} + p\, q\, \tau_{\mathrm{o}}\, 2^{0.5(p+1)} + 13\, n\, \tau_f}{127\, n + 13\, n\, \tau_f} \right)^{-1},$$

where τ_c and τ_o are relations of the communication linkage time between two processes and time required for the exchange by one number between them, respectively, to time required for the performing of one arithmetic operation, while τ_f is relation of average time required for the computation of one component of the vector-function to time required for the performing of one arithmetic operation.

Euler-Cauchy method. On the i-th step integration this method is implemented by formulas:

$$\begin{aligned} y^{(i+1)(0)} &= y^{(i)} + h_i\, f\left(y^{(i)}\right), \\ y^{(i+1)(k)} &= y^{(i)} + 0.5\, h_i\, \left(f\left(y^{(i)}\right) + f\left(y^{(i+1)(k-1)}\right)\right), \end{aligned} \tag{2.79}$$

$k = 1, 2, 3, \ldots$ is the number of iteration, $t_{i+1} = t_i + h_i$ (i=0, 1, 2, ...).

The main advantage of this method is the fact that it is stable for purely imaginary eigenvalues of Jacobi matrix and therefore can be used for calculation of oscillation processes ensuring the second order of accuracy. Parallel algorithm for this method on p processes is implemented by the following computational scheme:

1. initial data are sent to each of p processes,
2. further on each step of integration:
   (a) diagonal matrix $\left\{ B_j^{(0)} \right\}$ $(j = 1, 2, \ldots, n)$ is computed by algorithm described in the previous paragraph; right-hand sides of system are always evaluated by blocks and immediately gathered in order to form a full vector in each process,
   (b) a length of integration step is evaluated by formula:
   $$h_0 = 1/\left(2 \max_{1 \le i \le n} \left| B_j^{(0)} \right|\right),$$
   proceeding from conditions of method's stability and convergence of the iterative process on the integration step,
   (c) the condition for running over the integration interval until the output point is attained $t_{i+1} \ge t_p$ is verified; if this condition is fulfilled the results are saved and integration is continued on the next interval until the next point is attained,

(d) if $t_{i+1} + h_{i+1} > T$ then the integration step's length is corrected $t_{i+1} = T - t_{i+1}$,

(e) 10 iterations are performed according to formula (2.79) since after 10 iterations initial error reduces by a factor of 10^6; in addition the condition $\left\| y^{(i+1)(k)} - y^{(i+1)(k-1)} \right\| \leq 1 \times 10^{-7}$ is verified at each iteration and if it is fulfilled the iterative process is finished at this iteration,

(f) $y^{(i+1)(k)}$ is considered to be a solution at point t_{i+1} and computations are continued beginning from item 'a'.

Upon computation of solution at the output point the following information is to be printed: the solution, Lipschitz constant as well as solution's error estimate evaluated by formula (2.73).

Coefficient of efficiency of this method on one step of integration is:

$$E_p \approx \frac{5\,t + t_f + S_2\,(t + t_f)}{5\,t + t_f + S_2\,(t + t_f) + S_1\,S_2\,T_2},$$

where t is time required for the performing of one arithmetic operation; t_f is an average time required for the computation of one component of the vector-function; $S_1 = p/n$, S_2 is the number of iterations; time required for gathering of vector in all processes is:

$$T_2 = \log_2 p\; t_{\mathrm{c}} + \left(\frac{s\,(s+1) + p^2 - 1}{2} \right) q\; t_{\mathrm{o}}, \qquad s = \left[\frac{p+1}{2} \right],$$

t_c and t_o are times required for the communication linkage and exchange by one number between two processes, respectively.

Adam's methods of order up to 12-th. These methods belong to linear multi-step methods and have the form:

$$y^{(i)} = y^{(i-1)} + h \sum_{s=0}^{m} b_{-s} f\left(t_i - s h_{i-1}, y^{(i-s)} \right),\ m \geq 0,$$

where m is method's order. Coefficients for these methods are given in [5].

Implementation of parallel algorithm for the se methods in the form of general computational scheme is based on the same ideas those employed as a basis for the implementation of computational schemes of the above-mentioned methods. Therefore only some general remarks will be given on the implementation of computational scheme without presenting the computational scheme itself.

1. Because of the fact that Adam's methods are multi-step methods the computation of groups of components of SODE's right-hand sides vector, similarly as in the above-described methods, is distributed between processes, but multi-gathering of the computed vector is not performed.

2. Groups of components of the solution vector are computed in parallel in processes, and multi-gathering of the solution vector in al processes is performed only prior to the computation of the right-hand sides of SODE's.

3. Norms of errors are also computed in processes by parts with subsequent summing of these parts for the obtaining of the full norm.

4. The solving of problem always begins from the 1-st order method.

5. Starting length of the integration step is always evaluated in the same manner as in 4-th order Runge-Kutta method, for example.

6. In the process of further searching for the solution in case of unsatisfactory length of the integration step, i.e. such length that doesn't ensure the attaining of given accuracy of the obtaining of approximate solution even if 3−5 iterations for the refining of solution have been performed, length of the integration step is 10 times reduced and attempts to obtain the required solution are repeated until the proper length of the integration step is found.

7. If method of a certain order in one step of integration without iterative refinement yields a solution whose error is less than given accuracy, then order of this method is increased by 1 with simultaneous increase in the integration step's length. The step's length is increased proportionally to the 'reduced accuracy to norm of solution's error' ratio for the chosen step-size.

8. After the performing of every integration step the condition are verified for the attaining both of output point and final point of the integration.

9. For every output of results it is necessary to compute both the Lipschitz constant and solution error determined by formula (2.73) with taking into account approximate nature of data.

Coefficient of efficiency of this method (if its order doesn't change) is as follows:

$$E_p \approx \frac{p\ ((2m+2+t_f)\ n+t)}{T_6}.$$

Here the notations introduced above are used, in particular, $T_6 = 2s\ (t_c + t_o)$, $\quad s = \left[\frac{p+1}{2}\right]$; m is the method's order.

Gear's methods of order up to 5-th. These methods are linear multi-step methods with prediction and correction of solution and employed for the integration of 'stiff' SODE. Gear's methods are based on multi-step Curtis-Hirshfelder. The m-th order method is determined as follows:

$$y^{(k)} = \sum_{j=1}^{m} \alpha_j\, y^{(k-j)} + \eta h_k f\left(t_k,\, y^{(k)}\right) \qquad (k = 0, 1, \ldots), \tag{2.80}$$

where m is an order of the method; h_k is the integration step's length; α_j and η are coefficients given in [5].

Implementation of parallel algorithm for these methods in form of general computational scheme is the same as that of the Adams's method. All remarks made as to implementation of Adam's methods are valid for Gear's methods. However, some extra comments should be made.

Formulas for the obtaining a solution to 'stiff' problems are implicit, therefore it is necessary to solve system of non-linear algebraic equations for the obtaining of solution. A procedure of solving non-linear system requires considerable expenses compared to methods intended for the solving of 'non-stiff' problems and this procedure is suitable for 'stiff' problems only due to the fact that during their solving the integration step's length turns out t be considerably greater than that for the Runge-Kutta and Adam's methods.

Let us apply it to the 1-st order implicit method:

$$\begin{aligned} y^{(k+1)} &= y^{(k)} + h_k f(y^{(k+1)}) \qquad (k = 0, 1, \ldots), \\ y^{(0)} &= u^{(0)}. \end{aligned} \tag{2.81}$$

This starting formula is transformed into the calculation formula:

$$y^{(k+1)} = y^{(k)} + \frac{h_k f\left(y^{(k)}\right)}{1 - h_k B^{(k)}}, \qquad y^{(0)} = u^{(0)}, \tag{2.82}$$

where diagonal matrix $B^{(k)}$ is evaluated, for example, in the manner described in paragraph 2.5.2. Such symbolic notation is used only because a denominator of each solution's component turns into a number. Besides, in the general case the denominator has a form $1 - \alpha h_k B^{(k)}$, and in particular, for the second order method the coefficient $\alpha = 2/30$. Instead of $\varepsilon \left(1 + f\left(y^{(k)}\right)\right)$ the following expression:

$$0,1 \left(h_k f\left(y^{(k)} + h_k f\left(y^{(k)}\right)\right) - h_k f\left(y^{(k)}\right)\right),$$

can be used as increments.

The next step of integration is evaluated similarly as for Adam's method. For such implementation of these methods the coefficient of efficiency of algorithm for Gear's methods on parallel computer is the same as that of Adam's method.

2.6 Intelligent software for investigating and solving of problems with approximate initial data

By intelligent software for the solving of a certain class of problems we'll mean a set of program tools enabling to:

1. formulate a problem in computer in terms of the subject area language (further referred to as a computer problem),

2. investigate mathematical characteristics of computer problem with approximate initial data,

3. choose both method and efficient algorithms for the solving of computer problem according to the revealed characteristics as well as according to mathematical and engineering characteristics of computer,

   (a) determine the number of processes required for the efficient solving of problem and choose an appropriate topology of inter-processor communications for the MIMD-computer,

   (b) form executable code of parallel program implementing chosen solution algorithms,

   (c) solve the computer problem,

   (d) investigate reliability of the obtained results and evaluate estimates both for inherited and computational errors in the obtained solution of the computer problem.

Composition of the intelligent software Inparsoft as well as its interaction with applications software is depicted in Fig. 2.3.

At the structural level Inparsoft consists of the intelligent software Inpartool intended for the automatic investigating and solving of problems (copyright certificate No. 23462 as of 17.01.2008 issued by State department of intellectual property) and library Inparlib consisting of intelligent programs for investigating and solving problems of the computational mathematics (copyright certificate No. 17213 as of 11.07.2006 issued by State department of intellectual property).

At the functional level Inparsoft is, on one hand, an instrumental tool for the investigating and solving of basic classes of problems of the computational mathematics listed above and, on the other hand, parallel programs included in Inparlib library serve as material (as reuse programs) for the creation of applications software for the solving of scientific and engineering problems. From the user's point of view, Inpartool is a product of the end user, while Inparlib is an instrumental tool of a user-developer of applications software.

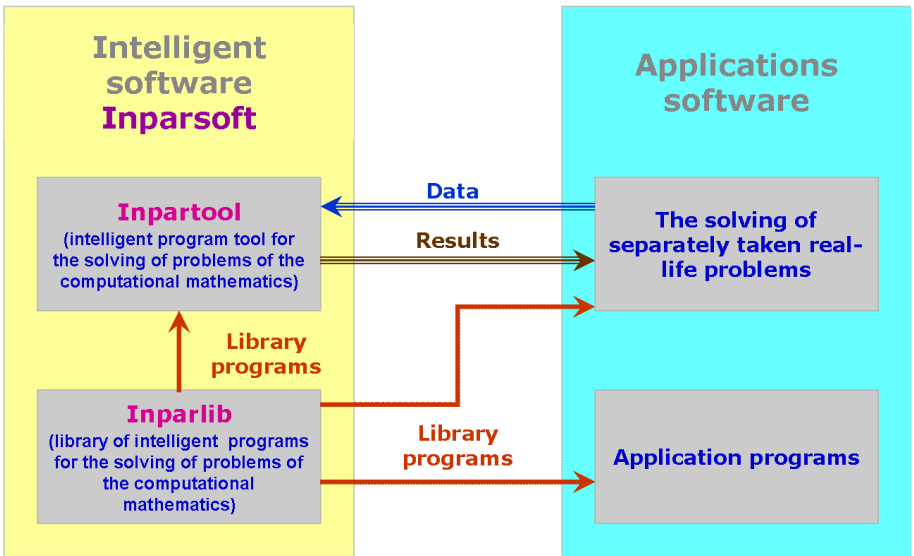


Figure 2.3: Scheme of Inparsoft's composition and its interaction with applications software

2.6.1 Intelligent software Inpartool

Conception of Inpartool. In the computer solving of most scientific and engineering problems the solving of one or several problems belonging to basic classes of the computational mathematics is of fundamental importance, namely:

1. linear algebraic systems,
2. algebraic eigenvalue problem,
3. non-linear equations and systems,
4. initial-value problems for systems of ordinary differential equations.

In so doing the main characteristic feature is an approximate nature of the initial data of problems being solved.

It should be emphasized that due to error in the initial data the mathematical problem should be considered as a problem with a priori unknown characteristics. Therefore for the efficient solving of mathematical problem with approximate initial data the following should be carried out:

1. investigation of the correctness of problem, i.e. existence and uniqueness of its solution,

2. determination of region of the solution's stability,

3. estimation of an inherited error in solution of the problem, i.e. assessment of influence of approximate initial data on the solution.

A problem, either input into or formed in the computer (a computer problem) which in the final analysis is solved, always has an approximate nature with respect to original problem due to errors occurring during inputting of the numerical data into computer and performing of arithmetic operations over them in the computer. These errors are caused, in particular, by the following:

1. continuum of real numbers is approximated by finite set of rational numbers in computer (even during the input of numerical data the rounding-off errors arise),

2. a phenomenon of 'machine zero' gives rose to a number of difficulties during the implementation of computational algorithms (any computer has the least positive number which can be represented in it); all numbers less in modulus than this number are replaced by zero,

3. as a rule due to finite length of machine words in computer the associative, distributive and commutative laws of mathematics are violated during performing of floating-point arithmetic operations.

Thus, characteristics of computer problem may differ from characteristics of its corresponding original problem. Therefore, computer methods and technologies (described in previous sections) are required for the investigating and solving of computer problems together with reliability estimates of the obtained results.

During the solving of problems on MIMD-computers, problems arise, as well, related to the efficient parallelization of solution algorithms taking into account both mathematical and engineering characteristics of computer, including structure of inter-processor communication, and characteristics of the system software, including parallel programming systems.

Difficulties mentioned above (for the end user) occurring during the solving of problems on MIMD-computers can be overcome by means of the intelligent software Inpartool intended for the solving of basic classes of problems of the computational mathematics.

The intelligent software Inpartool has been created on the basis of the following conceptual theses:

1. investigating of mathematical characteristics and solving of problems with approximate initial data,
2. automatic choice of efficient algorithms according to the revealed characteristics of problems being solved as well as according to mathematical and engineering characteristics of MIMD-computer,
3. investigation of reliability of computer solution,
4. implementation of the hidden parallelism principle.

The hidden parallelism principle consists in the following:

1. automatic control over distribution and re-distribution of information between processes; excluding of Hyden's effect,
2. automatic determination of number of processes required for the efficient solving of problem,
3. automatic construction of the efficient topology of inter-process communications.

Estimates both for coefficient of acceleration S_p and coefficient of efficiency E_p of parallel algorithms are serve as mathematical foundation for the hidden parallelism principle enabling to automatically distribute information and determine the required number of processes (these estimates are given in previous sections).

Thus, for the end user the same technology for the solving of problems as on mono-processor (personal) computer is provided on MIMD-computer.

Inpartool consists of four separate components each intended for the investigating and solving of problems belonging to one of four above-mentioned fundamental classes of problems of the computational mathematics.

As to class *'Linear algebraic systems'*, in addition to investigating and solving of LAS together with reliability estimates for the obtained results, Inpartool also allows solving of the following problems: inversion/pseudo-inversion of matrix together with estimates for results' reliability, evaluation of matrix condition number, singular-value decomposition of matrix, evaluation of the matrix rank. These problems can be solved for the following types of matrices: dense non-singular, dense symmetric positive definite, banded symmetric positive definite, banded symmetric positive semi-definite, rectangular of the arbitrary rank.

As to class *'algebraic eigenvalue problem'*, the following problems can be solved and investigated: full standard AEVP $Ax = \lambda x$ with tri-diagonal symmetric matrix; full standard AEVP with dense symmetric matrix; partial standard AEVP (finding of several minimal eigenvalues and their corresponding eigenvectors) with banded symmetric positive definite matrix; partial generalized AEVP $Ax = \lambda Bx$ with banded symmetric positive definite matrices. For the all above-mentioned problems Inpartool investigates reliability of the obtained results.

As to class *'Systems of non-linear equations'* Inpartool solves the following problems: finding of all roots of one equation in the given interval; evaluation of solution of system of equations in the given region. In the neighborhood of solution being sought (a root of one equation or system's solution) characteristics of problem are investigated and on this basis the inherited and computational errors of the obtained solution of the problem are estimated. When employing automatic solution procedure a globally convergent method is used while during the interactive solution procedure a list of iterative methods is proposed to the user for choice.

As to class *'Systems of ordinary differential equations'*, the initial-value problems for the 1-st SODE are investigated and solved. Inpartool enables to integrate both common and 'stiff' SODE with accuracy of different order, including any a priori given accuracy. Inpartool always investigates whether a system of ordinary differential equations is 'stiff' or not, evaluates the Lipschitz constant, estimates both computational and inherited errors in the obtained solution of the problem. At the automatic solution procedure Inpartool chooses numerical method for the solving of problem according to the revealed characteristics of problem. At the interactive solution procedure user gets information about SODE's characteristics and list of numerical methods is proposed to user enabling to evaluate o solution of problem within the required accuracy.

2.6.2 Examples of solving of problems by means of Inpartool

Some computational potentialities of Inpartool will be illustrated below on the example of some test problems.

Problem 1

Investigate and solve LAS $Ax = b$, where:

$A = (a_{ij}), \qquad a_{ii} = n - i, \qquad a_{ij} = n + 1 - max(i, j)$

$(i,\, j\, = 1,\ldots,\, n, \qquad n\, = 3w\, + 1, \qquad w\, = 1,\, 2,\, \ldots);$

$b = \{b_i\}_1^n\,, \qquad b_1 = n - 1, \qquad b_2 = n - 2, \qquad b_i = n + 1 - i, \qquad (i > 2)\ .$

Exact solution of the system is:

$$x = (\ 0 \quad 1 \quad 0 \quad ... \quad 0\ )^T\,.$$

Thus, matrix A has the form:

$$A = \begin{pmatrix} n-1 & n-1 & n-2 & ... & 2 & 1 \\ n-1 & n-2 & n-2 & ... & 2 & 1 \\ n-2 & n-2 & n-3 & ... & 2 & 1 \\ ... & ... & ... & ... & ... & ... \\ 2 & 2 & 2 & ... & 1 & 1 \\ 1 & 1 & 1 & ... & 1 & 0 \end{pmatrix},$$

while its second column is a vector of the right-hand side b. The input data for LAS are given accuracy, i.e. maximal relative errors both in the matrix elements and right-hand side are equal to zero.

Protocol of automatic investigating and solving of problem 1 by Inpartool

```
P R O B L E M:
      solving of the linear algebraic system
      with a symmetric positive defined matrix
D a t a :
     - matrix dimension                    = 1000
     - number of the right-hand side
       of the systems                      = 1
     - maximum relative error
       of the matrix elements              = 0.00000e+00
     - maximum relative error
       of elements of the right-hand sides = 0.00000e+00
P r o c e s s  o f  i n v e s t i g a t i n g
a n d  s o l u t i o n  m e t h o d:
     - Cholesky decomposition
R E S U L T S:
     !!! THE MATRIX IS NOT POSITIVE DEFINED !!!
     Number of processors: 4
M e t h o d:
     - Gauss elimination with partial pivoting
R E S U L T S:
     !!! THE MATRIX IS MACHINE-SINGULAR !!!
     Number of processors: 4
M e t h o d:
     - singular value decomposition of a general matrix
R E S U L T S:
     SOLUTION WAS CALCULATED
     first 4 components of solution (vector 1) are:
     -3.7747582837255322e-010    1.0000000000000031e+000
      3.8857805861880479e-010    3.6489927986770073e-010
The vecror(s) of solution are successfully stored
in the file result.out
     Error estimations:   4.99145e-08
     P r o p e r t i e s:
     - estimation of conditional number: 7.49316e+07
     - matrix rank: 999
     Number of processors: 12
```

From this protocol the fact is obvious that since the system's matrix is sym-

metric then an algorithm for the Cholesky method, as the most economic method for such matrices (see section 2.3.1), was chosen as test algorithm for the investigating of this problem by means of Inpartool. However, in the process of investigation the matrix turned out to be not positive definite and algorithm for the Gauss method was chosen for further investigation of problem by means of Inpartool. During this investigation of LAS the matrix turned out to be singular. For such kind of LAS a generalized solution of the computer problem can be evaluated by using of matrix singular-value decomposition (see section 2.3.1). The problem was solved together with reliability estimates for the obtained solution. During the process of problem's investigating and solving the number of processes optimal for each algorithm was chosen automatically, an efficient topology was constructed and data were distributed between processes according to the solution algorithm.

Problem 2

Evaluate 8 minimal eigenvalues and their corresponding eigenvectors of the generalized eigenvalue problem with banded symmetric positive definite matrices A and B.

Matrices A and B were obtained during the finite-element discretization of eigenvalue problem for the Laplace operator in rectangular whose one side is fixed. In this case the matrices are block tri-diagonal:

$$A = \begin{pmatrix} A_1 & A_2 & & & 0 \\ A_2 & 2A_1 & A_2 & & \\ & \ddots & \ddots & \ddots & \\ & & A_2 & 2A_1 & A_2 \\ 0 & & & A_2 & 2A_1 \end{pmatrix}, \quad B = \begin{pmatrix} 2B_1 & B_1 & & & 0 \\ B_1 & 4B_1 & B_1 & & \\ & \ddots & \ddots & \ddots & \\ & & B_1 & 4B_1 & B_1 \\ 0 & & & B_1 & 4B_1 \end{pmatrix},$$

$$A_1 = \begin{pmatrix} a_2 & f & & & 0 \\ f & 2a_2 & f & & \\ & \ddots & \ddots & \ddots & \\ & & f & 2a_2 & f \\ 0 & & & f & a_2 \end{pmatrix}, \quad A_2 = \begin{pmatrix} g & a_1 & & & 0 \\ a_1 & 2g & a_1 & & \\ & \ddots & \ddots & \ddots & \\ & & a_1 & 2g & a_1 \\ 0 & & & a_1 & g \end{pmatrix},$$

$$B_1 = \begin{pmatrix} 2c_0 & c_0 & & & 0 \\ c_0 & 4c_0 & c_0 & & \\ & \ddots & \ddots & \ddots & \\ & & c_0 & 4c_0 & c_0 \\ 0 & & & c_0 & 2c_0 \end{pmatrix}, \quad \begin{array}{l} a_1 = -(c_1 + c_2), \\ a_2 = 2(c_1 + c_2), \\ f = c_1 - 2c_2, \\ g = c_2 - 2c_1, \end{array}$$

$$c_0 = \frac{1}{36 N_x N_y}, \quad c_1 = \frac{N_y}{6 N_x}, \quad c_2 = \frac{N_x}{6 N_y},$$

where N_x, N_y are numbers' of rectangular region's partitioning in horizonatal and vertical directions, respectively. The order of square blocks A_1, A_2 and B_1 is equal to N_{x+1}, the number of such blocks in matrices A and B is N_y. Thus, the order of matrices A and B is $n = (N_x+1)N_y$, while semi-bandwidth of these matrices is $m = N_x + 2$ (the band width is $2m + 1 = 2N_x + 5$).

Exact eigenvalues of Problem 2 are evaluated by formula:

$$\tilde{\lambda}_{kl} = \frac{1}{c_0} \left(\frac{c_2(1 - s_k)}{2 + s_k} + \frac{c_1(1 - t_l)}{2 + t_l} \right),$$

where $s_k = \cos\frac{\pi k}{N_x}$, $t_l = \cos\frac{(l-0.5)\pi}{N_y}$, $0 \le k \le N_x$, $1 \le l \le N_y$.

Protocol describing the process of solving of Problem 2 by method of iterations on the sub-space (see section 2.3.2) is given below with $N_x = 319$, $N_y = 50$, i.e. $n = 16\ 000$, while the semi-bandwidth is $m = 321$ (a full bandwidth is equal to 643). The problem was solved on 16 processes.

Protocol of automatic investigating and solving of problem 2 by Inpartool

```
P R O B L E M :
   Solving of Partial Generalized
   Eigenvalue Problem
   for Band Symmetric Matrices
INPUT PARAMETERS:
        order of matrices              = 16000
        bandwise of matrix A           = 643
        bandwise of matrix B           = 643
        maximal relative errors:
                  of matrix A elements  = 0.000e+00
                  of matrix B elements  = 0.000e+00
        number of minimal eigenvalues
                           to calculate  = 8
Exact eigenvalues
    2.467604042554091e+00     1.233728821340764e+01
    2.222305254921369e+01     3.209273672006724e+01
    4.194729797545172e+01     6.170274648211132e+01
    6.181196631645549e+01     7.168165048730904e+01
    9.130050516997107e+01     1.012916602493531e+02
    1.110559536766307e+02     1.213906838857423e+02
    2.011944685514055e+02     3.015383945680863e+02
    1.312603680565959e+02     2.110641527222591e+02
P r o c e s s  o f  r e s e a r c h  a n d  s o l u t i o n
of the problem
M e t h o d :    Subspace Iterations
    matrix blocksize           = 10
    number of processors       = 16
R e s u l t s :    SOLUTION WAS CALCULATED
                   by 16 iterations  (mit=32)
All calculated eigenvalues are minimal
Eigenvalues (calculated)     Estimates of Errors
    2.467604042574461e+00         4.493e-15
    1.233728821342248e+01         2.096e-12
    2.222305254923304e+01         2.674e-15
    3.209273672008100e+01         3.727e-12
    4.194729797546218e+01         5.368e-10
    6.170274648216704e+01         4.187e-07
    6.181196631647476e+01         6.440e-09
    7.168165048952910e+01         2.728e-06
```

Problem 3 Solve a system of non-linear equations:

$$\sum_{j=0}^{n-1} x_j - 0.5\,(3n+1) + 2x_i^2 - 2\left(1 + 2\frac{i}{n} + \left(\frac{i}{n}\right)^2\right) = 0\,,$$

in the region $D = \{-1000 \le x_i \le 1000\}$ ($i = 0, 1, 2, \ldots, n-1$).

Exact solution of this problem is $x_i = 1 + \frac{i+1}{n}$, the starting approximation was chosen to be $1 + 0.5\left(\frac{i+1}{n}\right)$ ($i = 0, 1, 2, \ldots, n-1$).

The problem 3 was solved by Burdakov method (see section 2.4) with $n = 100$, limiting number of iterations $it = 100$, given accuracy $\varepsilon = 10^{-10}$, error in function's pre-assignation $\delta = 10^{-10}$. The problem was solved on 4 processes.

Protocol of automatic investigating and solving of problem 3 by Inpartool

```
 read n  = 100            - order of the system
 read it = 100            - the limiting number of iterations
 read eps=  1.000000e-10  - the given accuracy
 read del=  1.000000e-10  - accuracy of the obtained solution
 read a[i] - left  border of the area


  -1.0000000000e+03   -1.0000000000e+03   -1.0000000000e+03
  -1.0000000000e+03   -1.0000000000e+03   -1.0000000000e+03
  -1.0000000000e+03   -1.0000000000e+03   -1.0000000000e+03
  -1.0000000000e+03
read b[i] - right border of the area
  1.0000000000e+03   1.0000000000e+03   1.0000000000e+03
  1.0000000000e+03   1.0000000000e+03   1.0000000000e+03
  1.0000000000e+03   1.0000000000e+03   1.0000000000e+03
  1.0000000000e+03
read x[i] - initial approximation
  1.0000000000e+00   1.0500000000e+00   1.1000000000e+00
  1.1500000000e+00   1.2000000000e+00   1.2500000000e+00
  1.3000000000e+00   1.3500000000e+00   1.4000000000e+00
  1.4500000000e+00
```

```
T h e   p r o t o c o l   o f   t h e   d e c i s i o n
Task was solved by Burdakov method on 4  processors
The results of the computation are written in a file
result_bur.out
The number of the executed iterations it = 4
The given accuracy  eps  = 5.4760572666e-15
Accuracy of the obtained solution del = 1.0000005476e-08
Residuals on a sequence of iterations new =
  2.7293374767e+00   2.4199117540e-02   2.1327656368e-06
  4.8584631731e-12
Solution
  1.0100000000e+00   1.1100000000e+00   1.2100000000e+00
  1.3100000000e+00   1.4100000000e+00   1.5100000000e+00
  1.6100000000e+00   1.7100000000e+00   1.8100000000e+00
  1.9100000000e+00
```

Problem 4 In the interval [0; 0,004] solve the following system of ordinary differential equations:

$$\frac{du_i}{dt} = -\sum_{j=0}^{n-1} u_j - u_i + n\,(1+t) + 2 + t\,,$$

under the initial conditions $u_i(0) = 1, \quad (i = 0, 1, 2, \ldots, n-1)$.

$u_i = 1 + t, \quad (i = 0, 1, 2, \ldots, n-1)\,.$

A protocol describing the solving of Problem 4 by Adams' method is given below (see section 2.5) for error $n = 4\,000$ and initial data error $\delta_1 = \delta_2 = 10^{-10}$. The integration step's length has been chosen from condition that solution's error should not exceed $\varepsilon = 10^{-6}$. The problem was solved on 16 processes.

Protocol of automatic investigating and solving of problem 4 by Inpartool

```
The problem is non-stiff
The problem is being solved by 12-th order
ADAMS methods order of system n= 4000
```

```
  the number of processors 16
point t= 4.000000e-02
At this point the local Lipschitz constant L=  4.041011e+03
Solution = 1.040000e+00 1.040000e+00 1.040000e+00 1.040000e+00
At this point the solution is obtained with accuracy
delta= 1.000000e-06
```

2.6.3 Library of intelligent programs Inparlib

Purpose and creation principles of Inparlib library. By the intelligent program we'll mean a program which during the problem's solution process verifies agreement between solution algorithm and characteristics of the computer problem; in automatic mode forms efficient processes configuration according to the number of processes in use; distributes data between processes; solves the problem and estimates reliability of the obtained computer solution or yields indicator of reason of refusal in the solving the problem.

Intelligent programs included in Inparlib library implement the solving of the same classes of the computational mathematics as Inpartool. From the end use's point of view, library programs are re-use components for the solving of application problems for which problem or problems of the computational mathematics are components of the solution processes.

Programs (program modules) of Inparlib are based on the following principles:

1. parallelism,
2. scalability,
3. modularity,
4. efficient utilization of cache-memory.

Parallelism ensures the simultaneous performing of operations that is essential for programs which are performed on several processes.

Scalability provides possibility to change the number of processes in use. Modularity reflects an extent of decomposition of complicated objects on

simpler components. Modular structure of programs ensures the using of the same program modules in different programs.

Investigation of various algorithms for the solving of problems of the computational mathematics has demonstrated that their efficiency considerably depends on the efficiency of employing of processes' cache-memory. During the performing of any algorithm the latest instructions and data are saved in the cache-memory, so cycles and operations over arrays of data are performed considerably more rapidly. If the required data are not available in the cache-memory the process performing a task is to address to the main operating memory that considerably slows down the performing of algorithm. At that the program could be made more efficient if loop operations implementing matrix-vector operations be performed with unfolding of inner loop with taking into account the length of computer's instruction pipeline. Therefore algorithms for programs included in Inparlib library are implemented with taking into account dimensions both of the cache-memory and instruction pipeline of Inparcom's processors.

Programs included in Inparlib library implement:

1. investigation of characteristics of computer problems with appropriate initial data,
2. verification of agreement between revealed characteristics of computer problem and solution algorithm,
3. construction of Inparcom's processor topology,
4. solving of problem,
5. investigating of reliability of the obtained results, including evaluation of estimates for the inherited and computational errors.

As to linear algebraic systems, Inparlib's program modules enable to investigate and solve problems with different structure matrices, estimate solution's reliability, invert matrices, evaluate singular values and matrix rank as well as to estimate the matrix condition numbers.

Program modules included in Inparlib enable to investigate and solve both full standard and partial standard or generalized algebraic eigenvalue problem with various structure symmetric matrices (dense, banded or profile). By

means of Inparlib's programs one may investigate reliability of the obtained results and evaluate estimates for errors in solutions.

Programs included in Inparlib enable to investigate and solve of non-linear algebraic and transcendental equations; at that the local condition number of function $f(x)$ and local condition number of vector-function $F(x)$ are determined; termination criterions for iterative processes ensuring the obtaining of computer solution within the given accuracy are implemented; solution's error is estimated with taking into account approximate nature of the initial data.

For the investigating and solving of the initial-value problems for systems of ordinary differential equations Inpartool contains programs enabling to integrate both common and 'stiff' systems of equations within various order of accuracy, including a priori given accuracy. Inparlib's program modules enable to investigate 'stiffness' of systems of ordinary differential equations, evaluate both Lipschitz constant and error in the obtained solution with taking into account approximate initial data.

Bibliography

[1] C.G. Broyden: A class of methods for solving nonlinear simultaneous equations Math. Comput, 1965, 19, No. 92, pp. 577–593.

[2] O.P. Burdakov: Some globally convergent modifications of the Newton's method for solving systems of non-linear equations, Dokl. AN SSSR, 254, No. 3, (3980), pp. 521-523, (in Russian).

[3] J.E. Dennis, J.J. More: A characterization of super-linear convergence and its application to quazi-Newton methods, Math. Comput., 28, No. 126, (3974), pp. 549-560.

[4] E. Fehlberg: Klassische Runge-Kutta Formelniverter und niedregerer Ordnung mit Schriftweitenkontrolle und ihre Anwendung auf Warmelitungsprobleme, Computing, 6(3970), pp. 61-71.

[5] C.W. Gear: Numerical initial value problem in ordinary differential equations, Prentice-Hall, New Jersey, 1971, – 278p.

[6] A.N. Khimich, I.N. Molchanov, A.V. Popov, M.F. Yakovlev, V.I. Mova, et al.: Parallel algorithms for the solving of problems in the computational mathematics, Kyiv: Naukova dumka, 2008. – 247 p. (in Russian)

[7] A.N. Khimich, I.N. Molchanov, V.I. Mova, et al.: Numerical software for the intelligent MIMD-computer Inparcom, Kyiv: Naukova dumka, 2007, – 216 p. (in Russian)

[8] V.S. Mikhalevich, N.A. Bic, V.N. Brusnikin, A.N. Khimich, et al.: Numerical methods for multiprocessor computation complex ES, Ed. by I.N. Molchanov: Moscow, Aviation Academy Academy named after prof. N.Ye. Zhukovsky, 1986, – 401 p.

[9] I.N. Molchanov: Introduction to algorithms of parallel computations, Naukova Dumka, Kyiv, 1990. – p. 127. (in Russian)

[10] J. Ortega: Introduction to parallel and vector solution of linear systems, -Moskov., 1991, – 364 p. (in Russian)

[11] V.V. Voyevodin, Vl.V. Voyevodin: Parallel computations. BHV, St. Peterbourg, 2002, 608 p. (in Russian)

[12] http://www.netlib.org/scalapack

[13] http://www.mcs.anl.gov/mpi/mpich

Chapter 3

Multiprocessor computing structures

V. Tulchinsky, O. Perevozchikova, W. Surtel, A. Smolarz

3.1 Introduction

Since the beginning of modern computing, most computer programs have been written using a serial programming model. Apart from a brief era of parallel programming in the late '80s on systems such as Thinking Machines and MasPAR, serial programming has been the predominant programming model for more than 50 years. But with the availability of new, powerful data parallel processors, such as the General-Purpose Graphics Processing Units (GPGPU, also referred to as GPU), as well as the arrival of highly multi-core Central Processing Units (CPU), the serial programming model faces significant sailing challenges. The emerging world of highly parallel systems requires a programming model that scales to the new generation of parallel architectures that are already coming onto the market.

The starting point for the new parallel programming era is probably 1994 when Thomas Sterling and Donald Becker at NASA developed a high-perfor-

mance parallel computing cluster Beowulf. Beowulf was compound of inexpensive personal computer hardware with free and open source software. Since publication in 1997 'Beowulf HOWTO' of Jacek Radajewski and Douglas Eadline the Beowulf-like clusters rapidly became deployed worldwide, chiefly in support of scientific computing. As result High Performance Computing (HPC) from privileged use of rare supercomputers for solution of extraordinary mostly research problems has come to be applied to various scientific, engineering and business tasks.

Traditional 'single-core' CPU hit a fundamental performance limitation in early 2000 due to power and implementation difficulties. Until this wall was reached, applications could move seamlessly from one generation of processor to the next and take advantage of the increased clock speed without needing to change anything in their code. Now, however, the major CPU manufacturers have shifted to a multi-core strategy. High-end commodity CPUs today are quad-core x86 processors. Gulftown which first release under the brand name Core i7 980X was in the first quarter of 2010 contains 6 cores with hyper-threading and so serves up to 12 threads in parallel. Two such processors can convert usual PC in a powerful enough hardware engine suitable for efficient parallel service of multiple asynchronous requests to host a web-server, a database server an application server as well as for a small scale HPC calculations.

More direct way to explicitly parallel microprocessor architecture was selected by an alliance of Sony Computer Entertainment, Toshiba, and IBM, known as STI. They combined a general-purpose PowerPC based core of modest performance with vector coprocessor elements which greatly accelerate multimedia and uniform data processing applications. The architectural design and first implementation were carried out at the STI Design Center in Austin, Texas in 2001-2004. The developed Cell Broadband Engine Architecture, commonly abbreviated CBEA in full, Cell BE or Cell in part was first commercialized in Sony's PlayStation 3 game console (2005). In May 2008, an Opteron- and PowerXCell 8i-based supercomputer, the IBM Roadrunner system, became the world's first system to achieve one petaFLOPS, and was the fastest computer in the world until fall 2009. The world's three most energy efficient supercomputers, as represented by the Green500 list, are similarly based on the PowerXCell 8i.

Stream processor architectures became new players in the HPC game little after the Cell. Stream processors extend the capabilities of vector processors by adding a layer to the register hierarchy, and adding a layer of instruction

sequencing that enables them to operate in record (rather than operation) order. The idea was first implemented in 2003 at Stanford University Graphics Lab. Stream architecture computer Merrimac used it to give an order of magnitude more performance per unit cost than cluster-based scientific computers built that time from the same technology. In late 2006, NVIDIA and ATI (acquired by AMD in 2006) began marketing GPU boards as streaming coprocessor boards. Both companies released software to expose the processor to application programmers. Since then the hardware has grown more powerful and the available software has become more high-level and developer-friendly.

GPGPU exceeded alternative parallel architectures by proposing hundreds threads and up to thousand time speedup for dedicated computations. The stream software development appeared to be more simple and intuitive than for vector computers, digital signal processors (DSP) and transputers. As result GPGPU market primary based on NVIDIA CUDA architecture have raised so rapidly that IBM currently promotes Cell BE as a compromise of CPU and GPU concepts.

Processing dependent computing has flourished under the wealth of software applications that have been written to take advantage of these new modern processors, and as a society, we have eagerly integrated these advances into our daily lives. Businesses perhaps now more than ever rely on computers to process transactions, data, and a plethora of other information, and home users have begun using their computers for much more than typing and games. Universities have also capitalized on this growth in computational power and are just beginning to realize the potential in harnessing massive amounts of computational power for aiding research.

The more computation resources are available the more computation power is desirable. Grid computing is the concept behind harnessing the resources of multiple computers/cluster over a network, and using that power to solve very big compute-intensive problems. Grid computing is currently a hot topic amongst corporate and academic circles, and is most prominent in fields where massive amounts of processing are done on a daily basis. Some of these fields include nuclear physics, physical chemistry, biomedical research, scientific simulation.

In the most general form a modern computing system is a heterogeneous grid of clusters combined of CPU/GPU nodes. Its efficient utilization requires cooperation of multiple technologies.

The section 3.2 introduces parallel computing. The next three sections describe modern parallel hardware, parallel programming tools and grid middleware. More general questions of performance estimates and program engineering for parallel software are discussed in the sections 3.7-3.8. Massive parallel data processing is examined in the last section of the chapter.

3.2 Subject and Jargon of Parallel Computing

As parallel computers become larger and faster, it becomes feasible to solve problems that previously took too long to run. Parallel computing is used in a wide range of fields, from bioinformatics (to do protein folding) to economics (to do simulation in mathematical finance). Here we present and briefly describe main concepts of parallel computing [29].

Task. The first step in designing a parallel program is to break the problem up into tasks. A task is a sequence of instructions that operate together as a group. This group corresponds to some logical part of an algorithm or program. For example, consider the multiplication of two N-order matrices. Depending on how we construct the algorithm, the tasks could be (1) the multiplication of subblocks of the matrices, (2) inner products between rows and columns of the matrices, or (3) individual iterations of the loops involved in the matrix multiplication. These are all legitimate ways to define tasks for matrix multiplication; that is, the task definition follows from the way the algorithm designer thinks about the problem.

Unit of execution (UE). To be executed, a task needs to be mapped to a UE such as a process or thread. A process is a collection of resources that enables the execution of program instructions. These resources can include virtual memory, I/O descriptors, a runtime stack, signal handlers, user and group IDs, and access control tokens. A more high-level view is that a process is a 'heavyweight' unit of execution with its own address space. A thread is the fundamental UE in modern operating systems. A thread is associated with a process and shares the process's environment. This makes threads lightweight (that is, a context switch between threads takes only a small amount of time). A more high-level view is that a thread is a 'lightweight' UE that shares an address space with other threads. We will use unit of execution or UE as a generic term for one of a collection of

possibly concurrently executing entities, usually either processes or threads. This is convenient as less depended from a particular architecture.

Processing unit (PU). We use the term processing unit as a generic term for a hardware element that executes a stream of instructions. The element of hardware considered to be a PU depends on the context. For example, some programming environments view each workstation in a cluster of SMP workstations as executing a single instruction stream; in this situation, the PU would be the workstation. A different programming environment running on the same hardware, however, might view each processor of each workstation as executing an individual instruction stream; in this case, the PU is the individual processor, and each workstation contains several PUs. In GPGPU case PU is usually associated with a single core able to execute a work thread.

Load balancing. To execute a parallel program, the tasks must be mapped to UEs, and the UEs to PUs. How the mappings are done can have a significant impact on the overall performance of a parallel algorithm. It is crucial to avoid the situation in which a subset of the PUs is doing most of the work while others are idle. Load balancing is the process of allocating work to PUs, either statically or dynamically, so that the work is distributed as evenly as possible.

Synchronization. In a parallel program, due to the non-determinism of task scheduling and other factors, events in the computation might not always occur in the same order. For example, in one run a task might read variable x before another task reads variable y; in the next run with the same input, the events might occur in the opposite order. In many cases, the order in which two events occur does not matter. In other situations, the order does matter, and to ensure that the program is correct, the programmer must introduce synchronization to enforce the necessary ordering constraints.

Synchronous versus asynchronous. We use these two terms to qualitatively refer to how tightly coupled in time two events are. If two events must happen at the same time, they are synchronous; otherwise they are asynchronous. For example, message passing (that is, communication between UEs by sending and receiving messages) is synchronous if a message sent must be received before the sender can continue. Message passing is asynchronous if the sender can continue its computation regardless of what happens at the receiver, or if the receiver can continue computations while waiting for a receive to complete.

Fine-grained, coarse-grained, and embarrassing parallelism. Applications are often classified according to how often their subtasks need to synchronize or communicate with each other. An application exhibits fine-grained parallelism if its subtasks must communicate many times per second; it exhibits coarse-grained parallelism if they do not communicate many times per second, and it is embarrassingly parallel if they rarely or never have to communicate. Embarrassingly parallel applications are considered the easiest to parallelize.

Race conditions. A race condition is a kind of error peculiar to parallel programs. It occurs when the outcome of a program changes as the relative scheduling of UEs varies. Because the operating system and not the programmer controls the scheduling of the UEs, race conditions result in programs that potentially give different answers even when run on the same system with the same data. Race conditions are particularly difficult errors to debug because by their nature they cannot be reliably reproduced. Testing helps, but is not as effective as with sequential programs: a program may run correctly the first thousand times and then fail catastrophically on the thousand-and-first execution – and then run again correctly when the programmer attempts to reproduce the error as the first step in debugging.

Race conditions result from errors in synchronization. If multiple UEs read and write shared variables, the programmer must protect access to these shared variables so the reads and writes occur in a valid order regardless of how the tasks are interleaved. When many variables are shared or when they are accessed through multiple levels of indirection, verifying by inspection that no race conditions exist can be very difficult. Tools are available that help detect and fix race conditions, such as ThreadChecker from Intel Corporation, and the problem remains an area of active and important research.

Deadlocks. Deadlocks are another type of error peculiar to parallel programs. A deadlock occurs when there is a cycle of tasks in which each task is blocked waiting for another to proceed. Because all are waiting for another task to do something, they will all be blocked forever. As a simple example, consider two tasks in a message-passing environment. Task A attempts to receive a message from task B, after which A will reply by sending a message of its own to task B. Meanwhile, task B attempts to receive a message from task A, after which B will send a message to A. Because each task is waiting for the other to send it a message first, both tasks will be blocked forever. Fortunately, deadlocks are not difficult to discover, as the tasks will stop at

the point of the deadlock.

3.2.1 Computer architecture classification

Flynn's taxonomy (Fig. 3.1) is the most common way to characterize different computer architectures. It was proposed by Michael J. Flynn in 1966 [40].

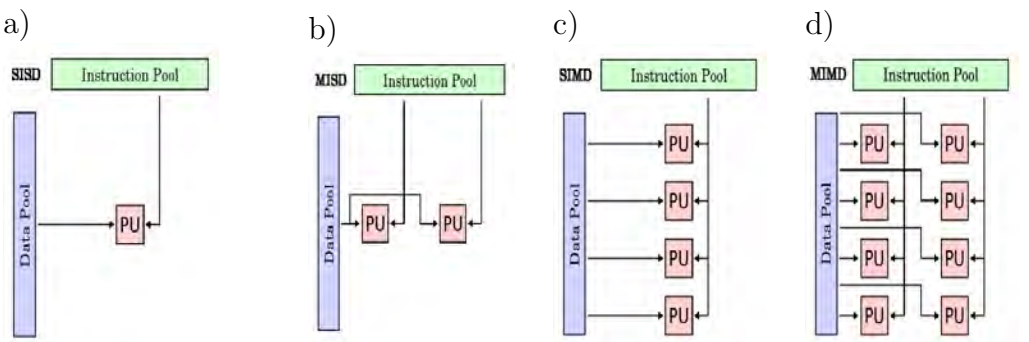


Figure 3.1: Flynn's taxonomy of computer architectures

Single Instruction, Single Data stream (SISD)

A sequential computer which exploits no parallelism in either the instruction or data streams. Examples of SISD architecture are PDA like PocketPC. The old time machines like Intel 8086 based PCs or historical mainframes were really serial. But more recent ones were not pure SISD. Even Intel 80386 pipelined instructions, and the Pentium of 1993 was the superscalar processor.

Multiple Instruction, Single Data stream (MISD)

Multiple instructions operate on a single data stream. Uncommon architecture which is generally used for fault tolerance. Heterogeneous systems operate on the same data stream and must agree on the result. Examples include the Space Shuttle flight control computer.

Single Instruction, Multiple Data streams (SIMD)

A computer which exploits multiple data streams against a single instruction stream to perform operations which may be naturally parallelized. For example, an array (vector) processor or GPU (at the warp/work-group level).

Both AMD Brook+ and NVIDIA CUDA (since Fermi) provide concurrent execution of multiple kernels and so are beyond of SIMD limitations.

Multiple Instruction, Multiple Data streams (MIMD)

Multiple autonomous processors simultaneously executing different instructions on different data. Distributed systems are generally recognized to be MIMD architectures.

Some further divide the MIMD category into the following categories:

- **Single Program, Multiple Data (SPMD)**

Multiple autonomous processors simultaneously executing the same program (but at independent points, without synchronism that SIMD imposes) on different data. Also referred to as 'Single *Process*, Multiple Data'. SPMD is the most common style of parallel programming. The term was originally coined by G.F. Pfister.

- **Multiple Program Multiple Data (MPMD)**

Multiple autonomous processors simultaneously operating at least 2 independent programs. Typically such systems pick one node to be the 'host' ('the explicit host/node programming model') or 'master' (the 'Master/Worker' strategy), which runs one program that farms out data to all the other nodes which all run a second program. Those other nodes then return their results directly to the master.

The MIMD category of Flynn's taxonomy is too broad to be useful on its own. With a few marginal exceptions all modern computers tend to the MIMD architecture. So MIMD is typically decomposed according to memory organization (Fig. 3.2).

Shared Memory refers to a large block of random access memory that can be accessed by all PUs uniformly. A shared memory system is relatively easy to program since all processors share a single view of data and the communication between processors can be as fast as memory accesses to a same location. The issue with shared memory systems is that many CPUs need fast access to memory and will likely cache memory, which has two complications:

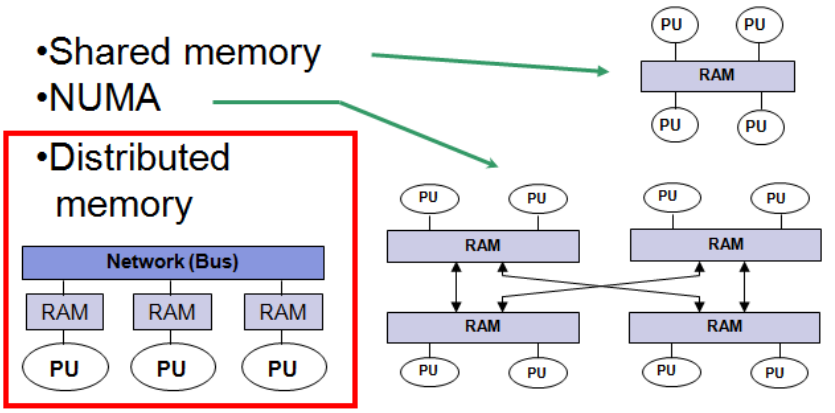


Figure 3.2: Classification of MIMD Parallel Computers by memory (RAM) access

1. CPU-to-memory connection becomes a bottleneck. Shared memory computers cannot scale very well. Most of them have ten or fewer processors,

2. cache coherence: Whenever one cache is updated with information that may be used by other processors, the change needs to be reflected to the other processors, otherwise the different processors will be working with incoherent data. Such coherence protocols can, when they work well, provide extremely high-performance access to shared information between multiple processors. On the other hand they can sometimes become overloaded and become a bottleneck to performance.

Shared memory typically corresponds to Symmetric Multiprocessing (SMP) computer architecture where two or more identical processors or cores are connected to a single shared main memory and are controlled by a single Operation System (OS) instance. SMP systems allow any processor to work on any task no matter where the data for that task are located in memory, provided that each task in the system is not in execution on two or more processors at the same time; with proper OS support, SMP systems can easily move tasks between processors to balance the workload efficiently.

The Burroughs B5500 first implemented SMP in 1961. Then SMP architecture was popular among mainframes. Nowadays most entry and mid-level servers use between two and eight processors. High-end systems, with sixteen or more processors, are also available. Cluster nodes are commonly multiprocessor/multicore SMP. Though, shared memory architecture exists only

at some level of abstraction as cash memory of CPUs is not shared.

Non-Uniform Memory Access (NUMA) logically follow in scaling from SMP architectures. Since '70s CPUs operate considerably faster than the main memory to which they are attached. Limiting the number of memory accesses provided the key to extracting high performance from a modern computer. NUMA attempts to address this problem by providing separate memory for each processor, avoiding the performance hit when several processors attempt to address the same memory. For problems involving spread data (common for servers and similar applications), NUMA can improve the performance over a single shared memory by a factor of roughly the number of PUs (or separate memory banks). Of course, more than one processor may require the same data. To handle these cases, NUMA systems include additional hardware or software to move data between banks. This operation has the effect of slowing down the EUs attached to those banks, so the overall speed increase due to NUMA will depend heavily on the nature of the tasks run on the system at any given time. NUMA commercial development came in work by Burroughs, Convex Computer, Silicon Graphics, Sequent Computer Systems, Data General and Digital during the '90s.

Nearly all CPU architectures use a small amount of very fast non-shared memory known as cache to exploit locality of reference in memory accesses. With NUMA, maintaining cache coherence across shared memory has a significant overhead. Although simpler to design and build, non-cache-coherent NUMA systems become prohibitively complex to program in the standard von Neumann architecture programming model. As a result, most NUMA computers sold to the market use special-purpose hardware to maintain cache coherence, and thus class as 'cache-coherent NUMA', or ccNUMA. Typically, this takes place by using inter-processor communication between cache controllers to keep a consistent memory image when more than one cache stores the same memory location. For this reason, ccNUMA performs poorly when multiple processors attempt to access the same memory area in rapid succession. Operating-system support for NUMA attempts to reduce the frequency of this kind of access by allocating processors and memory in NUMA-friendly ways and by avoiding scheduling and locking algorithms that make NUMA-unfriendly accesses necessary. Alternatively, cache coherency protocols such as the MESIF protocol attempt to reduce the communication required to maintain cache coherency.

In a **Distributed Memory** system there is typically some form of interconnection that allows programs executed on each separated PU to interact with

each other. The interconnect can be organized with point to point links or via a switching network. The network topology is a key factor in determining how the multi-processor machine scales. The links between nodes can be implemented using some standard network protocol (for example Ethernet), using bespoke network links (used in for example the Transputer), or using dual ported memories.

The key issue in programming distributed memory systems is how to distribute the data over the memories. Depending on the problem solved, the data can be distributed statically, or it can be moved through the nodes. Data can be moved on demand, or data can be pushed to the new nodes in advance.

Data can be kept statically in nodes if most computations happen locally, and only changes on edges have to be reported to other nodes. An example of this is simulation where data is modeled using a grid, and each node simulates a small part of the larger grid. On every iteration, nodes inform all neighboring nodes of the new edge data.

Cluster and computational grid systems are typical examples of distributed memory systems at some level of abstraction.

Hybrid Systems combine features of multiple memory architectures. With some insight any real hardware system can be assigned to this class. For example, a distributed memory cluster typically consists of shared memory SMP nodes which are really of NUMA class if account low level hardware issues like the cache memory. GPGPU as well integrates different types of classical memory architecture: local/private memory of cores corresponds to distributed memory architecture as well as the whole GPU device being viewed as a PU, shared/local memory of a compute unit resembles shared memory architecture at thread block/work group level, but in use together with global/constant data cache provides NUMA features at the device level.

Contrariwise, the software development environments traditionally tried to hide these implementation difficulties. In CPU-based parallel programming either pure SMP/shared or pure distributed memory model is proposed. (Naturally there are sophisticated approaches to account real hardware architecture.) But both GPGPU and Cell programming require accounting of their hybrid architecture specificity.

3.2.2 Classification of parallel software development tools

In 1988 four distinct paths for application software development on parallel computers were identified by McGraw and Axelrod [26]:

1. extend an existing compiler to translate sequential programs into parallel programs,
2. extend an existing language with new operations that allow users to express parallelism,
3. add a new language layer on top of an existing sequential language,
4. define a totally new parallel language.

Currently two additional approaches became popular:

1. provide an Application Program Interface (API) to utilize existent compilers in concurrent execution environments,
2. create efficient parallel solutions (libraries, objects, services) for typical problems to be called from sequential programs.

Compiler extension

Design parallelizing compilers that exploit parallelism in existing programs written in a sequential language.

1. Advantages:
   (a) billions of dollars and thousands of years of programmer effort have already gone into legacy programs,
   (b) automated parallelization can save money and labour,
   (c) it has been an active area of research for over twenty years.
2. Disadvantages:

(a) pits programmer and compiler in game of hide and seek. The programmer hides parallelism in loops and control structures, and the compiler might irretrievably lose some parallelism.

ParaWise (previously known as CAPTools) takes an existing serial FORTRAN 77, FORTRAN 90 or FORTRAN 95 code and automatically generates parallel programs for OpenMP, MPI, PVM and Cray SHMEM. Modern general purpose compilers such as Intel and Portland Group vectorize cycles to use low level data parallelism of modern CPUs.

Programming language extension

Extend a sequential language with constructions of parallel programming.

1. Advantages:
   (a) easy to learn as is based on a wide known language,
   (b) gives programmers flexibility with respect to program development.
2. Disadvantages:
   (a) requires development of new compilers, but on the base of existent ones,
   (b) Some parallel language extensions such as C* were not adapted as standard compromising severely portable code.

The recent developments of this type were inspirited by the emerging stream architecture. Brook of the Stanford University Graphics group's in the forms of BrookGPU and ATI/AMD Brook+ is used for GPGPU on the base of OpenGL v1.3+, DirectX v9+ or AMD's Close to Metal for the computational backend and runs on both Microsoft Windows and Linux. Unlike Occam and CHILL Brook is not a completely new language but rather a dialect of ANSI C with small deviations related to specification of multiple GPU memory types. Its short history looks to be finishing with AMD transition to Khronos group's OpenCL (Open Computing Language), that is another C dialect currently supporting all main OS, GPU and Cell hardware. The strongest competitor in the GPGPU area, NVIDIA's CUDA, was initially a C dialect, but since v.3 proposes full-scale C++ with similar extensions.

The title 'C/C++ Language Extensions for Cell Broadband Engine Architecture' says for itself. It is another C++ dialect for the hardware-specific parallel programming. Among the listed C/C++ extensions OpenCL only is standard and portable.

Parallel programming layer

Let us consider a parallel program consisting of two layers. The bottom layer contains the core of the computation which manipulates its portion of data to gets its result. The upper layer controls creation and synchronization of processes. A compiler would then translate these two levels into code for execution on parallel machines.

1. Advantages:
   (a) allows users to depict parallel programs as directed graphs with nodes depicting sequential procedures and arcs representing data dependencies among procedures.
2. Disadvantages:
   (a) requires programmer to learn and use a new parallel programming system.

The most common example is OpenMP. OpenMP programmer adds special pragma / comment instructions in the sequential program to express its parallel behavior. A program can be converted from a sequential to parallel form in reliable stepwise manner. Besides, any changes can be easy debugged with use of its persisting sequential version.

Another approach is represented by Linda coordination language. It targets parallel program development from scratch. First the program parallel skeleton is developed with use of only 6 Linda operators. Then the skeleton is translated to a sequential language for further development. Whereas message-passing models require tightly-coupled processes sending messages to each other in some sequence or protocol, Linda processes are decoupled from other processes, communicating only through the tuplespace; a process need have no notion of other processes except for the kinds of tuples consumed or produced (data coupling). Linda implementations can be found for Prolog, Ruby (Rinda), Python, C, Smalltalk, Java, TCP and Lisp. There are

other close initiatives like JavaSpaces, TSpaces and a visual language called Graphical Calculus of Communicating Systems (GCCS).

Specialized language

Develop a parallel language from scratch. Let the programmer express parallel operations explicitly.

1. Advantages:
   (a) explicit parallelism means programmer and compiler are now allies instead of adversaries.
2. Disadvantages:
   (a) requires development of new compilers. It typically takes years for vendors to develop high-quality compilers for their parallel architectures,
   (b) user resistance – few programmers want to learn another language.

The programming languages Occam (the late 1980s' Inmos transputer program language) and CHILL (CCITT language used in telecommunications switches) are such famous examples. Despite promoting both those tools have failed to expand. A free CHILL compiler bundled with GCC up to version 2.95 was removed from the later versions. In late 1999 CCITT stopped maintaining the CHILL standard, but it was later supported by ISO. The last revision of the 'CHILL — The ITU-T programming language' standard is ISO/IEC 9496:2003.

Application program interface

Extend a sequential language with API functions that allow programmers to create, terminate, synchronize and communicate with parallel processes. This does not suppose development of specialized compilers but rather utilization of existing language mechanisms for expression of concurrency.

1. Advantages:
   (a) easiest, quickest, and least expensive implementation since it only requires the development of a subroutine library,

   (b) gives programmers flexibility with respect to program development,

   (c) Libraries meeting the MPI standard exist for almost every parallel computer.

2. Disadvantages:

   (a) compiler is not involved in generation of parallel code therefore it cannot flag errors,

   (b) it is very easy to write parallel programs that are difficult to debug.

MPI, PVM, BSPlib, Titanium for Java are typical examples. It's important to highlight that some parallel programming tools such as fork/spawn functions, sockets, events have been available from sequential languages for decades. POSIX threads are among the most used tools of parallel programming. API tools are attributed sometimes to the programming language extension class.

Parallel mathematical libraries

Exploit high-performance parallel implementations of mathematical methods from serial programs. Avoid parallel programming at all.

1. Advantages:

   (a) easiest and quickest for application development from scratch,

   (b) reliability, verified performance, easiest debug,

   (c) gives programmers ready-to-use efficient solutions for typical problems.

2. Disadvantages:

   (a) nothing but solutions of typical problems can be paralleled,

   (b) need to fit and learn a library for almost every new parallel computer architecture and class of problems,

   (c) overheads for the data transformation into formats of each particular function.

Scientific and technical HPC problems can be frequently reduced to computer solutions of classical math problems or their combinations. From this viewpoint one can concentrate efforts on high performance solution of those problems and neglect paralleling other program parts.

Common types of problems found in parallel computing applications are:

1. dense linear algebra,
2. sparse linear algebra,
3. spectral methods (such as Cooley-Tukey fast Fourier transform),
4. n-body problems (such as Barnes-Hut simulation),
5. structured grid problems (such as Lattice Boltzmann methods),
6. unstructured grid problems (such as found in finite element analysis),
7. Monte Carlo simulation,
8. combinational logic (such as brute-force cryptographic techniques),
9. graph traversal (such as sorting algorithms),
10. dynamic programming,
11. branch and bound methods,
12. graphical models (such as detecting hidden Markov models and constructing Bayesian networks),
13. finite-state machine simulation.

Many libraries support parallel linear algebra implementations for computer clusters: PLAPACK, ScaLAPACK, MKL, HSL, Inparsoft. Some of them solve other math problems like systems of non-linear or differential equations. MATLAB Distributed Computing Server Perform MATLAB and Simulink computations on computer clusters and server farms. In the GPGPU area NVIDIA proposes CUDPP for some data parallel primitive operations, cuFFT for Fast Fourier Transforms, and cuBLAS for linear algebra. AMD also provides the AMD Computation and Math Library (ACML), comprised of BLAS and LAPACK for linear algebra, FFT, math transcendental, and pseudo-random number generation libraries. STI Center of Competence for the Cell Broadband Engine Processor at Georgia Tech proposes free Cell optimized libraries for FFT, MPEG, compression, encryption, and some other software.

3.3 Parallel Computing Hardware

The history of the microprocessor over the last 40 years describes the greatest period of sustained technical progress the world has ever seen. Moore's Law, which describes the rate of this progress, has no equivalent in transportation, agriculture, or mechanical engineering. The first thirty years of the microprocessor focused almost exclusively on serial workloads: compilers, managing serial communication links, user-interface code, and so on. More recently, CPUs have evolved to meet the needs of parallel workloads in markets from financial transaction processing to computational fluid dynamics.

CPUs are easy to program, because compilers evolved right along with the hardware they run on. Software developers can ignore most of the complexity in modern CPUs; microarchitecture is almost invisible, and compiler magic hides the rest. Multicore chips have the same software architecture as older multiprocessor systems: a simple coherent memory model and a sea of identical computing engines. But CPU cores continue to be optimized for single-threaded performance at the expense of parallel execution. This fact is most apparent when one considers that integer and floating-point execution units occupy only a tiny fraction of the die area in a modern CPU [14].

Figure 3.3 shows the portion of the die area used by ALUs in the Core i7 processor (the chip code-named Bloomfield) based on Intel's Nehalem microarchitecture.

With such a small part of the chip devoted to performing direct calculations, it's no surprise that CPUs are relatively inefficient for HPC applications. Most of the circuitry on a CPU, and therefore most of the heat it generates, is devoted to invisible complexity: those caches, instruction decoders, branch predictors, and other features that are not architecturally visible but which enhance single-threaded performance.

At the heart of this focus on single-threaded performance is a concept known as **speculation**. At a high level, speculation encompasses not only speculative execution (in which instructions begin executing even before it is possible to know their results will be needed), but many other elements of CPU design. Caches, for example, are fundamentally speculative: storing data in a cache represents a bet that the data will be needed again soon. Caches consume die area and power that could otherwise be used to implement and operate more execution units. Whether the bet pays off depends on the

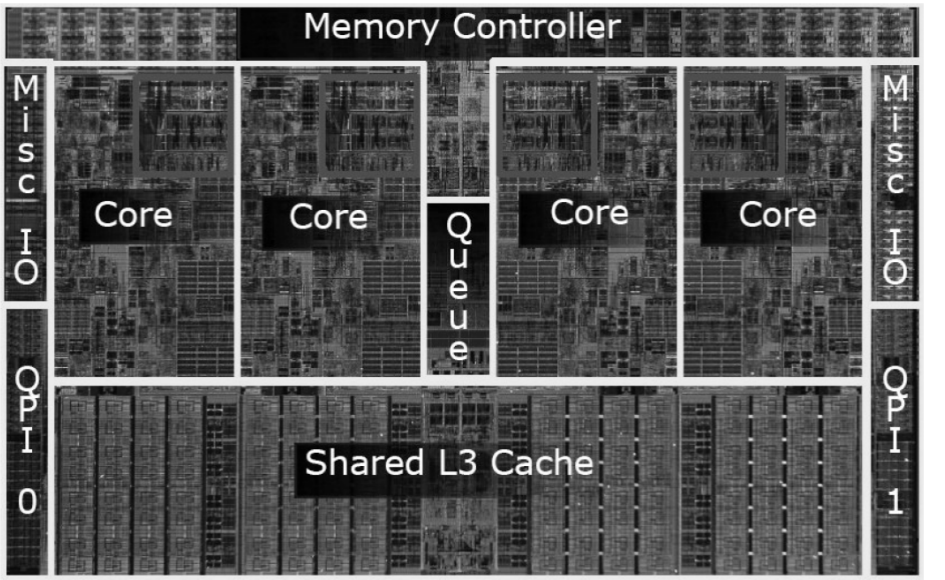


Figure 3.3: Intel's Core i7 CPU. Red outlines highlight the portion of each core occupied by execution units

nature of each workload. Similarly, multiple execution units, out of order processing, and branch prediction also represent speculative optimizations. All of these choices tend to pay off for code with high data locality (where the same data items, or those nearby in memory, are frequently accessed), a mix of different operations, and a high percentage of conditional branches. But when executing code consisting of many sequential operations of the same type – like scientific workloads – these speculative elements can sit unused, consuming die area and power.

The market demands general-purpose processors that deliver high singlethreaded performance as well as multi-core throughput for a wide variety of workloads on client, server, and HPC systems. This pressure has given us almost three decades of progress toward higher complexity and higher clock rates. This progress hasn't always been steady. Intel cancelled its Tejas processor, which was rumored to have a 40-stage pipeline, and later killed off the entire Pentium 4 NetBurst product family because of its relative inefficiency. The Pentium 4 ultimately reached a clock rate of 3.8 GHz in the 2004 Prescott model, a speed that Intel has been unable to match since. In the more recent Core 2 (Conroe/Penryn) and Core i7 (Nehalem) processors, Intel uses increased complexity to deliver substantial performance improvements over the Pentium 4 line, but the pace of these improvements is slowing. Each new

generation of process technology requires ever more heroic measures to improve transistor characteristics; each new core microarchitecture must work disproportionately harder to find and exploit instruction-level parallelism (ILP).

As these challenges became more apparent in the 1990s, CPU architects began referring to the 'power wall', the 'memory wall', and the 'ILP wall' as obstacles to the kind of rapid progress seen up until that time. It may be better to think of these issues as mountains rather than walls – mountains that begin as mild slopes and become steeper with each step, making further progress increasingly difficult. Nevertheless, the inexorable advance of process technology provided CPU designers with more transistors in each generation.

By 2005, the competitive pressure to use these additional transistors to deliver improved performance (at the chip level, if not at the core level) drove AMD and Intel to introduce dual-core processors. Since then, the primary focus of PC processor design has been continuing to increase the core count on these chips. That approach, however, has reached a point of diminishing returns. Dual-core CPUs provide noticeable benefits for most PC users, but are rarely fully utilized except when working with multimedia content or multiple performance-hungry applications. Quad-core CPUs are only a slight improvement, most of the time. Recently there is a six-core CPU, but it will likely be difficult to sell most desktop customers on the value of the additional cores. Selling further increases will be even more problematic.

3.3.1 Networks for cluster computing

The well known Moore's law gives us an estimation that the number of transistors on a chip doubles every two years. In addition, Moore's law also means decreasing cost of manufacturing and therefore price of the product decreases for consumers. From 1993 to 2000, Internet and the number of networked hosts with graphical user interface grew rapidly and the price of personal computer decreased at the same time with network equipment.

After year 2000 the evolution has continued even faster. There have been only three years between Ethernet standards from 100 Mbit/s to 1 Gigabit and four years from 1 Gbit/s to 10 Gigabit. 100 Gigabit Ethernet standard IEEE 802.3ba approval is scheduled for June 2010, that is just 3 years after

10 Gigabit one. The network bandwidth raise has surpassed the Moore's law.

These changes created synergy with wide distribution of Beowulf HPC cluster architecture. As Beowulf clusters consist of commodity hardware their performance was initially restricted by big communication overheads. To minimize them, initial Beowulf design was based on hypercube of coupled Gigabit Ethernet inter-node connections.

HPC requests for high interconnect speed stimulated significant changes in LAN communication technologies. The first was Myrinet, ANSI/VITA 26-1998, a high-speed local area networking system designed by Myricom to be used as an interconnect between multiple machines to form computer clusters [40]. Myrinet had much lower protocol overhead than standards such as Ethernet, and therefore provided better throughput, less interference, and lower latency while using the host CPU. Although it can be used as a traditional networking system, Myrinet is often used directly by programs that 'know' about it, thereby bypassing a call into the operating system.

Myrinet physically consists of two fibre optic cables, upstream and downstream, connected to the host computers with a single connector. Machines are connected via low-overhead routers and switches, as opposed to connecting one machine directly to another. Myrinet includes a number of fault-tolerance features, mostly backed by the switches. These include flow control, error control, and 'heartbeat' monitoring on every link. The fourth-generation Myrinet of 2005, called Myri-10G, supports a 10 Gbit/s data rate and is interoperable with 10 Gigabit Ethernet on the physical layer (cables, connectors, distances, signaling). In 2005 Myrinet was used by 28.2% of Top500 supercomputers, but later its part has reduced to 1.4% under the press of Infiniband and 10 Gigabit Ethernet.

The Virtual Interface Architecture (VIA) is an abstract model of a user-level zero-copy network, and is the basis for InfiniBand and iWARP [40]. Created by Microsoft, Intel, and Compaq, the original VIA sought to standardize the interface for high-performance network technologies known as System Area Networks (SANs; not to be confused with Storage Area Networks).

In traditional networks such as Ethernet, the network as a shared resource is protected by the kernel, which presents a tremendous performance bottleneck when latency is an issue. A virtual network protected across process boundaries has been implemented by a network interface card (NIC). The virtual interface (VI) of VIA refers to this network and is merely the destination of

the user's communication requests. Communication takes place over a pair of VIs, one on each of the processing nodes involved in the transmission. In the 'kernel-bypass' communication, the user manages its own buffers.

Another facet of traditional networks is that arriving data is placed in a pre-allocated buffer and then copied to the user-specified final destination. Copying large messages can take a long time, and so eliminating this step is beneficial. With direct memory access (DMA) a device can access main memory directly while the CPU is free to perform other tasks. In a network with remote direct memory access (RDMA), the sending NIC uses DMA to read data in the user-specified buffer and transmit it as a self-contained message across the network. The receiving NIC then uses DMA to place the data into the user-specified buffer. There is no intermediary copying and all of these actions occur without involvement of the CPUs, which has an added benefit of lower CPU utilization.

For the NIC to actually access the data through DMA, the user's page must be in memory. In VIA, the user must 'pin-down' its buffers before transmission, so as to prevent the OS from swapping the page out to the disk. This action – one of the few that involves the kernel – ties the page to physical memory. To ensure that only the process that owns the registered memory may access it, the VIA NICs require permission keys known as 'protection tags' during communication.

Acceleration of communications provided conditions for the extensive raise of cluster size and performance. HPC clusters quickly increased up to hundreds and even thousand nodes. As result power consumption and related space/conditioning limitations became a bottleneck of further performance increasing.

These changes has destroyed Intel Itanium as too hot and reinforced concept of 'green computing' in the HPC domain. The issue of insufficient performance of a single PC node returned on the new level.

3.3.2 Specialized microprocessor architectures

The illusion of a flat and uniformly accessible memory provided by traditional CPUs is increasingly costly to maintain. Latencies to main memory, in spite of designer's best efforts range in the several hundreds of cycles and

approach a thousand cycles in multi-GHz SMP systems. With such a large penalty associated with a cache miss, managing memory locality becomes the main factor determining software performance. Developers of compilers and high performance software alike spend much of their time reverse-engineering and defeating the sophisticated mechanisms that automatically bring data on to and off the chip. Given the large number of transistors devoted to these mechanisms this is an unsatisfactory situation. Power limitations and limitations on main memory access latency stimulate a re-evaluation of microprocessor architecture.

Green computing primary targeted to minimal impact on the environment also strives to achieve economic viability and improved system use, while abiding by our social and ethical responsibilities. Thus, green IT includes the dimensions of environmental sustainability, the economics of energy efficiency, and the total cost of ownership, which includes the cost of disposal and recycling. For the HPC clusters with their minimalist node design the power necessary for cooling is the main source of the environment impact. CPU is commonly a hottest heater on the motherboard, despite its small size, because of extremely high power density raising with transition to smaller elements (Fig. 3.4). The technology restrictions related to this effect are known as a 'power wall'.

The Cell processor architecture [17] incorporated many new ideas to improve microprocessor efficiency. On the Cell processor an 64-bit Power Processor Element (PPE), a traditional CPU of IBM Power Architecture, is combined with multiple Synergistic Processor Elements (SPEs) and associated memory transfer mechanisms. Memory translation and protection of Cell are consistent with the Power Architecture. It supports multiple operating systems and virtualization. In particular real-time operating systems (such as an embedded or game OS) and non-real time OS (such as Linux) multiple Cell processors can run simultaneously alike Power processors. The PPE virtualization layer governs the allocation of resources including the SPEs to the various OS partitions. Unlike PPE, the SPEs operate only on their local memory called local store (LS). Code and data must be transferred into the associated LS for an SPE to execute or operate on LS addresses. Each the LS address have an alias in the PPE address map, and synchronization transfers between the SPE LSs and the memory are coherent. As result a pointer to a data structure that has been created on the PPE can be passed to an SPE and the SPE can use this pointer to issue a DMA command to bring the data structure into its LS in order to perform operations on it. If after operating on this data structure the SPE (or PPE) issues a DMA

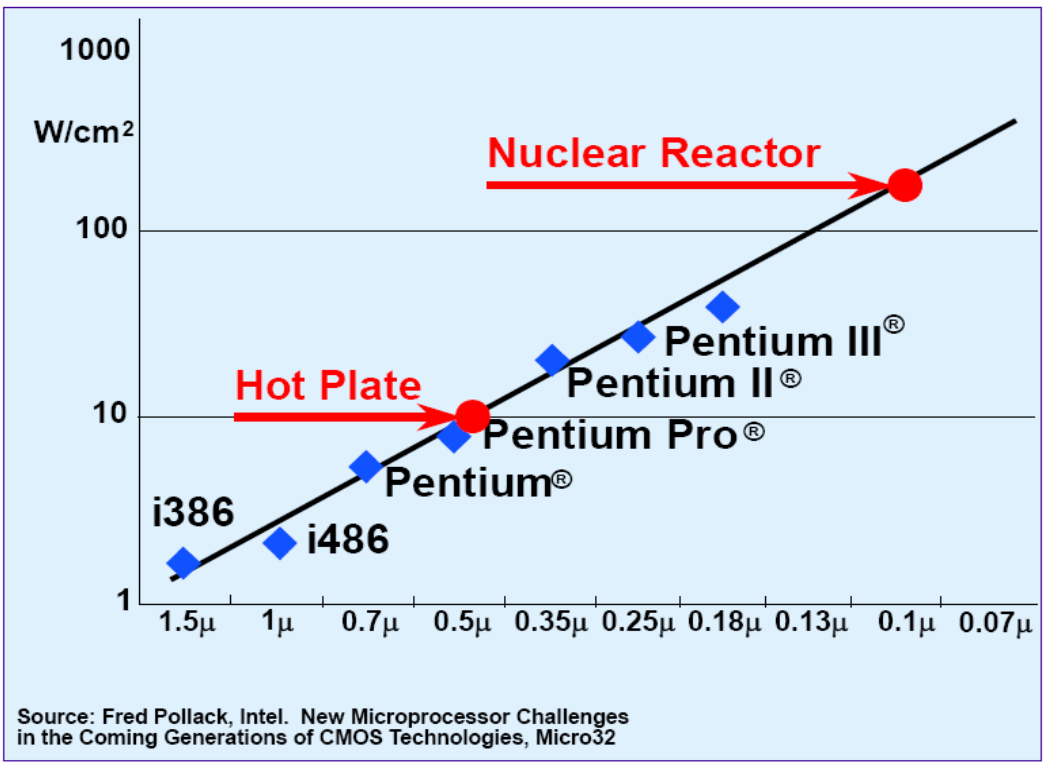


Figure 3.4: Power density of desktop Intel x86 CPUs

command to place it back in non-LS memory, the transfer is again coherent in the system according to the normal Power memory ordering rules.

Lower speed of SPEs together with the big multilevel register/local cache memory and asynchronous DMA to the global RAM help to bypass the fundamental for SMP 'memory wall'.

Cell processor architecture addresses the 'power wall' problem by reducing transistor energy stream through decreasing the operating voltage, optimization of the chip design (smaller channel length) and the technology (limited oxide thickness scaling). Besides, the SPE cores are relatively simple and due to absent speculation execute less transistor switching per command.

Vector conveyer operations of multiple concurrent SPEs compensate lower speed of individual threads; high design frequency attack the 'frequency wall' for embarrassingly parallel and fine-grained operations.

Equivalents of the Power processor locking instructions, as well as a memory mapped mailbox per SPE, are used for synchronization and mutual exclusion. Figure 4.5 shows the Cell processor with highlighted execution units of both PPE and SPEs to compare with a multicore Intel processor (Fig. 3.5).

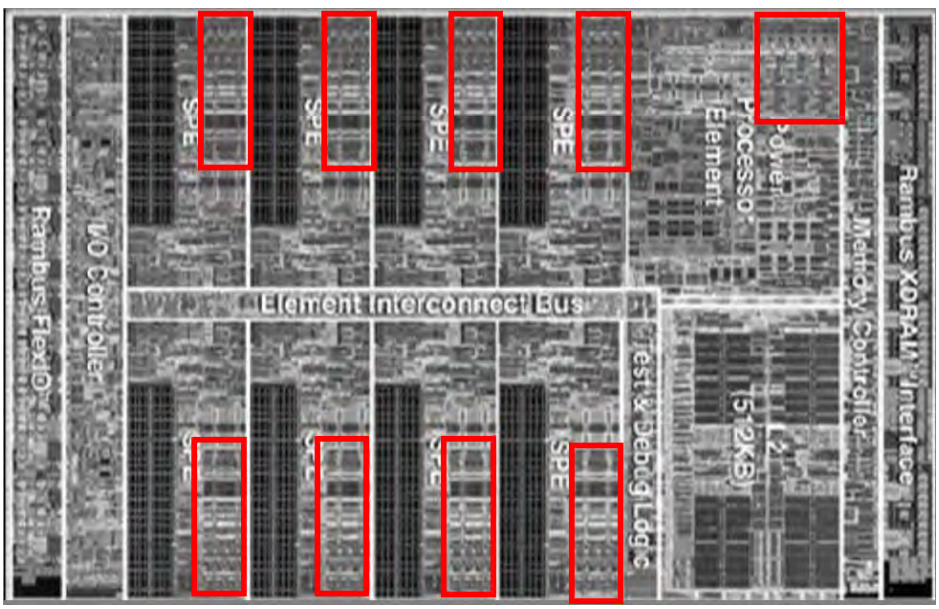


Figure 3.5: IBM's PowerXCell 8i of Cell architecture. Red outlines highlight the portion of each core occupied by execution units

While the Cell architecture looks revolutionary in comparison with the traditional one, it is just a halfhearted measure from the GPGPU stream viewpoint.

GPGPU programming evolved as a way to perform non-graphics processing on graphics-optimized SIMD architectures, typically by running carefully crafted shader code against data presented as vertex or texture information and retrieving the results from a later stage in the pipeline. Though sometimes awkward, GPGPU programming showed great promise.

There were attempts to build chip-scale parallel processors in the 1990s, but the limited transistor budgets in those days favored more sophisticated single-core designs. The real path toward GPU computing began, not with GPUs, but with nonprogrammable 3D-graphics accelerators. Multi-chip 3D rendering engines were developed by multiple companies starting in the 1980s,

but by the mid-1990s it became possible to integrate all the essential elements onto a single chip. From 1994 to 2001, these chips progressed from the simplest pixel-drawing functions to implementing the full 3D pipeline: transforms, lighting, rasterization, texturing, depth testing, and display.

In late 2006, NVIDIA and ATI (acquired by AMD in 2006) began marketing GPU boards as streaming coprocessor boards [18]. Both companies released software to expose the processor to application programmers. Since then the hardware has grown more powerful and the available software has become more high-level and developer-friendly. Streaming architecture have radical changed rate of computing units per chip (see Fig. 3.6). Since most of the circuitry within each core is dedicated to computation, rather than speculative features meant to enhance single-threaded performance, most of the die area and power consumed by Fermi goes into the application's actual algorithmic work [14].

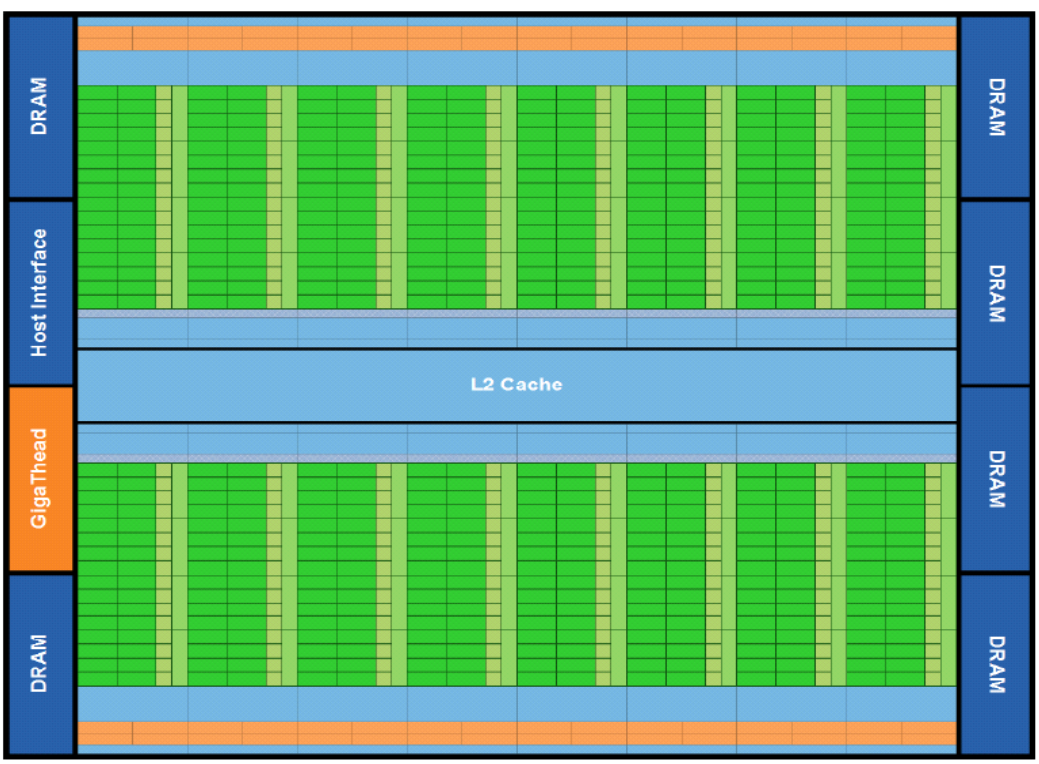


Figure 3.6: NVIDIA's Fermi GPU consists of 32 streaming multiprocessors (SMs), each consisting of 32 cores. Green highlights the portion of GPU occupied by execution units

Unlike the early attempts at chip-scale multiprocessing back in the '90s, NVIDIA Tesla and AMD FireStream were high-volume hardware platforms right from the beginning. General purpose computing with GPUs became feasible when vendors made it possible to program them without using graphics primitives. Higher level control has gradually become available, to the point where now both NVIDIA and AMD offer a tool chain based on a slightly enhanced version of C. The basic programming model is to write ordinary C code that runs on the CPU for sequential operations and for control, with the parallel code segregated into kernels (data parallel functions) that run in parallel on the GPU.

Although GPU computing is only a few years old now, it's likely there are already more programmers with direct GPU computing experience than have ever used a Cray. NVIDIA says it has shipped over 100 million CUDA-capable chips. Academic support for GPU computing is also growing quickly. NVIDIA says over 200 colleges and universities are teaching classes in CUDA programming; the availability of OpenCL will drive that number even higher.

GPU computing isn't meant to replace CPU computing. Each approach has advantages for certain kinds of software. As explained earlier, CPUs are optimized for applications where most of the work is being done by a limited number of threads, especially where the threads exhibit high data locality, a mix of different operations, and a high percentage of conditional branches. GPU design aims at the other end of the spectrum: applications with multiple threads that are dominated by longer sequences of computational instructions. Over the last few years, GPUs have become much better at thread handling, data caching, virtual memory management, flow control, and other CPU-like features, but the distinction between computationally intensive software and control-flow intensive software is fundamental.

Here are some impressive reports from HPC community members succeeded with GPU, all reporting a speedup of two orders of magnitude or more:

- 1080x: Monte Carlo simulation of photon migration,
- 675x: stochastic differential equations,
- 470x: k nearest neighbor search,
- 420x: generalized harmonic analysis,
- 340x: power system simulation,

- 300x: cone beam computed tomography,
- 270x: Particle swarm optimization,
- 263x: FHD-spiral MRI reconstruction,
- 250x: N-body simulation,
- 172x: support vector machine training and classification,
- 169x: signal and image reconstruction,
- 130x: quantum chemistry two-electron integral evolution,
- 120x: 3D particle Boltzmann solver,
- 109x: sliding window object detection,
- 100x: visualization of volumetric white matter connectivity,
- 100x: prestack seismic data interaction,
- 100x: folding@home,
- 100x: pricing and risk management of exotic financial structures,
- 100x: incompressible Navier-Stokes solver.

While GPGPU allows high performance solution of many problems it is not suitable for solution of many others. In fact only embarrassingly parallel applications with few memory volume per atomic task are guaranteed to be efficiently paralleled on GPUs. Fine grained applications are frequently suitable for GPUs, but device-level memory use significantly decreases performance.

3.4 Parallel Programming

The section introduces popular software development tools for the described hardware architectures. We have selected OpenMP for shared memory computers as more relevant for HPC than POSIX threads and MPI for distributed memory computers as more universal and wider distributed than

PVM. Choice of GPU programming tool is more complex: OpenCL is recognized as a standard cross-platform program interface for both Cell and GPUs, but NVIDIA dominates on the stream processor market, and its CUDA implementation completely fitted to hardware can provide some performance benefits for NVIDIA's GPUs. Despite this we have selected OpenCL. Besides, to avoid language mix issues all the next samples are in C, that is common for the listed platforms.

To discuss the implementation issues we will use a simple N-Body simulation. This algorithm is used frequently in demonstrations of computational performance and is an interesting algorithm for several reasons. First, the simulation of the motion of particles subject to particle-particle interactions represents a general class of algorithms with applications ranging from chemistry to astrophysics. Second, the scaling of the algorithm is $O(N)^2$ in computation and $O(N)$ in communication, where N is the number of particles. This makes N a convenient tuning parameter for studying the performance of different architectures. For small or large N, one expects an architecture to be relatively communication or compute bound, respectively, and, measured performance can provide valuable information about an underlying architecture. Finally, the algorithm is relatively simple and easy to implement making it useful for a tutorial such as this one. The back-ground physics is well known to wide audience. The sequential C version of the N-Body simulation program follows. The code fragment contains just a function implementing the algorithm itself. Other related stuff like main(), data initialization and output is omitted.

```
#include <stdio.h>
#include <math.h>
#define eps 1.0e-12
int nbody(int n, double dt,
   double m[], double x[], double y[], double z[],
   double vx[], double vy[], double vz[],
   double xnew[], double ynew[], double znew[])
{
   double ax, ay, az, dx, dy, dz, d2, f, invr, invr3;
   int i, j, num_impacts, impact;
   num_impacts=0;
   for(i=0; i<n; i++) { /* Foreach particle "i" ... */
     ax=ay=az=0.0; impact=0;
```

```
      for(j=0; j<n; j++) { /*Loop over all particles "j"*/
        if(j==i) continue;
        dx=x[j]-x[i]; dy=y[j]-y[i]; dz=z[j]-z[i];
        d2 = dx*dx+dy*dy+dz*dz;
        invr = 1.0/sqrt(d2+eps);
        invr3 = invr*invr*invr;
        f = m[j]*invr3;
        ax += f*dx; /* accumulate the acceleration from */
        ay += f*dy; /* gravitational attraction */
        az += f*dz;
        if(d2<=eps) impact=1; /*close particles*/
      }
      num_impacts+=impact; /*number of impacts*/
      /* update position of particle "i": */
      xnew[i] = x[i] + dt*vx[i] + 0.5*dt*dt*ax;
      ynew[i] = y[i] + dt*vy[i] + 0.5*dt*dt*ay;
      znew[i] = z[i] + dt*vz[i] + 0.5*dt*dt*az;
      /* update velocity of particle "i": */
      vx[i] += dt*ax; vy[i] += dt*ay; vz[i] += dt*az;
    }
    return num_impacts;
}
```

3.4.1 OpenMP for Multiprocessor/Multicore Shared Memory SMP

OpenMP [34] is a collection of compiler directives and library functions that are used to create parallel programs for shared-memory computers. OpenMP is combined with C, C++, or Fortran to create a multithreading programming language; that is, the language model is based on the assumption that the UEs are threads that share an address space. Its support under both Windows and Linux is provided by the general purpose compilers.

The formal definition of OpenMP is contained in a pair of specifications, one for Fortran and the other for C/C++. They differ in some minor details, but for the most part, a programmer who knows OpenMP for one language can pick up the other language with little additional effort.

OpenMP is based on the fork/join programming model. An executing OpenMP

program starts as a single thread. At points in the program where parallel execution is desired, the program forks additional threads to form a team of threads. The threads execute in parallel across a region of code called a parallel region. At the end of the parallel region, the threads wait until the full team arrives, and then they join back together. At that point, the original or master thread continues until the next parallel region (or the end of the program).

OpenMP was designed around two key concepts: sequential equivalence and incremental parallelism. A program is said to be *sequentially equivalent* when it yields the same1 results whether it executes using one thread or many threads. A sequentially equivalent program is easier to maintain and, in most cases, much easier to understand (and hence write). *Incremental parallelism* refers to a style of parallel programming in which a program evolves from a sequential program into a parallel program. A programmer starts with a working sequential program and finds pieces of code that are worthwhile to execute in parallel. Thus, parallelism is added incrementally. At each phase of the process, there is a working program that can be verified, greatly increasing the chances that the project will be successful.

It is not always possible to use incremental parallelism or to create sequentially equivalent OpeniMP programs. Sometimes a parallel algorithm requires complete restructuring of the analogous sequential program. In other cases, the program is constructed from the beginning to be parallel and there is no sequential program to incrementally parallelize. Also, there are parallel algorithms that do not work with one thread and hence cannot be sequentially equivalent. Still, incremental parallelism and sequential equivalence guided the design of the OpenMP API and are recommended practices.

OpenMP is an explicitly parallel programming language. The compiler doesn't guess how to exploit concurrency. Any parallelism expressed in a program is there because the programmer directed the compiler to 'put it there'. To create threads in OpenMP, the programmer designates blocks of code that are to run in parallel. This is done in C and C++ with the pragma:

```
#pragma omp directive-name [clause[ clause] ... ]
```

Each pragma relates to a structured block that is either the next C statement or the next sequence of C statements put in braces { and }. An OpenMP program is not allowed to branch into or out of a structured block. To

attempt to do so is a fatal error generally caught at compile time. Likewise, the structured block cannot contain a return statement. The only branch statements allowed are those that shut down the entire program (exit() in C). Some examples of OpenMP pragmas in C or C++ follow:

```
#pragma omp parallel private(ii, jj, kk)
#pragma omp barrier
#pragma omp for reduction(+:result)
```

When using the parallel construct alone, every thread executes the same block of statements. There are times, however, when we need different code to map onto different threads. This is called *worksharing*.

The most commonly used worksharing construct in OpenMP is the construct to split loop iterations between different threads. Designing a parallel algorithm around parallel loops is an old tradition in parallel programming. This style is sometimes called loop splitting. In this approach, the programmer identifies the most time-consuming loops in the program. Each loop is restructured, if necessary, so the loop iterations are largely independent. The program is then parallelized by mapping different groups of loop iterations onto different threads. For example, consider the described program part:

```
#include <stdio.h>
#include <math.h>
#define eps 1.0e-12
int nbody(int n, double dt,
   double m[], double x[], double y[], double z[],
   double vx[], double vy[], double vz[],
   double xnew[], double ynew[], double znew[])
{
   double ax, ay, az, dx, dy, dz, d2, f, invr, invr3;
   int i, j, num_impacts, impact;
   num_impacts=0;
#pragma omp parallel for private(i,j,num_close)\textbackslash
 private(ax,ay,az,dx,dy,dz,d2,f,invr,invr3)\textbackslash
 reduce(+,num_impacts)
```

```
   for(i=0; i<n; i++) { /* Loop is converted to fork */
     ax=ay=az=0.0; impact=0;
     for(j=0; j<n; j++) { /*Loop over all particles "j"*/
       if(j==i) continue;
       dx=x[j]-x[i]; dy=y[j]-y[i]; dz=z[j]-z[i];
       d2 = dx*dx+dy*dy+dz*dz;
       invr = 1.0/sqrt(d2+eps);
       invr3 = invr*invr*invr;
       f = m[j]*invr3;
       ax += f*dx; /* accumulate the acceleration from */
       ay += f*dy; /* gravitational attraction */
       az += f*dz;
       if(d2<=eps) impact=1; /*close particles*/
     }
     num_impacts+=impact; /*number of impacts*/
     /* update position of particle "i" */
     xnew[i] = x[i] + dt*vx[i] + 0.5*dt*dt*ax;
     ynew[i] = y[i] + dt*vy[i] + 0.5*dt*dt*ay;
     znew[i] = z[i] + dt*vz[i] + 0.5*dt*dt*az;
     /* update velocity of particle "i" */
     vx[i] += dt*ax; vy[i] += dt*ay; vz[i] += dt*az;
   } /* Join is supposed to be executed here */
   return num_impacts;
}
```

Just one OpenMP pragma has been used to parallelize the code. The function can be paralleled by OpenMP without any changes in other parts of the program source code.

The pragma directive parallel means beginning of parallelized block. This time it contains the only statement `for(i=0; i<n; i++){...}`. The pragma directive `for` means the loop have to be vectorized. These directives can be set separately with the same effect:

```
 #pragma omp parallel {          #pragma omp parallel for
   #pragma omp for         =       for(...) {
   for(...) {...}                  ...
 }                                 }
```

The pragma directive `private` specifies which variables to be cloned for each thread. In C++ case any variables declared in a parallel region are cloned. Other variables are shared.

Several other clauses change how variables are shared between threads. The most commonly used ones follow.

1. `firstprivate`(*list*). Just as with private, for each name appearing in the list, a new private variable for that name is created for each thread. Unlike private however, the newly created variables are initialized with the value of the variable that was bound to the name in the region of code preceding the construct containing the `firstprivate` clause. This clause can be used on both parallel and the worksharing constructs.

2. `lastprivate`(*list*). Once again, private variables for each thread are created for each name in the list. In this case, however, the value of the private variable from the sequentially last loop iteration is copied out into the variable bound to the name in the region following the OpenMP construct containing the `lastprivate` clause. This clause can only be used with the loop-oriented workshare constructs.

The final clause used in the sample is the `reduction` clause. A reduction is an operation that, using a binary, associative operator, combines a set of values into a single value. Reductions are very common and are included in most parallel programming environments. In OpenMP, the reduction clause defines a list of variable names and a binary operator. For each name in the list, a private variable is created and initialized with the value of the identity clement for the binary operator (for example, zero for addition). Each thread carries out the reduction into its copy of the local variable associated with each name in the list. At the end of the construct containing the reduction clause, the local values are combined with the associated value prior to the OpenMP construct in question to define a single value. This value is assigned to the variable with the same name in the region following the OpenMP construct containing the reduction.

In the example, the `reduction` clause was applied with the + operator to compute a summation and leave the result in the variable 'num_impacts'.

Although the most common reduction involves summation, OpenMP also supports reductions with the standard C/C++ operators *, -, &, |, ^, &&,

and ||. The C and C++ languages do not include a number of useful intrinsic functions such as 'min' or 'max' within the language definition. Hence, OpenMP can not provide reductions in such cases; if they are required, the programmer must code them explicitly by hand.

Variables generally can appear in the list of only a single data clause. The exception is with `lastprivate` and `firstprivate`, because it is quite possible that a private variable will need both a well-defined initial value and a value exported to the region following the OpenMP construct in question.

In this example the runtime system is allowed to select the number of threads. This is the most common approach. It is possible to change the operating system's default number of threads to use with OpenMP applications by setting the `OMP_NUM_THREADS` environment variable. The number of threads to be used can also be set inside the program with the `num_threads` clause. For example, to create a parallel region with three threads, the programmer would use

```
#pragma omp parallel num_threads(3)
```

Many OpenMP programs can be written using only the `parallel` and `parallel for` constructs. There are algorithms, however, where one needs more careful control over how variables are shared. When multiple threads read and write shared data, the programmer must ensure that the threads do not interfere with each other, so that the program returns the same results regardless of how the threads are scheduled. This is of critical importance since as a multithreaded program runs, any semantically allowed interleaving of the instructions could actually occur. Hence, the programmer must manage reads and writes to shared variables to ensure that threads read the correct value and that multiple threads do not try to write to a variable at the same time.

The major synchronization constructs in OpenMP are the following.

- `flush` defines a synchronization point at which memory consistency is enforced. Basically, a modern computer can hold values in registers or buffers that are not guaranteed to be consistent with the computer's memory at any given point. Cache coherency protocols guarantee that all processors ultimately see a single address space, but they do not guarantee that memory references will be consistent at every point in time. The syntax of flush is:

```
#pragma omp flush [(list)]
```

where `list` is a comma-separated list of variables that need to be flushed. If the list is omitted, all variables visible to the calling thread will be flushed. Programmers only rarely need to call flush because it is automatically inserted at most points where it is needed. Typically it is used only by programmers building their own low-level synchronization primitives,

– `barrier` provides a synchronization point at which the threads wait until every member of the team has arrived before any threads continue. The syntax of a barrier is:

```
#pragma omp barrier
```

A barrier can be added explicitly, but it is also implied where it makes sense (such as at the end of parallel or worksharing constructs). A barrier implies a flush.

By default, there is an implicit barrier at the end of any OpenMP workshare construct; that is, all the threads wait at the end of the construct and only proceed after all of the threads have arrived. This barrier can be removed by adding a nowait clause to the worksharing construct:

```
#pragma omp for nowait
```

– `critical` implements a critical section for mutual exclusion. In other words, only one thread at a time will execute the structured block within a critical section. The other threads will wait their turn at the top of the construct. The syntax of a critical section is:

```
#pragma omp critical [(name)]{a structured block}
```

where name is an identifier that can be used to support disjoint sets of critical sections. A critical section implies a call to flush on entry to and on exit from the critical section.

The example code may be rewritten using critical section instead of reduction:

```
#include <stdio.h>
#include <math.h>
#define eps 1.0e-12
int nbody(int n, double dt,
   double m[], double x[], double y[], double z[],
   double vx[], double vy[], double vz[],
   double xnew[], double ynew[], double znew[])
{
   double ax, ay, az, dx, dy, dz, d2, f, invr, invr3;
   int i, j, num_impacts, impact;
   num_impacts=0;
#pragma omp parallel for private(i, j, num_close)
 private(ax, ay, az, dx, dy, dz, d2, f, invr, invr3)
   for(i=0; i<n; i++) { /* Loop is converted to fork */
     ax=ay=az=0.0; impact=0;
     for(j=0; j<n; j++) { /*Loop over all particles "j"*/
       if(j==i) continue;
       dx=x[j]-x[i]; dy=y[j]-y[i]; dz=z[j]-z[i];
       d2 = dx*dx+dy*dy+dz*dz;
       invr = 1.0/sqrt(d2+eps);
       invr3 = invr*invr*invr;
       f = m[j]*invr3;
       ax += f*dx; /* accumulate the acceleration from */
       ay += f*dy; /* gravitational attraction */
       az += f*dz;
       if(d2<=eps) impact=1; /*close particles*/
     }
#pragma omp critical
     {
     num_impacts+=impact; /*number of impacts*/
     }
     /* update position of particle "i" */
     xnew[i] = x[i] + dt*vx[i] + 0.5*dt*dt*ax;
     ynew[i] = y[i] + dt*vy[i] + 0.5*dt*dt*ay;
     znew[i] = z[i] + dt*vz[i] + 0.5*dt*dt*az;
     /* update velocity of particle "i" */
     vx[i] += dt*ax; vy[i] += dt*ay; vz[i] += dt*az;
```

```
    } /* Join is supposed to be executed here */
    return num_impacts;
}
```

As much as possible, the syntax of OpenMP is expressed through compiler directives. Certain features of the language, however, can only be handled with runtime library functions. The function prototypes are in the `omp.h` include file. Of the functions within the runtime library, the most commonly used ones are the following:

- `omp_set_num_threads(int new_num_threads)` takes an integer argument and requests that the OS provide that number of threads in subsequent parallel regions,
- `omp_get_num_threads()` returns the actual number of threads in the current team of threads,
- `omp_get_thread_num()` returns the ID of a thread, where the ID ranges from 0 to the number of threads minus 1. The thread with ID of 0 is the master thread,
- `omp_init_lock(omp_lock_t *lock)`. Initialize the lock,
- `omp_destroy_lock(omp_lock_t *lock)`. Destroy the lock, thereby freeing any memory associated with the lock,
- `omp_set_lock(omp_lock_t *lock)`. Set or acquire the lock. If the lock is lice, then the thread calling `omp_set_lock()` will acquire the lock and continue. If the lock is held by another thread, the thread calling `omp_set_lock()` will wait on the call until the lock is available,
- `omp_unset_lock(omp_lock_t *lock)`. Release the lock so some other thread can acquire it,
- `omp_test_lock(omp_lock_t *lock)`. Test or inquire if the lock is available. If it is, then the thread calling this function will acquire the lock and continue. The test and acquisition of the lock is done atomically. If it is not, the function returns nonzero integer and the calling thread continues. This function is used so a thread can do useful work while waiting for a lock.

The lock functions use an opaque data type `omp_lock_t` defined in the `omp.h`. The lock functions guarantee that the lock variable itself is consistently updated between threads, but do not imply a flush of other variables.

The key to performance in a loop-based parallel algorithm is to schedule the loop iterations onto threads such that the load is balanced between threads. OpenMP compilers try to do this for the programmer. Although compilers are excellent at managing data dependencies and are very effective at generic optimizations, they do a poor job of understanding, for a particular algorithm, memory access patterns and how execution times vary from one loop iteration to the next. To get the best performance, programmers need to tell the compiler how to divide the loop iterations among the threads. This is accomplished by adding a schedule clause to the for or do worksharing constructs. The schedule clause takes the form:

```
#pragma omp for schedule(sched [,chunk])
```

where *sched* is cither `static`, `dynamic`, `guided` or `runtime` and *chunk* is an optional integer parameter.

- `schedule (static [,chunk])`. The iteration space is divided into blocks of size chunk. If chunk is omitted, then the block size is selected to provide1 one approximately equal-sized block per thread. The blocks are dealt out to the threads that make up a team in a round-robin fashion. For example, a chunk size of 2 for 3 threads and 12 iterations will create 6 blocks containing iterations (0,1), (2,3), (4,5), (6,7), (8,9), and (10,11) and assign them to threads as [(0,1), (6,7)] to one thread, [(2,3), (8,9)] to another thread and [(4,5), (10,11)] to the last thread.
- `schedule (dynamic [,chunk])`. The iteration space is divided into blocks of size chunk. If chunk is omitted, the block size is set to 1. Each thread is initially given one block of iterations to work with. The remaining blocks are placed in a queue. When a thread finishes with its current block, it pulls the next block of iterations that need to be computed off the queue. This continues until all the iterations have been computed.
- `schedule (guided [,chunk])`. This is a variation of the dynamic schedule optimized to decrease scheduling overhead. As with dynamic scheduling, the loop iterations are divided into blocks, and each thread

is assigned one block initially and then receives an additional block when it finishes the current one. The difference is the size of the blocks: for the first block it is implementation-dependent, but large; block size is decreased rapidly for subsequent blocks, down to the value specified by chunk. This method has the benefits of the dynamic schedule, but by starting with large block sizes, the number of scheduling decisions at runtime, and hence the parallel overhead, is greatly reduced.

– `schedule (runtime)`. The runtime schedule stipulates that the actual schedule and chunk size for the loop is to be taken from the value of the environment variable OMP_SCHEDULE. This lets a programmer try different schedules without having to recompile for each trial.

The definition of the full OpenMP API is only 50 pages of text. The specification also includes more than 25 panes of examples.

3.4.2 MPI for Distributed/Shared Memory Clusters

MPI (Message Passing Interface) [MPI] is the standard programming environment for distributed-memory parallel computers. The central construct in MPI is message passing: one process packages information into a message and sends that message to another process. MPI, however, includes far more than simple message passing. MPI provides routines to synchronize processes, sum numbers distributed among a collection of processes, scatter data across a collection of processes, and much more.

MPI was created in the early 1990s to provide a common message-passing environment that could run on clusters, NUMA computers, MPPs, and even shared-memory machines including SMPs. MPI is distributed in the form of a library, and the official specification defines bindings for C and Fortran, although bindings for other languages have been defined as well. The overwhelming majority of MPI programmers today use MPI version 1.1 (released in 1995). The specification of an enhanced version, MPI 2.0, with parallel I/O, dynamic process management, one-sided communication, and other advanced features was released in 1997. Unfortunately, it was such a complex addition to the standard that for the long time, only a handful of MPI implementations supported MPI 2.0. When it has been already widely implemented its main features became outdated. For example parallel file systems are better administrated and provide usually higher I/O performance than

MPI-IO. As their simultaneous use creates additional overheads MPI-IO is rarely used on modern clusters. Most MPI programs are developed in the boundaries of MPI 1.1 functions.

The MPI approach is based on two core elements: *process groups* and a *communication context*. A process group is a set of processes involved in a computation. In MPI, all the processes involved in the computation are launched together when the program starts and belong to a single group. As the computation proceeds, however, the programmer can divide the processes into subgroups and precisely control how the groups interact.

A communication context provides a mechanism for grouping together sets of related communications. In any message-passing system, messages must be labeled so they can be delivered to the intended destination or destinations. The message labels in MPI consist of the ID of the sending process, the ID of the intended receiver, and an integer tag. A receive statement includes parameters indicating a source and tag, either or both of which may be wild cards. The result, then, of executing a receive statement at process i is the delivery of a message with destination i whose source and tag match those in the receive statement.

While straightforward, identifying messages with source, destination, and tag may not be adequate in complex applications, particularly those that include libraries or other functions reused from other programs. Often, the application programmer doesn't know any of the details about this borrowed code, and if the library includes calls to MPI, the possibility exists that messages in the application and the library might accidentally share tags, destinations IDs, and source IDs. This could lead to errors when a library message is delivered to application code, or vice versa. One way to deal with this problem is for library writers to specify reserved dial users must avoid in their code. This approach has proved cumbersome, however, and can lead to error because it requires programmers to carefully read and follow the instructions in the documentation.

The elegant MPI's solution to this problem is based on the notion of communication contexts. Both send and receive belong to a communication context, and only those communication events that share a communication context will match. Hence, even if messages share a source, a destination, and a tag, they will not be confused with each other as long as they have different contexts. Communication contexts are dynamically created and guaranteed to be unique.

In MPI the process group and communication context are combined into a single object called a communicator. At program startup, the runtime system creates a common communicator called `MPI_COMM_WORLD`.

MPI is the most commonly used API for parallel programming. MPI's low-level constructs are closely aligned to the MIMD model of parallel computers. This allows MPI programmers to precisely control how the parallel computation unfolds and write highly efficient programs. Perhaps even more important, this lets programmers write portable parallel programs that run well on shared-memory machines, massively parallel supercomputers, clusters, and even over a grid. MPI required much more coding to implement parallel version of the N-Body algorithm. Nevertheless, the changes in the `nbody` function itself are small:

```
#include <stdio.h>
#include <math.h>
#define eps 1.0e-12
int nbody(int n, int first, int last, double dt,
   double m[], double x[], double y[], double z[],
   double vx[], double vy[], double vz[],
   double xnew[], double ynew[], double znew[])
{
   double ax, ay, az, dx, dy, dz, d2, f, invr, invr3;
   int i, j, num_impacts, impact;
   num_impacts=0;
   for(i=first; i<=last; i++) { /* For selected particles */
     ax=ay=az=0.0; impact=0;
     for(j=0; j<n; j++) { /*Loop over all particles "j"*/
       if(j==i) continue;
       dx=x[j]-x[i]; dy=y[j]-y[i]; dz=z[j]-z[i];
       d2 = dx*dx+dy*dy+dz*dz;
       invr = 1.0/sqrt(d2+eps);
       invr3 = invr*invr*invr;
       f = m[j]*invr3;
       ax += f*dx; /* accumulate the acceleration from */
       ay += f*dy; /* gravitational attraction */
       az += f*dz;
       if(d2<=eps) impact=1; /*close particles*/
     }
```

```
      num_impacts+=impact; /*number of impacts*/
      /* update position of particle "i": */
      xnew[i] = x[i] + dt*vx[i] + 0.5*dt*dt*ax;
      ynew[i] = y[i] + dt*vy[i] + 0.5*dt*dt*ay;
      znew[i] = z[i] + dt*vz[i] + 0.5*dt*dt*az;
      /* update velocity of particle "i": */
      vx[i] += dt*ax; vy[i] += dt*ay; vz[i] += dt*az;
   }
   return num_impacts;
}
```

MPI constructions and functions are mostly used in `main()`.

```
#include <stdio.h>
#include <mpi.h> /* MPI functions, structures & constants */
#define BUF_LEN 10000; /* for memory allocation */
#define NUM_STEPS 100; /* number of simulation steps */
#define DT 0.0001; /* time step of the simulation */
/* external functions */
int nbody(int n, int first, int last, double dt,
   double m[], double x[], double y[], double z[],
   double vx[], double vy[], double vz[],
   double xnew[], double ynew[], double znew[]);
int init_data(int buf_len, double * buf); /* it's easy */
/* the program entry point */
int main(int argc, char* argv[]) {
 int thread, num_threads, first, last;
 double *buf, *m, *x, *y, *z, *vx, *vy, *vz;
 double *xnew, *ynew, *znew, *tmp;
 int n, buf_len, n_impacts, s, k;
 /*necessary MPI initialisation*/
 if(MPI_Init(&argc, &argv) != MPI_SUCCESS) {
   return 1; /* exit on MPI initialization error */
 }
 /* get the number of all running EUs */
 MPI_Comm_size(MPI_COMM_WORLD, & num_threads);
 /* get the ID number of this EU */
 MPI_Comm_rank(MPI_COMM_WORLD, & thread);
```

```
 /* allocate buffer */
 buf=calloc(BUF_LEN,sizeof(double));
 if(buf==NULL) { /* exit on memory error */
   MPI_Abort();/* interruption of the MPI program */
   return 2;
 }
 if(thread==0) { /* input or generate particles */
   n = init_data(BUF_LEN,buf); /* number of particles */
 }
/* distribute the number of particles from 0 thread */
MPI_Bcast(&n,1,MPI_INT,0,MPI_COMM_WORLD);
/* prepare buffer pointers */
m=buf; x=m+n; y=x+n; z=y+n; vx=z+n; vy=vx+n; vz=vy+n;
xnew=vz+n; ynew=xnew+n; znew=ynew+n;
 buf_len=n*10; /* number of logical buffers = 10 */
//{{main part
 for(s=0; s<NUM_STEPS; s++) { /* main loop */
  /* distribute the input data from 0 thread */
MPI_Bcast(tmp,buf_len,MPI_DOUBLE,0,MPI_COMM_WORLD);
  /* calculate particle diapason for the EU */
  first=n*thread/num_threads;
  last=(n+1)*thread/num_threads-1;
  /* run simulation */
   k=nbody(n, first, last, DT, m, x, y, z,
             vx, vy, vz, xnew, ynew, znew);
  /* synchronization */
  MPI_Barrier(MPI_COMM_WORLD);
  /* exchange of coordinate buffers */
  tmp=x; x=xnew; xnew=tmp;
  tmp=y; y=ynew; ynew=tmp;
  tmp=z; z=znew; znew=tmp;
}
MPI_Reduce(& k, & n_impacts, 1, MPI_DOUBLE,
MPI_SUM, 0, MPI_COMM_WORLD); /* sum from all trace */
//}}main part
 if(thread==0) { /* print result */
   printf("%d of %d particles stick together with other"
   "due to %d impacts\n",n_impacts,n,n_impacts/2);
 }
 free(tmp);
```

```
 MPI_Finalize(); /* correct finish of a MPI program */
 return 0; /* exit */
}
```

All MPI programs include a few basic elements. Consider the above program. We will explore the elements of MPI required to turn this into a parallel program where multiple processes execute the function. First, the function prototypes for MPI need to be defined. This is done with the MPI include file:

```
#include <mpi.h>
```

Next, the MPI environment must be initialized. As part of the initialization, the communicator shared by all the processes is created. As mentioned earlier, this is called `MPI_COMM_WORLD`:

```
MPI_Init(&argc, &argv);
```

The command-line arguments are passed into `MPI_Init` so the MPI environment can influence the behavior of the program by adding its own command-line arguments (transparently to the programmer). This function returns an integer status flag used to indicate success or failure of the function call. With very few exceptions, all MPI functions return this flag. Possible values of this flag are described in the MPI include file, `mpi.h`.

Although not required, almost every MPI program uses the number of processes in the group and the rank of each process in the group to guide the computation. This information is found through calls to the functions `MPI_Comm_size` and `MPI_Comm_rank`.

When the MPI program finishes running, the environment needs to be cleanly shut down. This is done with this function:

```
MPI.Finalize();
```

If at any point within a program a fatal condition is detected, it might be necessary to shut down the program. If this is not done carefully, processes can

be left on some of the nodes. These processes, called orphan processes, can in principle stay around 'forever' waiting for interaction with other processes within the group. The following function tells the MPI runtime environment to make a best-effort attempt to shut down all the processes in the MPI program:

```
MPI_Abort();
```

Earlier discussed point-to-point communication is not, however, necessary for any MPI program. I our example we didn't use pairwise message passing at all. Instead according to SPMD model we have used a set of operations in which all the processes in the group work together to carry out a complex communication. These collective communication operations are extremely important in MPI programming. In fact, many MPI programs consist primarily of collective operations.

The most commonly used collective operations include the following:

- `MPI_Barrier(MPI_COMM group)`. A barrier defines a synchronization point at which all processes must arrive before any of them are allowed to proceed. For MPI, this means that every process using the indicated communicator must call the barrier function before any of them proceed. The only parameter of this function is a group communicator,
- `MPI_Bcast(void *buff, int count, MPI_TYPE type, int source, MPI_COMM group)`. A broadcast sends a message from one process of rank `source` to all the processes in a group. Data are copied so from `buff` of the sending process to `buff` of each other process of the group. The `buff` points to array of `count` items of a specified `type`. The most commonly used types are `MPI_DOUBLE, MPI_INT, MPI_LONG`, and `MPI_FLOAT`,
- `MPI_Reduce(void *inbuff, void *outbuff, int count, MPI_TYPE type, int destination, MPI_COMM group)`. A reduction operation takes a set of values (in the buffer pointed to by `inbuff`) spread out around a process group and combines them using the indicated binary operation. To be meaningful, the operation in question must be associative. The most common examples for the binary function are summation and finding the maximum or minimum of a set of values. Notice that the final reduced value (in the buffer pointed to by

`outbuff`) is only available in the indicated `destination` process. If the value is needed by all processes, there is a variant of this routine called `MPI_All_reduce()`.

The same program can be paralleled using point-to-point MPI communications. To do this let's rewrite the main part:

```
//{{main part
MPI_Status status;
if(thread==0) { /* the master thread */
  for(s=0; s<NUM\_STEPS; s++) { /* main loop */
    /* distribute the input data from 0 thread */
    MPI_Bcast(tmp,buf_len,MPI_DOUBLE,0,MPI_COMM_WORLD);
    /* calculate particle diapason for the EU */
    for(k=0; k<min(n, num_threads); k++) {
      first=k*thread/num_threads;
      MPI_Send(&first,1,MPI_INT,k,77,MPI_COMM_WORLD);
    }
    for(; k<n; k++) {
      MPI_Recv(&last,1,MPI_INT,MPI_ANY_SOURCE,88,MPI_COMM_WORLD,
      &status);
      /* from any thread */
      first=k*thread/num_threads;
      MPI_Send(&first,1,MPI_INT,status.MPI_SOURCE,77,
      MPI_COMM_WORLD);
      /* to the thread */
      n_impacts+=last;
    }
    for(k=0; k<min(n, num_threads); k++) {
      MPI_Recv(&last,1,MPI_INT,MPI_ANY_SOURCE,88,MPI_COMM_WORLD,
      &status);
      /* from any thread */
      n_impacts+=last;
    }
    for(k=0; k<n; k++) {
      first = s+1<NUM_STEPS? -1: -2;
      MPI_Send(&first,1,MPI_INT,k,77,MPI_COMM_WORLD);
    }
```

```
    /* exchange of coordinate buffers */
    tmp=x; x=xnew; xnew=tmp;
    tmp=y; y=ynew; ynew=tmp;
    tmp=z; z=znew; znew=tmp;
  }
}
else { /* a worker thread */
  /* distribute the input data from 0 thread */
  MPI_Bcast(tmp,buf_len,MPI_DOUBLE,0,MPI_COMM_WORLD);
  for(;;) {
    MPI_Recv(&first,1,MPI_INT,0,77,MPI_COMM_WORLD,&status);
    if(first==-2) break; /* finish of computations */
    if(first==-1) { /* a new step */
      /* distribute the input data from 0 thread */
      MPI_Bcast(tmp,buf_len,MPI_DOUBLE,0,MPI_COMM_WORLD);
    }
    else { /* run simulation */
      first=last;
      n_impacts=nbody(n, first, last, DT, m, x, y, z,
             vx, vy, vz, xnew, ynew, znew);
      MPI_Send(&n_impacts,1,MPI_INT,0,88,MPI_COMM_WORLD);
    }
  }
//}}main part
```

The above code looks both more complex and less efficient then the previous one. Nevertheless, there are multiple tasks, especially in heterogenic clusters, which benefit from dynamic load balancing. The code implements 'master/worker' strategy using the most commonly used message-passing MPI functions `MPI_Send/MPI_Receive`:

- `MPI_Send (void *buff, int count, MPI_TYPE type, int dest, int tag, MPI_COMM group)` sends `buff` of `count` items of type `MPI_TYPE` to the process of rank `dest` from the communication group `group` and immediately proceed. The message is identified by a user selected integer number `tag`.
- `MPI_Recv (void *buff, int count, MPI_TYPE type, int source, int tag, MPI_COMM group, MPI_Status * stat)` waits for and receives a message identified by the number `tag`. The message identifier

`tag`, the number of receiving items `count`, and the item type `type`, as well as the communicator `group` must fit to the same parameters of the sending `MPI_Send`. Instead of specification of the sending process rank one may set `source` to the predefined value `MPI_ANY_SOURCE`. In this case the rank of sending process can be obtained later as the value of `MPI_SOURCE` member of the `stat` structure.

They are the blocking send/receive routines to send a message from one process to another. 'Blocking' means that the receive function waits for the sending data. Most MPI programmers use only these standard point-to-point message-passing functions. However, there are about 20 other functions in MPI for point-to-point communication. This large set of message-passing functions provides the controls needed to optimize the use of communication buffers and specify how communication and computation overlap. Probably the most important of these advanced message-passing routines are the nonblocking or asynchronous communication functions.

Learning MPI can be intimidating. It is huge, with more than 125 different functions even in MPI 1.1. The large size of MPI does make it complex, but most programmers avoid this complexity and use only a small subset of MPI. Many parallel programs can be written with just six functions: MPI_Init, MPI_Comm_Size, MPI_Comm_Rank, MPI_Send, MPI_Recv, and MPI_Finalize.

3.4.3 OpenCL for GPU, Cell, SMP CPU and Heterogeneous Computing

Stream computing language such as OpenCL, CUDA or Brook+ may be called an 'assembly code' of parallel programming. Efficient GPGPU computing requires accounting of much more implementation details than application programmers have got accustomed.

OpenCL [OCL] execution model is very similar to other stream language's one. It is based on queued SPMD and multilevel distributed memory. The main EU called 'kernel' uses its 1D, 2D or 3D integer 'coordinate' in index-space of elements called work-items to select a data part to process or to modify its behavior.

Although there is no requirement that implementations of OpenCL rely on

a multi-threaded execution model, it is useful to make this connection conceptually since many programmers are familiar with threads, and kernels must be written to support such an execution model, i.e., they must be thread-safe. Using the language of threads, work-items can be thought of as enumerated threads, where the index-space defines the enumeration. As an example, if one needed to process each pixel in a two-dimensional (2D) image, the OpenCL index-space would map to the 2-D array of pixels and the programmer would write a kernel that would be executed, possible in parallel, for each pixel. 1D kernel index directly corresponds to rank of MPI. Typical for GPGPU 2D index identifies PU by its row and column numbers. 3D coordinates help to deal with 3D data grids. OpenCL program contains of either one or multiple kernels.

OpenCL is comprised of two parts designed to facilitate the programming of heterogeneous platforms with co-processors. In order to program the actual co-processor device OpenCL provides a C-like language for writing computational kernels, which implement the core computational algorithms. The execution of the kernel code is controlled by the host platform through a runtime API that allow the programmer to orchestrate the execution of the kernels and provides all supporting facilities necessary to do this efficiently in an inherently asynchronous computing environment.

In contrast to other GPGPU programming tools OpenCL can run kernels on a collection of devices of different nature (CPUs, GPUs, Cell, etc.). The host code required to orchestrate the execution of OpenCL kernels provides for the ability to control the operations on the host plus multiple co-processor devices.

One of the most significant issues for such an architecture is memory management and OpenCL provides a memory management model (Fig. 3.7) that allows relaxed memory consistency. This places the burden on the programmer to ensure that the memory used in operations that are generally concurrent and asynchronous remains consistent. OpenCL provides a platform layer designed to enable support for a range of devices, which the programmer may query for and, if present, determine their respective capabilities and utilize accordingly.

Many elements of the OpenCL programming model can be thought of as operating system functionality moved into user-space since the API provides for careful control of (enumerated) threads within an inherently asynchronous concurrent environment. This is very similar to the basic operation of a

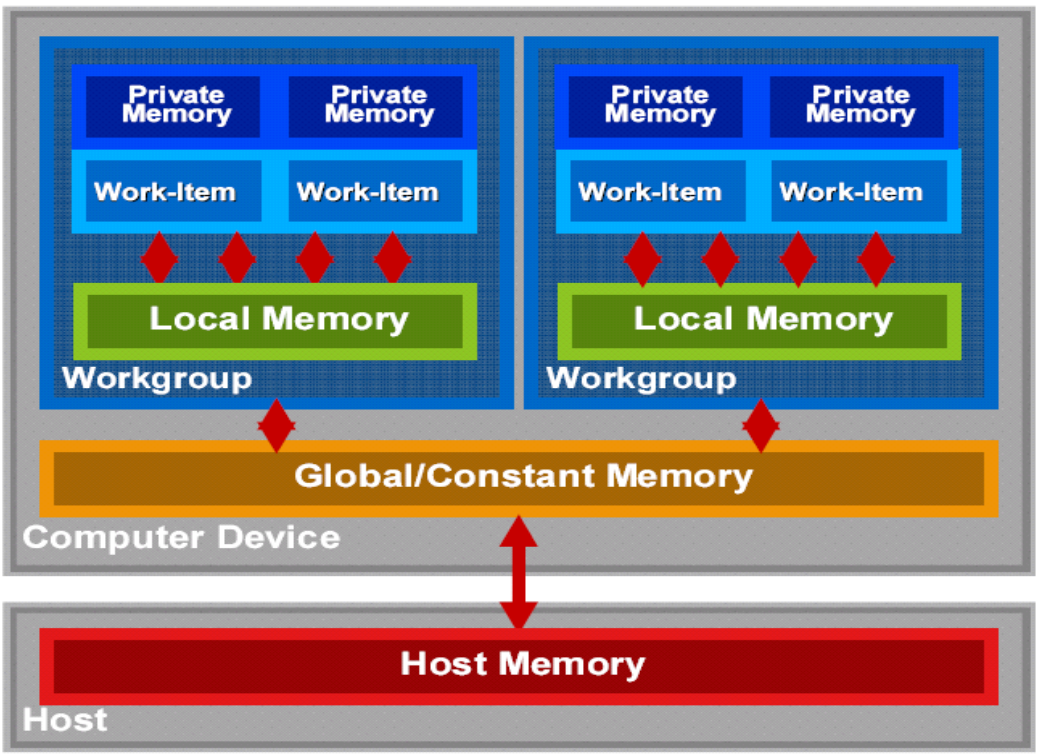


Figure 3.7: Multilevel memory model of OpenCL

generic UNIX kernel. As a result, many of the concepts familiar to the management of threads within an operating system apply conceptually to the OpenCL programming model, e.g., memory consistency, locking and synchronization, work queues, event lists, etc. Being designed to run kernels on various separated devices OpenCL provides restricted synchronization abilities (Fig. 3.8).

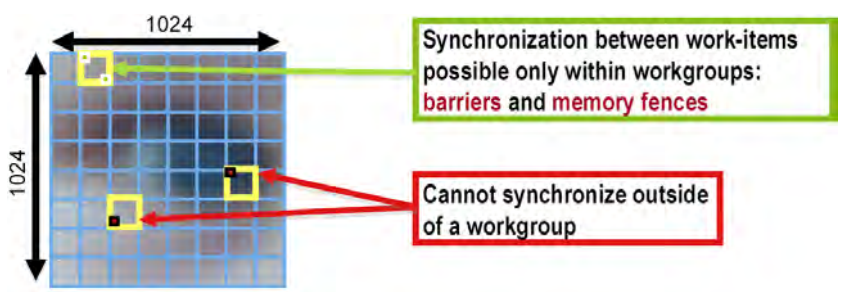


Figure 3.8: Synchronization in OpenCL

As in the MPI example let's start from the `nbody` function implementation, that means from the kernel source file `'nbody\_kern.cl'`. The kernel code is very different from what we had:

```
__kernel void nbody (
  float dt1,
  __global float4* pos_old,
  __global float4* pos_new,
  __global float4* vel,
  __local float4* pblock /* is used to optimize speed */ )
  __global int *n_impact )
{
  const float eps = 1.0e-12;
  const float4 dt = (float4)(dt1,dt1,dt1,0.0f);
  int gti = get_global_id; /* global work item rank*/
  int n = get_global_size; /* global size is equal
                              to the number of particles */
  int ti = get_local_id; /* the work item \# in workgroup*/
  int nt = get_local_size; /* a workgroup size */
  int nb = n/nt; /* the number of workgroups */
  float4 p = pos_old[gti]; /* current particle's x,y,z,m */
  float4 v = vel[gti]; /* current particle's vx,vy,vz */
  float4 a = (float4)(0.0f,0.0f,0.0f,0.0f);
  int impact = 0;
  for(int jb=0; jb < nb; jb++) { /* For each block ... */
    pblock[ti] = pos_old[jb*nt+ti]; /* Cache ONE particle */
    barrier(CLK_LOCAL_MEM_FENCE); /* Wait for others */
    for(int j=0; j<nt; j++) { /* For ALL cached particles */
      float4 p2 = pblock[j]; /* Read a cached particle */
      float4 d = p2 - p;
      float d2 = d.x*d.x + d.y*d.y + d.z*d.z;
      if(d2<=eps) impact=1;
      float invr = rsqrt(d2 + eps);
      float f = p2.w*invr*invr*invr; /* p2.w contains m */
      a += f*d; /* Accumulate acceleration */
    }
    barrier(CLK_LOCAL_MEM_FENCE); /* Wait for others */
  }
```

```
  p += dt*v + 0.5*f*dt*dt*a;
  v += dt*a;
  pos_new[gti] = p;
  vel[gti] = v;
  n_impact[gti] = impact;
}
```

The qualifier `__kernel` denote the fact this function is kernel. An OpenCL program can include both kernels and ordinary functions (to be called from kernels). Two other qualifiers `__global` and `__local` correspond to different levels of shared memory in the OpenCL memory model (Fig. 3.7). All variables without those qualifies are private ones of a work item (EU).

Of the many OpenCL vector data types, `float4` is probably the most common. It was used since times when GPUs were just graphics accelerators. In large part due to this history, the four components of a `float4` can be referenced using the suffix `.x, .y, .z`, and `.w`, respectively. (OpenCL also provides a more generic notation `.s0, .s1, .s2, .s3` that can be generalized to larger vectors).

The use of vector data types in OpenCL is not a mere convenience, but is important for performance since an architecture may have a natural vector width within its processing cores. As an example, both the AMD/ATI and NVIDIA GPU architectures have had a natural type of `float4` and efficient implementations on these architectures must exploit this through small SIMD or SSE-like constructions. OpenCL provides extensive support for vector data types to enable the programmer to exploit vector operations on the underlying architecture of different co-processor devices.

For our N-Body program we pack the particle coordinates and particle mass into a four-vector as {x, y, z, m}, and the three velocity components are packed into a four-vector with the last component ignored: {vx, vy, vz, -}.

The technique of packing the particle mass into the fourth component of the 'position vector' may create a degree of syntactic confusion since one must remember that the mass has been put there in defiance of the obvious interpretation of the variable name pos. This is unavoidable without performance loose.

This time kernel is designed for 1D index space. So `get_global_id(0)` corresponds to MPI rank, and `get_global_size(0)` corresponds to MPI

size. The local (workgroup) rank from `get_local_id(0)` and size from `get_local_size(0)` are used for memory access optimization.

Initially all the particle data are located in the device global memory. The kernel first copies its 'own' particle data in private variables. Then it implements computing by blocks of size equal to the workgroup size.

First to fill the block buffer `pblock` each kernel copies just one value from the global `pos_old` to the local shared array `pblock`. After passing the first barrier all the block data are put in the local memory by parallel efforts of the workgroup members to be used in farther computing.

Within the internal `for` loop each work item computes accumulated impulse of the cached particles on its 'own' one. The code is very similar to the ordinary C code with two exceptions: intensive use of the `float4` vector operations and application of the OpenCL function `rsqrt()` meaning `1/sqrt()`. OpenCL contains a big number of math functions for both vector and scalar values.

OpenCL is different from CUDA and Brook+ in that OpenCL does not provide a standalone compiler for creating device ready binary code. The OpenCL interface provides methods for compiling kernels given a string containing the kernel code at runtime (`clCreateProgramWithSource`). Once a kernel is compiled it can be launched on the device. The host-side support of OpenCL kernels is extremely complex because of the universal and multilayer approach [31]. To simplify it David Richie from Brown Deer Technology [4] has developed the Standard Compute Layer Library (*libstdcl*) which provides a simple host-side interface to the OpenCL API for parallel programming of heterogeneous platforms with co-processor devices. It is designed to support the most typical use-cases in a style inspired by familiar and traditional UNIX APIs for C programming. The libstdcl library implemented by Brown Deer Technology is freely available and distributed under the GNU Lesser General Public License (LGPL). The next sample codes are based on libstdcl.

The host-side of OpenCL presents a hierarchy of constructs that must be set-up and managed in order to execute computational kernels. In practical terms, setting up OpenCL for running a relatively simple program requires the following steps:

1. create a context `cl_context` containing the target devices. As a common example, the programmer can create a context to contain all GPU

devices. The devices belonging to a given context are identified by a device list `cl_device_id`,

2. create command queues `cl_command_queue` for each device in the context,

3. create memory objects `cl_mem` needed to share memory with the co-processor devices,

4. load and link one or more computational kernels `cl_kernel`. In the simplest approach one can build a program `cl_program` based on kernels in a .cl file and the compile and link all kernels for all devices in a context. Note that just-in-time (JIT) compilation is an important aspect of the OpenCL.

Executing the computational kernels then generally involves these additional steps:

1. define an index-space NDRange over which the kernel is to be executed,

2. set the arguments of the kernel – in a sense you must explicitly 'push' the arguments onto an imaginary stack before executing a kernel. The analogy to a stack breaks down somewhat since setting the arguments need not be in order,

3. ensure that the memory to be used by the kernel is consistent, i.e., make sure all data is where it is needed. This can be accomplished with memory operations that are enqueued on one of the command queues for a particular device,

4. enqueue a kernel for execution on one of the command queues for a particular devices,

5. monitor the associated event `cl_event` corresponding to the enqueued operation. This also applies to the memory operations referred to in step 3,

6. read back any results needed on the host. This is not too different from step 3.

The main host-side C99 code based on *libstdcl* follows:

```
#include <stdio.h>
#include <stdcl.h> /*functions, types, constants of libstdcl*/
/* a simple data initialization not detailed here: */
void nbody_init(int n,cl_float4* pos,cl_float4* vel);
/* the program entry point: */
int main(int argc, char** argv) {
  int step, i, impact;
  int n = 8192; /* MUST be a power of two for simplicity */
  float dt = 0.0001;
  int nstep = 100; /* the number of simulation steps */
  int nthread = 64; /* chosen for ATI Radeon HD 5870 */
  if (!stdgpu) exit(-1); /* No GPGPU device is available */
  cl_float4* pos1 = (cl_float4*)clmalloc(stdgpu,
  n*sizeof(cl_float4),0); /* allocate memory for GPU */
  cl_float4* pos2 = (cl_float4*)clmalloc(stdgpu,
  n*sizeof(cl_float4),0); /* allocate memory for GPU */
  cl_float4* vel = (cl_float4*)clmalloc(stdgpu,
  n*sizeof(cl_float4),0); /* allocate memory for GPU */
  cl_int* nimpact = (cl_int*)clmalloc(stdgpu,
  n*sizeof(cl_int),0); /* allocate memory for GPU */
  nbody_init(n,pos1,vel); /* fill the input data arrays */
  /* open and compile the OpenCL kernel file: */
  void* h = clopen(stdgpu,"nbody_kern.cl",CLLD_NOW);
  cl_kernel krn = clsym(stdgpu,h,"nbody",CLLD\_NOW);
  /* define 1D index space: */
  clndrange_t ndr = clndrange_init1d(0,n,nthread);
  /* assign the kernel function arguments but pos1/pos2: */
  clarg_set(krn,0,dt);
  clarg_set_global(krn,3,vel);
  clarg_set_local(krn,4,nthread*sizeof(cl_float4));
  clarg_set_global(krn,5,nimpact);
  /* assign the kernel function arguments but pos1/pos2: */
  for(step=0; step$<$nstep; step++) {
    /* assign the kernel function arguments pos1/pos2: */
    clarg_set_global(krn, 1, (step&1)? pos2: pos1);
    clarg_set_global(krn, 2, (step&1)? pos1: pos2);
    if(step<nstep)
    /* run the kernel on GPGPU */
    clfork(stdgpu},0,krn,&ndr,CL_EVENT_NOWAIT);
  }
```

```
  /* synchronization barrier: */
  clwait(stdgpu,0,CL_KERNEL_EVENT|CL_EVENT_RELEASE);
  /* synchronization of GPU/RAM content of arrays: */
  if(nstep & 1) {
    clmsync}(stdgpu,0,pos2,CL_EVENT_WAIT|CL_EVENT_RELEASE);
    memcpy(pos1, pos2, n*sizeof(cl_float4)); }
  else clmsync(stdgpu,0,pos1,CL_EVENT_WAIT|CL_EVENT_RELEASE);
    clmsync}(stdgpu,0,nimpact,CL_EVENT_WAIT|CL_EVENT_RELEASE);
  for(impact=i=0; i<n; i++) impact+=nimpact[i];
  printf("%d of %d particles stick together with other"
  "due to %d impacts\n", impact, n, impact/2);
  clclose}(stdgpu,h); /* release of the compiled kernel */
  /* memory release: */
  clfree(pos1); clfree(pos2); clfree(vel); clfree(nimpact);
  return 0;
}
```

Several **default contexts** provided by *libstdcl* similar to the default I/O streams of *stdio* are suitable for the typical OpenCL applications. They are:

- `stddev` – all devices for a given platform supported by OpenCL,
- `stdcpu` – all multicore CPU processors for a given platform supported by OpenCL,
- `stdgpu` – all GPU processors for a given platform supported by OpenCL,
- `stdrpu` – all reconfigurable processors for a given platform supported by OpenCL.

The set of functions `clopen(), clsym(), clclose()` provide a convenient interface capable of dynamically loading CL programs embedded within the executable as well as from an external file. The function `clopen()` loads the CL source or binary program file named by the NULL-terminated string filename and returns an opaque handle that may be used as a reference in subsequent calls. If filename is a NULL pointer then a handle for the main program executable is returned. The funcion `clsym()` takes a handle to a CL source or binary program and a NULL-terminated kernel function name and returns the associated CL kernel. A CL context pointer and device number must be specified to identify the appropriate CL kernel to return. If

handle is NULL then all CL pro- grams loaded into the specified CL context are searched. The function `clclose()` decrements the reference count on the associated handle. If the reference count drops to zero then the CL program is unloaded. It returns the reference count on success and -1 on error. The supplementary function `clerror()` returns a human readable string describing the most recent error that has occurred as a result of a call to any of the functions.

It is worth pointing out that the overall interface of `clopen()` and `clsym()` will likely be convenient to programmers already familiar with the functions `dlopen()` and `dlsym()` for conventional shared libraries. This is not an accident since the interface for loading OpenCL code was modeled after these calls. It is also worth noting that *libstdcl* facilitates the embedding of OpenCL code directly within the executable, which would make the specification of the separate .cl file unnecessary. Here we use the 'separate file model' to emphasize the just-in-time (JIT) nature of OpenCL programs, where the machine code generation must be deferred until runtime, when actual coprocessor device present is identified. In this example, the backend compiler will perform a JIT compilation and target our GPU at runtime.

Memory management functions of *libstdcl* implement allocating and managing memory that may be shared between the host and CL coprocessor devices. Memory may be allocated with `clmalloc()` and used transparently as the global memory for kernel execution on a CL device. The programmer uses a single pointer representing the allocated memory which may be reattached to various CL contexts using `clmattach()` and `clmdetach()`. Memory consistency can be maintained using the `clmsync()` function which synchronizes memory between host and CL coprocessor device.

Executing a kernel on a particular CL coprocessor device is supported using `clfork()` which allows both blocking and non-blocking execution behavior. Before executing a kernel one must assign values to the kernel function arguments. This can be unfamiliar to ordinary C programmers since it represents a syntactically verbose call model of the OpenCL API. The functions `clarg_set(), clarg_set_global()` and `clarg_set_local()` are used to set the argument identified by its sequential number to the kernel referred by the first parameter of each the function. Among them `clarg_set()` is used for setting arguments of intrinsic type such as `cl_int, cl_float` or `cl_float4`, etc. The next one `clarg_set_global()` is used for setting arguments of pointers to global memory allocated using a call to `clmalloc()` and attached to the CL context of the target kernel.

The last one `clarg_set_local()` associates arguments with pointers to the device local memory of specified size in bytes.

Management of asynchronous multi-device operations is implemented through events. The function `clwait()` can be used to block on selected events within one of several per device event lists managed transparently. The function behavior is modified using combinations of the next flags:

- `CL_KERNEL_EVENT` blocks on events in the ordered kernel event list,
- `CL_MEM_EVENT` blocks on events in the ordered memory event list,
- `CL_EVENT_RELEASE` forces `clwait()` to release all events on upon completion for all events on which it blocks.

  If this flag is not used the programmer is responsible for releasing the returned event using `clReleaseEvent()`.

3.5 Computational Grid

An important part of supercomputer tasks represents solution of very big scientific problems. For economic reasons research labs usually do not have own computational resources of necessary for those tasks performance. Moreover, even the most power world supercomputers sometimes can't solve such a task for reasonable time and cost. To solve big problems multiple cooperating labs may join their computational resources. Grid computing is a technology primary devoted to solution of computation intensive problems by virtual organizations of independent trusted partners.

From the hardware viewpoint a grid is the collaboration of computers from multiple administrative domains for a common goal. Grid computing is a special type of distributed parallel computing that relies on complete computers or computer clusters connected to a network (private, public or the Internet) by conventional network interfaces, such as Ethernet. This is in contrast to the traditional notion of a supercomputer, which has many processors interconnected by a high-speed local network or a computer bus.

From the software development viewpoint a grid is not another programming tool, API or infrastructure. *The application programs are not supposed to be rewritten or recompiled for grid.* Instead grid proposes middleware for

automated distribution of input data and computational programs over the Internet, for the job execution on fitted computers in corresponded system environments, and for collection of the computation results in the specified place. Sequential programs can be used in grid calculations as well as GPGPU and cluster ones. Grid use means rather specification and administration efforts then software development.

The real and specific problems that underlay the grid concept are *coordinated resource sharing* and *problem solving in dynamic, heterogeneous, multi-institutional virtual organizations*. The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource brokerage strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a Virtual Organization (VO). Some examples of VOs are: the application service providers, storage service providers, cycle providers, and consultants engaged by a car manufacturer to perform scenario evaluation during planning for a new factory; members of an industrial consortium bidding on a new aircraft; a crisis management team and the databases and simulation systems that they use to plan a response to an emergency situation; and members of a large, international long-term high energy physics collaboration. Each of these examples represents an approach to computing and problem solving based on collaboration in computation- and data-rich environments.

Architecture. The users interact with the Grid Resource Broker to solve problems, which in turn performs resource discovery, scheduling, and the processing of application jobs on the distributed grid resources. The grid resource management is internally based on information from Catalogs of *Grid Monitoring Architecture* (e.g., Relational GMA, R-GMA). Another key components of a grid are Computing Elements which take control over calculations on subordinated Worker Nodes through *Local Resource Management System* and are themselves managed as resources by a Workload Management System via *Grid Gates*. Storage Elements take control over data storing through the *Local Transfer Service* and are managed themselves by a Storage Resource Manager. Logging and Bookkeeping System provide information about both the grid system and job state.

To access the grid services an end-user have to become a member of a VO. Then he or she interacts with grid applications and/or web portals which are supported by the VO (Fig. 3.9).

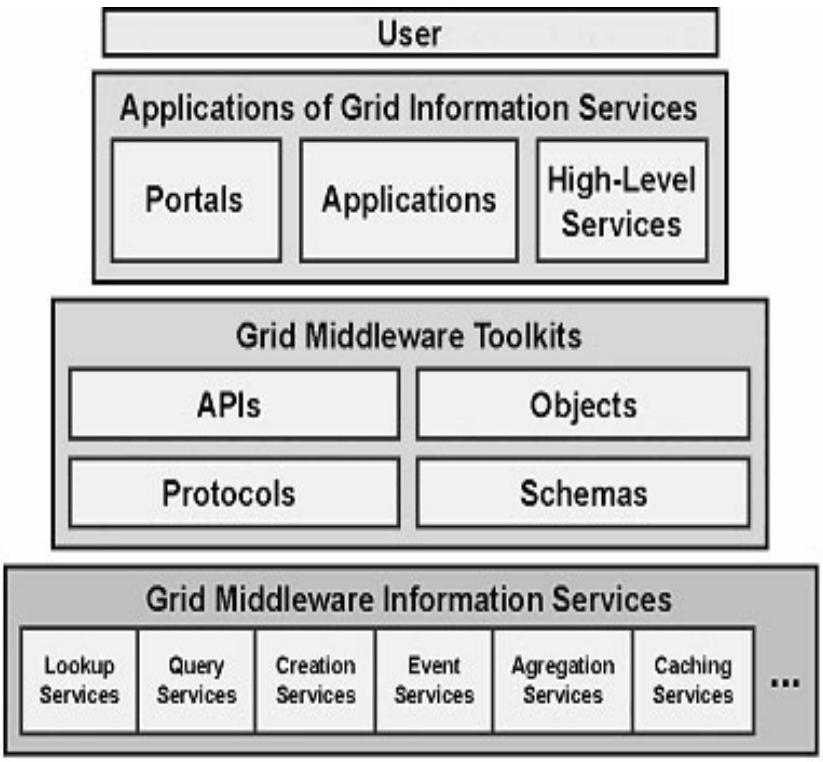


Figure 3.9: General grid interface architecture

The grid applications/portals provide the user with tools to script an execution plan (an abstract job) containing a sequence of program runs in specified environments together with related data transition operations, to control the execution state, to analyze resource availability, and so on. These tools ether are parts of a **Grid Middleware** or were implemented using the grid middleware API.

The grid middleware implementations are big complex distributed software systems. ARC (Advanced Resource Connector) and gLite are two of the major production-ready grid middleware solutions being used by hundreds researchers every day. The ARC middleware was born from the Nordic Council of Ministers' supported project 'Nordic Testbed for Wide Area Computing and Data Handling' which ran in 2001-2003 and gave rise to the Nor-

duGrid collaboration. gLite's history is more complex and indirect. The gLite middleware is a deployment ready software stack built from components of other middlewares. The most important are those from European Data Grid (EDG) project, followed by elements of AliEn and the middleware developed by the EGEE project, gLite itself. In addition, gLite in its various versions often offers several different solutions for the same middleware component, simplifying transition from one implementation to another. The third by distribution grid middleware is UNICORE (UNiform Interface to COmputer REsources) funded by the German Ministry of Education and Research.

The most basic building block of all the above middlewares is the Globus Toolkit v.2.6 (GT2). Globus by itself is not a deployment ready grid solution, rather it is a toolkit for building other grid middleware. GT2 defines a security scheme (GSI – Globus Security Infrastructure) for user authentication, authorization and delegation of rights on the base of Virtual Organisation Membership Service (VOMS). The scheme is based on X509 certificates and short lived proxy certificates inspired by the Kerberos 5 security framework. GT2 also defines protocols for file transfer (GSIFTP), job submission by secure Globus Resource Allocation Manager (GRAM) and an information system based on LDAP (MDS – Monitoring and Discovery Service). Finally, GT defines a resource specification language, RSL.

Current Globus Toolkit version 4 is deeply integrated with web service architecture. Despite its known advantages, it don't have enough popular 3rd party middleware implementations.

Even though all listed middlewares are based on the same technology, there are substantial architectural and implementational divergences. An ordinary user faces difficulties trying to cross the boundaries of the systems (see Tab. 3.1).

Interoperability. The ARC and gLite middleware projects shared a common background in the early stages of the EDG project to primarily enable the data acquisition and processing of the experiments on the Large Hadron Collider at CERN and, by achieving this goal, also enable the use of the middleware by other research communities. One of the LHC experiments, the ATLAS experiment, has conducted several production test runs of its application-specific software. One of these, ATLAS Data Challenge 2, used resources in US and Europe and as such had to use several middlewares. For this purpose a meta-scheduler, the ATLAS Production System, was cre-

Table 3.1: Protocols, languages and tool of different grid middlewares [8]

	Globus Toolkit 4	ARC	gLite 3	UNICORE6
Job Management	GRAM	GridManager	WMS (WM)	WF-engine
Local Services	Local Task Scheduler Adapter	LRMS (PBS, Condor)	CE	TSF, TSS, JMS
Job Specification	RSL	XRSL	JDL	AJO, JSDL
Data Management	GridFTP+RFT	SE, GridFTP	SE, GridFTP, FTS	SMS, GridFTP
Catalogs-Replicas	DRS (RLS)	SSE (RLS, RC, ...)	LFC, SRM	
Information Services	WebMDS, Index, Aggregat. Trigger	LIS, IS, RP (LDAP)	R-GMA, L&B, DGAS	WS-resources
Security	GSI, VOMS, WS-Security	GSI, VOMS	GSI, VOMS, WS-Security	SSL, X.509, UUDB
Web-Services	+	(NOX+)	+/-	+
API/SDK	+(e.g., CoG Kits)	ARCLib (C/C++), JARCLib(Java), DataMove	+(for all subsystems)	GPE, GridBeans

ated, enabling job-submission to the three grids: Grid3 (now OSG), gLite and ARC. It is the desire to streamline this work into actual interoperability between several grids that motivates the OSG LCG interoperability as well as this activity.

Functionality. Grids can be used for the variety of services [3]:

- ***Computational services***. These are concerned with providing secure services for executing application jobs on distributed computational resources individually or collectively. Resources brokers provide the services for collective use of distributed resources. A grid providing computational services is often called a Computational Grid. Some examples of computational grids are: NASA IPG, the World Wide Grid, and the NSF TeraGrid.

- ***Data services***. These are concerned with proving secure access to distributed datasets and their management. To provide a scalable storage and access to the data sets, they may be replicated, catalogued, and even different datasets stored in different locations to create an illusion of mass storage. The processing of datasets is carried out using computational grid services and such a combination is commonly called Data Grids. Sample applications that need such services for management, sharing, and processing of large datasets are high-energy physics and accessing distributed chemical databases for drug design.

- ***Application services***. These are concerned with application management and providing access to remote software and libraries trans-

parently. The emerging technologies such as web services are expected to play a leading role in defining application services. They build on computational and data services provided by the grid. An example system that can be used to develop such services is NetSolve.

- ***Information services***. These are concerned with the extraction and presentation of data with meaning by using the services of computational, data, and/or application services. The low-level details handled by this are the way that information is represented, stored, accessed, shared, and maintained. Given its key role in many scientific endeavors, the Web is the obvious point of departure for this level.

- ***Knowledge services***. These are concerned with the way that knowledge is acquired, used, retrieved, published, and maintained to assist users in achieving their particular goals and objectives. Knowledge is understood as information applied to achieve a goal, solve a problem, or execute a decision. An example of this is data mining for automatically building a new knowledge.

3.6 Parallel Program Efficiency: Performance and Scalability

Availability of big scale parallel systems has fueled interest in investigating the performance of parallel computers containing a large number of processors. While solving a problem in parallel, it is reasonable to expect a reduction in execution time that is commensurable with the amount of processing resources employed to solve the problem. The scalability of a parallel algorithm on a parallel architecture is a measure of its capacity to effectively utilize an increasing number of processors. Scalability analysis of a parallel algorithm-architecture combination can be used for a variety of purposes. It may be used to select the best algorithm-architecture combination for a problem under different constraints on the growth of the problem size and the number of processors. It may be used to predict the performance of a parallel algorithm and a parallel architecture for a large number of processors from the known performance on fewer processors. For a fixed problem size, it may be used to determine the optimal number of processors to be used and the maximum possible speedup that can be obtained. The scalability analysis can also predict the impact of changing hardware technology on the

performance and thus help design better parallel architectures for solving various problems.

The scalability analysis is usually based on several assumptions and terminology.

- **Parallel System** is the combination of a parallel architecture and a parallel algorithm implemented on it. The parallel computers assumed to be homogeneous in the scalability analysis; *i.e.,* all processors and communication channels are identical in speed.

- **Problem Size** W is a measure of the number of basic operations needed to solve the problem. There can be several different algorithms to solve the same problem. To keep the problem size unique for a given problem, we define it as the number of basic operations required by the fastest known sequential algorithm to solve the problem on a single processor. Problem size is a function of the size of the input. For example, for the problem of computing an N-point FFT, $W = O(N \log N)$.

- According to our definition, **the sequential time complexity** of the fastest known serial algorithm to solve a problem determines the size of the problem. If the time taken by an optimal (or the fastest known) sequential algorithm to solve a problem of size W on a single processor is T_1, then $T_1 \propto W$, or $T_1 = t_c W$, where t_c is a machine dependent constant.

- **Serial Fraction** σ: the ratio of the serial component of an algorithm to its execution time on one processor. The serial component of the algorithm is that part of the algorithm which cannot be parallelized and has to be executed on a single processor.

- **Parallel Execution Time** T_P: the time elapsed from the moment a parallel computation starts, to the moment the last processor finishes execution. For a given parallel system, T_P is normally a function of the problem size W and the number of processors p, and we will sometimes write it as $T_P(W, p)$.

- **Speedup** S: the ratio of the serial execution time of the fastest known serial algorithm T_1 to the parallel execution time of the chosen algorithm T_P. For a fixed size problem:

$$Speedup\,(p) = \frac{Performance\,(p)}{Performance\,(1)} = \frac{Time\,(1)}{Time\,(p)} \quad \text{or} \quad S_p = \frac{T_1}{T_p}$$

It is a well known fact that given a parallel architecture and a problem instance of a fixed size, the speedup of a parallel algorithm does not continue to increase with increasing number of processors. As early as in 1967, Gene Amdahl [2] made the observation that if s is the serial fraction in an algorithm, then its speedup is bounded, no matter how many processors are used. This statement, now popularly known as **Amdahl's Law,** has been used by Amdahl and others to argue against the usefulness of large scale parallel computers.

Amdahl's Law expressers an speedup of an ideally paralleled problem with a fixed serial fraction on an ideal parallel computer with neither synchronization nor communication overheads:

$$S_p = \frac{T_1}{T_p} = \frac{T_1}{\sigma T_1 + (1-\sigma)\,T_1/p} = \frac{1}{\sigma + (1-\sigma)\,/p} \Rightarrow S_\infty = \frac{1}{\sigma}.$$

The Amdahl's speedup dependency on the number of processors and the serial fraction is shown in Fig. 3.10.

The negative effect of Amdahl's Law is obvious even for small numbers: if 95% of a program is ideally paralleled by 20 PUs the speedup is:

$$S_p = \frac{1}{\sigma + (1-\sigma)\,/p} = \frac{1}{0.05 + 0.95/20} = 10.26,$$

two times worse than one may achieve.

Actually, in addition to the serial fraction, the speedup obtained by a parallel system depends upon a number of factors such as the degree of concurrency and overheads due to communication, synchronization, redundant work etc. For a fixed problem size, the speedup saturates either because the overheads grow with increasing number of processors or because the number of processors eventually exceeds the degree of concurrency inherent in the algorithm.

Nevertheless, Gustafson, Montry and Benner [15, 16] were the first to experimentally demonstrate that by scaling up the problem size one can obtain near-linear speedup on as many as 1024 processors. Gustafson et al. introduced a new metric called **scaled speedup** to evaluate the performance

on practically feasible architectures. This metric is defined as the speedup obtained when the problem size is increased linearly with the number of processors. If the scaled-speedup curve is good (e.g., close to linear w.r.t. the number of processors), then the parallel system is considered scalable. The formula is now known as Gustafson-Barsis' or Gustafson's Law:

$$Speedup\,(p) = \frac{Performance\,(p)}{Performance\,(1)} = \frac{Work\,(p)}{Work\,(1)}\,,$$

for a fixed time problem, hence scaled speedup:

$$S_p = \frac{W_p}{W_1} = \frac{\sigma W_1 + (1-\sigma)\,pW_1}{W_1} = \sigma + (1-\sigma)\,p\,.$$

Two generalized notions of scaled speedup were considered by Gustafson [16], Worley [41] and Sun & Ni [37]. They differ in the methods by which the problem size is scaled up with the number of processors. In one method, the size of the problem is increased to fill the available memory on the parallel computer. The assumption here is that aggregate memory of the system will increase with the number of processors. In the other method, the size of the problem grows with p subject to an upper-bound on execution time. Worley found that for a large class of scientific problems, the time-constrained

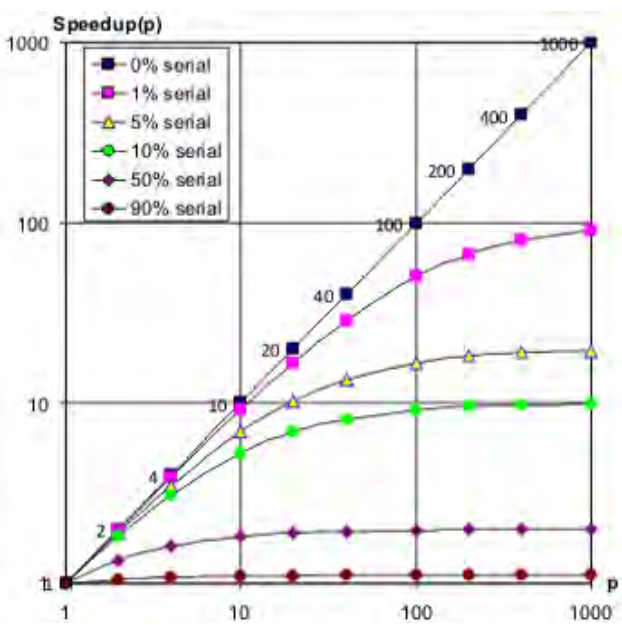


Figure 3.10: Amdahl's Law

speedup curve is very similar to the fixed-problem-size speedup curve. He found that for many common scientific problems, no more than 50 processors can be used effectively in current generation multicomputers if the parallel execution time was to be kept fixed. For some other problems, Worley found time-constrained speedup to be close to linear, thus indicating that arbitrarily large instances of these problems can be solved in fixed time by simply increasing p. In [12], Gupta and Kumar identified the classes of parallel systems which yield linear and sub-linear time-constrained speedup curves.

The scaled speedup concept has rehabilitated the massive parallel computing after Amdahl's 'attack'. From application viewpoint Gustafson-Barsis' Low answers the next questions:

1. a parallel program being run on 10 PUs spends 3% of time executing the serial code. How much has it been accelerated?

$$S_p = \sigma + (1 - \sigma)\, p = 0.03 + 0.97 \cdot 10 = 9.73\,,$$

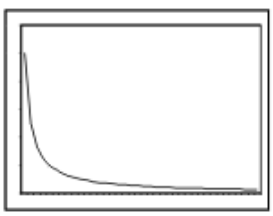
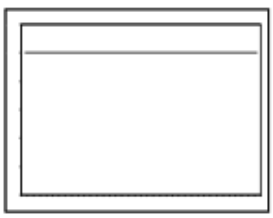
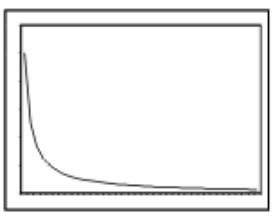
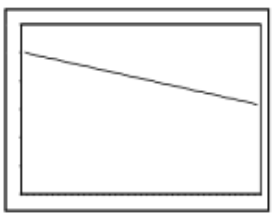
2. what is the maximal serial fraction for 7 times speedup by use of 8 PUs? The answer is $S_p = \sigma + (1 - \sigma)\, p \Rightarrow p - S_p = \sigma\,(p - 1) \Rightarrow \sigma = \frac{p - S_p}{p - 1} = \frac{8-7}{8-1} = \frac{1}{7} \approx 14\%$.

The concept of 'work size' sometimes need to be cleared up. Consider the matrix multiplication problem: multiplying two n-by-n matrices. The matrices contains of $M(n) = 3n^2$ elements and the multiplication requires $C(n) = O(n^3)$ floating point operations. What of the listed values n, $n(n+1)$ or n^3 is the work size? The answer is grounded on equity of the Amdahl's and Gustafson's prediction of speedup $S_p = p$ for an ideally paralleled solution of the problem ($\sigma = 0$). As result:

$$\frac{Work(p)}{Work(1)} = \frac{Time(1)}{Time(p)} \Rightarrow Work(p) = \frac{Work(1) \cdot Time(1)}{Time(p)} = \frac{const}{Time(p)} = O\left(\frac{C(n)}{p}\right) = O\left(\frac{n^3}{p}\right).$$

So the work size here is a number of necessary arithmetic operations, not the data size itself. The relations between Amdahl's and Gustafson-Barsis' Laws are shown in Tab. 3.2.

Table 3.2: Comparison of Amdahl's and Gustafson-Barsis' Laws

	Amdahl's Approach	**Gustafson's Approach**
What is constrained?	Work	Computation time
What is changed with p?	Computation time	Work
What is a scale for the serial fraction?	Computation time of a serial program	Computation time of either a serial or a parallel program
What is the dependency of execution time from number of PUs for a fixed serial fraction?		
What is the dependency of speedup from serial fraction for a fixed number of PUs?		

Both the laws deal with scaling of a problem. Their approach can be generalized if account other possible resource constrains, e.g. memory constrains:

$$Speedup\,(p) = \frac{Performance\,(p)}{Performance\,(1)} = \frac{\text{Throughput (p)}}{\text{Throughput (1)}} = \frac{Work\,(p)\,/Time\,(p)}{Work\,(1)\,/Time\,(1)}.$$

Let scale up the problem such that the memory usage per processor is fixed – so implicitly the problem size gets larger as more processors are used. In fact, execution time can be increased when more processors are used, i.e., $Time(p) > Time(1)$, because problem size can get very large.

By now we assumed use of ideal parallel computers. In reality a parallel implementation always leads to some additional delays due to synchronization, memory access issues or communication expenses. The next concepts deal

with these overheads.

1. **Total Parallel Overhead** T_o: the sum total of all the overhead incurred due to parallel processing by all the processors. It includes communication costs, non-essential work and idle time due to synchronization and serial components of the algorithm. Mathematically,

$$T_o = pT_P - T_1.$$

   In order to simplify the analysis, we assume that T_o is a non-negative quantity. This implies that speedup is always bounded by p. For instance, speedup can be superlinear and T_o can be negative if the memory is hierarchical and the access time increases (in discrete steps) as the memory used by the program increases. In this case, the effective computation speed of a large program will be slower on a serial processor than on a parallel computer employing similar processors. The reason is that a sequential algorithm using M bytes of memory will use only M/p bytes on each processor of a p-processor parallel computer. The core results of the paper are still valid with hierarchical memory, except that the scalability and performance metrics will have discontinuities, and their expressions will be different in different ranges of problem sizes. The flat memory assumption helps us to concentrate on the characteristics of the parallel algorithm and architectures, without getting into the details of a particular machine.

   For a given parallel system, T_o is normally a function of both W and p and so its precise declaration is $T_o(W, p)$.

2. **Efficiency** E: the ratio of speedup S to the number of processors p. Thus,

$$E = \frac{T_1}{p\,T_P} = \frac{1}{1 + T_o/T_1}. \tag{3.1}$$

   Kumar and Rao [23] developed a scalability metric relating the problem size to the number of processors necessary for an increase in speedup in proportion to the number of processors. This metric is known as the **isoefficiency function.** If a parallel system is used to solve a problem instance of a fixed size, then the efficiency decreases as p increases. The reason is that T_o increases with p. For many parallel systems, if the problem size W is increased on a fixed number of processors, then the efficiency increases because T_o grows slower than W. For these parallel

systems, the efficiency can be maintained at some fixed value (between 0 and 1) for increasing p, provided that W is also increased. We call such systems **scalable** parallel systems [27]. For some parallel systems, the maximum obtainable efficiency $E^{\max}$ is less than 1. Even such parallel systems are considered scalable if the efficiency can be maintained at a desirable value between 0 and $E^{\max}$.

For different parallel systems, W should be increased at different rates with respect to p in order to maintain a fixed efficiency. For instance, in some cases, W might need to grow as an exponential function of p to keep the efficiency from dropping as p increases. Such parallel systems are poorly scalable. The reason is that on these parallel systems, it is difficult to obtain good speedups for a large number of processors unless the problem size is enormous. On the other hand, if W needs to grow only linearly with respect to p, then the parallel system is highly scalable. This is because it can easily deliver speedups proportional to the number of processors for reasonable problem sizes.

The rate at which W is required to grow w.r.t. p to keep the efficiency fixed can be used as a measure of scalability of the parallel algorithm for a specific architecture. If W must grow as $f_E(p)$ to maintain an efficiency E, then $f_E(p)$ is defined to be the isoefficiency function for efficiency E and the plot of $f_E(p)$ vs. p is called the isoefficiency curve for efficiency E. Equivalently, if the relation $W = f_E(p)$ defines the isoefficiency curve for a parallel system, then p should not grow faster that $f_E^{-1}(W)$ if an efficiency of at least E is desired.

In Kumar and Rao's framework, a parallel system is considered scalable if its isoefficiency function exists; otherwise the parallel system is unscalable. The isoefficiency function of a scalable system could, however, be arbitrarily large; i.e., it could dictate a very high rate of growth of problem size w.r.t. the number of processors. In practice, the problem size can be increased asymptotically only at a rate permitted by the amount of memory available at each processor. If the memory constraint does not allow the size of the problem to increase at the rate necessary to maintain a fixed efficiency, then the parallel system should be considered unscalable from a practical point of view.

For example let's return to square matrices multiplication. The time complexity of the sequential algorithm is $T_1 = O(n^3)$. The only overhead is performing the broadcast/all-gather operations, and so for the rectangular

block summing algorithm and the blocked shared memory access:

$$T_o = O\left(\left(n \cdot \frac{n}{\sqrt{p}}\right) \cdot p\right) = O\left(n^2\sqrt{p}\right).$$

For isoefficiency conditions $O\left(n^3\right) \geq O\left(n^2\sqrt{p}\right) \Rightarrow n^3 \geq Cn^2\sqrt{p} \Rightarrow n \geq C\sqrt{p}$. Furthermore, given the memory utilization $M\left(n\right) = 3n^2$, the scalability function is given by $\frac{M(n)}{p} = \frac{3\left(C\sqrt{p}\right)^2}{p} = 3C^2 = const$, that means the algorithm is highly scalable.

Less overhead efficient version of this algorithm may suppose full matrices broadcasting. In this case $T_o = O\left(n^2p\right)$, $O\left(n^3\right) \geq O\left(n^2p\right) \Rightarrow n^3 \geq Cn^2p \Rightarrow n \geq Cp$, $M\left(n\right)/p = 3C^2p$ that means the algorithm is not highly scalable. (To maintain constant efficiency, memory utilization per processor must grow linearly with the number of processors).

Isoefficiency analysis has been found to be very useful in characterizing the scalability of a variety of parallel systems. An important feature of isoefficiency analysis is that in a single expression, it succinctly captures the effects of characteristics of a parallel algorithm as well as the parallel architecture on which it is implemented. By performing isoefficiency analysis, one can test the performance of a parallel program on a few processors, and then predict its performance on a larger number of processors.

Isoefficiency is a helpful tool for theoretical verification of parallel algorithms and architectures. But it says nothing about nature of experimentally observed overheads, and so is worse suitable for optimization of real programs. To fill this gap Karp and Flatt [20] introduced experimentally determined **serial fraction** $\sigma\left(p\right)$ as a new metric (widely recognized as Karp-Flatt Metric) for measuring the performance of a parallel system on a fix-sized problem. If S_p is the speedup on a p-processor system, then serial fraction is defined as

$$\sigma\left(p\right) = \frac{1/S_p - 1/p}{1 - 1/p}.$$

Smaller values of $\sigma\left(p\right)$ are considered better. The constant value of $\sigma\left(p\right)$ is possible if the loss in speedup is only due to serial component (i.e., if there are no other overheads). Linear approximation of $\sigma\left(p\right)$ rates slowdown impact of serial fraction vs. overheads (Tab. 3.3).

Table 3.3: Karp-Flatt Metric

The serial fraction dominates in the speedup limiting							
p	2	3	4	5	6	7	8
S_p	1.8	2.5	3.1	3.6	4.0	4.4	4.7
$\sigma(p)$	0.1	0.1	0.1	0.1	0.1	0.1	0.1
The overheads dominate in the speedup limiting							
p	2	3	4	5	6	7	8
S_p	1.9	2.6	3.2	3.7	4.1	4.5	4.7
$\sigma(p)$	0.07	0.075	0.08	0.085	0.09	0.095	0.1

If the value of $\sigma(p)$ decreases with increasing p, then Karp and Flatt consider it to be explained by phenomena such as superlinear speedup effects or cache effects.

The Karp-Flatt Metric can be used as a pure scalability flag with some restrictions. If serial fraction increases with the number of processors, then it is considered as an indicator of rising communication overhead, and thus an indicator of poor scalability. On the contrary, [22] shows that serial fraction can decrease for perfectly normal programs. Assuming that the serial and the parallel algorithms are the same, $\sigma(p)$ can be approximated by T_o/pT_1: $S_p = \frac{T_1}{T_P} = \frac{T_1 p}{T_1 + T_o}$, for big p, $1 - \frac{1}{p} \approx 1 \Rightarrow \sigma(p) \approx \frac{1}{S_p} - \frac{1}{p} = \frac{T_o + T_1}{pT_1} - \frac{1}{p} = \frac{T_o}{pT_S}$.

For a fixed W (and hence fixed T_S), $\sigma(p)$ will decrease provided T_o grows slower than $O(p)$. This happens for some fairly common algorithms such as parallel FFT on a SIMD hypercube. Also, the parallel algorithms for which $\sigma(p)$ increases with p for a fixed W are not uncommon. For instance, for computing vector dot products on a hypercube $T_o > O(p)$ and hence $\sigma(p)$ increases with p if W is fixed. But as shown in [13], this algorithm-architecture combination has an isoefficiency function of $O(p \log p)$ and can be considered quite scalable.

The 4 described asymptotic estimates (Amdahl's and Gustafson-Barsis' Law, isoefficiensy and Karp-Flatt Metrics) make up a 'gentleman's set' of highly parallel system performance and scalability measures. Many other ideas, metrics and approaches are reviewed in survey [21].

3.7 Paradigms of Parallel Programming

There is no simple recipe for designing parallel algorithms. However, one can benefit from a methodological approach. It allows the programmer to focus on machine-independent issues such as concurrency early in the design process and machine-specific aspects later.

The history of parallel design methodologies had started in '70s. The first generation methodologies appeared when parallel computers were rare and high cost specialized solutions for extremely important problems and domains. At that time parallel algorithms were developed by first class researchers and prominent engineers. They understood underlying concepts and implementation details very well, but the area of general purpose parallel algorithms was not yet mature. So the most important issues targeted performance and scalability analysis, so the methodologies of this type were build upon theoretical models of a parallel computer. Among them the most influenced are the next three approaches.

PRAM (Parallel Random Access Machine), a descendent of Random Access Machine, is a theoretical model of parallel computation in which an arbitrary but finite number of processors can access any value in an arbitrarily large shared memory in a single time step [1]. Introduced in the 1970s it still remains popular since it is theoretically tractable and gives algorithm designers a common target. The three most important variations on this model are:

- EREW (exclusive read exclusive write) where any memory location may be access only once in any one step,
- CREW (concurrent read exclusive write) where any memory location may be read any number of times during a single step but written to only once after the reads have finished,
- CRCW (concurrent read concurrent write) where any memory location may be written to or read from any number of times during a single step. Some rule or priority must be given to resolve multiple writes.

PRAM cannot be emulated optimally on all architectures. Problem lies in the assumption that every processor can access the memory simultaneously in a single step. For example in hypercube messages must take several hops

between source and destination and it grows logarithmically with the machines size. As a result any hypercube network cluster will experience a logarithmic slowdown relative to the PRAM model as its size increases.

BSP (Bulk Synchronous Parallel) model was proposed in 1989. BSP allows for the programmer to design an algorithm as a sequence of large step (supersteps in the BSP language [5]) each containing many basic computation or communication operations done in parallel and a global synchronization at the end, where all processors wait for each other to finish their work before they proceed with the next superstep.

BSP comprises of a computer architecture, a class of algorithms, and a function for charging costs to algorithms. The BSP computer consists of a collection of processors, each with private memory, and a communication network that allows processors to access each others memories. BSP satisfies all requirements of a useful parallel programming model:

- simple enough to allow easy development and analysis of algorithms,
- realistic enough to allow reasonably accurate modelling of real-life parallel computing,
- there exists a portability layer in the form of BSPlib,
- it has been efficiently implemented in the Oxford BSP toolset and Paderborn University BSP library.

Currently being used as a framework for algorithm design and implementation on clusters of PCs, networks of workstations, shared-memory multiprocessors and large parallel machines with distributed memory. BSPlib currently provide convertor from BSP to MPI-2, so MPI can be used for programming in the BSP style.

The **Task/Channel** model of Ian Foster (1990) represents a parallel computation as a set of tasks that may interact with each other by sending messages through channels [10]. It can be viewed as a directed graph where vertices represent tasks and directed edges represent channels. A task is a program, its local memory, and a collection of I/O ports. It can send local data values to other tasks via output ports, it also receives data values from other tasks via input ports. A channel is a message queue that connects one task's output port with another task's input port. Data values appear at the input

port in the same order as they were placed in the output port of the channel. Tasks cannot receive data until it is sent (i.e. receiving is blocked). Sending is never blocked.

The Foster's approach include a four stage design process commonly referred to as **PCAM**:

- partitioning – the process of dividing the computation and data into pieces,
- communication – the pattern of send and receives between tasks,
- agglomeration – process of grouping tasks into larger tasks to simplify programming or improve performance,
- mapping – the processes of assigning tasks to processors.

Task/channel with PCAM encourages parallel algorithm designs that maximize local computations and minimize communications.

The PCAM marked an important change in parallel software development audience which took part in late '80s. With parallel computers distributing especially in the scientific research area, people not familiar with HPC became involved in concurrent programming. Since that time program engineering aspects of parallel have come on foreground scene.

The **RAS** (Result, Agenda, Specialist) model answered those changed by proposing a technology to write a parallel program for problems of few typical classes [7]. It was introduced in 1990 by Nicholas Carriero and David Gelernter. The RAS procedure targets most common cases of natural parallelism:

1. choose a pattern that is most natural to the problem,
2. write a program using the method that is most natural for that pattern, and
3. if the resulting program is not efficient, then transform it methodically into a more efficient version.

RAS helps to envision parallelism in terms of Result, Agenda or Specialists.

1. ***Result*** focused on the shape of the finished product. Result means planning a parallel application around a data structure yielded as the final result. We get parallelism by computing all elements of the result simultaneously. It is a good starting point for any problem whose goal is to produce a series of values with predictable organization and inter-dependencies. If all the values are independent then all computations start in parallel. However, if some elements cannot be computed until certain other values are known, then those tasks are blocked. As a simple example consider adding two matrices of equal size.

2. ***Agenda*** focused on the list of tasks to be performed. Agenda means planning a parallel application around a particular agenda of tasks, and then assign many processes to execute the tasks. The Agenda parallelism adapts well to many different problems. The most flexible is the master-worker paradigm in which a master process initializes the computation and creates a collection of identical worker processes. Each worker process is capable of performing any step in the computation. Workers seek a task to perform and then repeat. When no tasks remain, the program is finished. An example could be a circuit simulation where each element is realized by a separate process.

3. ***Specialist*** focused on the make-up of the work. Plan an application around an ensemble of specialists connected into a logical network of some kind. Parallelism results in all nodes being active simultaneously – much like pipe-lining. Specialist parallelism involves programs that are conceived in terms of a logical network in which each node executes an autonomous computation and inter-node communication follows predictable paths. An example could be a circuit simulation where each element is realized by a separate process.

The main conclusion from RAS success was that most parallel applications can be classified into well defined programming paradigms. Each paradigm is a class of algorithms that have the same control structure. Experience suggests that there are a relatively few paradigms underlying most parallel programs. The choice of paradigm is determined by the computing resources which can define the level of granularity and type of parallelism inherent in the program which reflects the structure of either the data or application.

The most systematic definition of the paradigms comes from a technical report from the University of Basel (1993) entitled **BACS** (Basel Algorithm Classification Scheme). The factors which BACS have used to characterize a parallel algorithm include the process properties (structure, topology, execution), the interaction properties, the data properties (partitioning, placement). The following paradigms have been identified and described:

- Task-Farming (aka Master/Worker, Manager/Worker, Master/Slave),
- Single Program Multiple Data (SPMD),
- Data Pipelining (aka functional decomposition, data-flow parallelism: each PU executes a small part of the total algorithm, data units sequentially pass from one PU to another),
- Divide and Conquer (a problem is divided into two or more subproblems. Each subproblem is solved independently and may be divided the same manner),
- Speculative Parallelism (Employed when it is difficult to obtain parallelism through any one of the previous paradigms. Deals with complex data dependencies which can be broken down into smaller parts using some speculation or heuristic to facilitate the parallelism).

A most comprehensive modern methodology of parallel programming developed by Timothy Mattson, Beverly Sanders and Berna Massingill is called **PLPP** (Pattern Language for Parallel Programming) [29, 35]. Despite its sounding name, the pattern language is not a programming language. It is an embodiment of design methodologies which provides domain specific advise to the application designer.

Design Pattern is a description of a good solution to a recurring problem in a particular context. The conception was introduced in 1977 by Christopher Alexander for city planning and architecture. Usually design patterns are presented in a prescribed format:

- Pattern Name,
- Pattern Context,
- Pattern Goals & Constraints,

– Solution.

In 1987 by Beck and Cunningham have first brought it to software engineering. Design patterns for object-oriented programming have been released by Gamma, Helm, Johnson and Vlissides in 1994 [11]. MacDonald, Szafron and Schaeffer were first to apply patterns for parallel programing [25], but Mattson, Sanders and Massingill have applied patterns more consistently, comprehensively and with high refinement.

PLPP consists of a set of patterns that guide the programmer through the entire process of developing a parallel program, including patterns that help find the concurrency in the problem, patterns that help find the appropriate algorithm structure to exploit the concurrency in parallel execution, and patterns describing lower-level implementation issues.

The pattern language is organized into four design spaces:

1. ***Finding Concurrency***. This design space includes high-level patterns that help find the concurrency in a problem and decompose it into a collection of tasks,
2. ***Algorithm Structure***. This design space contains patterns that help find an appropriate paralleling paradigm to exploit the concurrency that has been identified,
3. ***Supporting Structures***. This design space represents an intermediate stage between the Algorithm Structure and Implementation Mechanisms design spaces. Two important groups of patterns in this space are those that represent program-structuring approaches and those that represent commonly used shared data structures,
4. ***Implementation Mechanisms***. The design space design space contains patterns that describe lower-level implementation issues. It concerned with how the patterns of the higher-level spaces are mapped into particular programming environments. It is used to provide descriptions of common mechanisms for process/thread management and interaction.

Design patterns of PLPP are presented in a prescribed format:

– problem,

- context,
- forces (goals and constraints),
- solution,
- examples.

Before starting to work with the patterns, the algorithm designer must first consider the problem to be solved and make sure the effort to create a parallel program will be justified: is the problem sufficiently large, and the results sufficiently significant, to justify expending effort to solve it faster? If so, the next step is to make sure the key features and data elements within the problem are well understood. Finally, the designer needs to understand which parts of the problem are most computationally intensive, since it is on those parts of the problem that the effort to parallelize the problem should be focused.

Generally one starts at the top in the Finding Concurrency design space and works down through the other design spaces in order until a detailed design for a parallel program is obtained. Sometimes it is necessary to iterate back and forth among the patterns in one design space or among design spaces. Once the preliminary analysis is complete, the patterns in the Finding Concurrency design space can be used to start designing a parallel algorithm.

Table 3.4: The design space 'Finding Concurrency'

Decomposition Patterns	
'Task Decomposition' Pattern	**'Data Decomposition' Pattern**
Dependency Analysis Patterns	
The '**Group Tasks**' Pattern	
The '**Order Tasks**' Pattern	
The '**Data Sharing**' Pattern	
The '**Design Evaluation**' Pattern	

The patterns in this design space can be organized into three groups (Tab. 3.4). First the Decomposition group patterns are used to decompose the problem into pieces that can execute concurrently. Either task or data decomposition is considered.

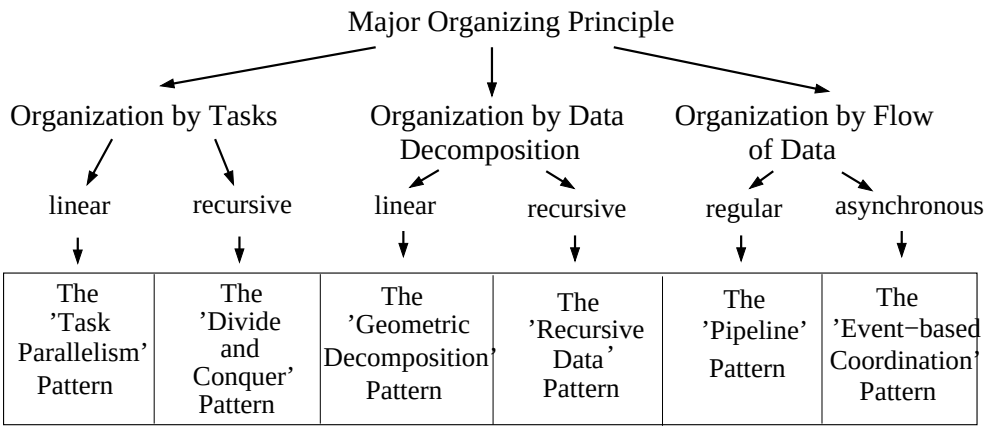
Table 3.5: Summary of the Decomposition group design patterns

Pattern: **Task Decomposition**	Pattern: **Data Decomposition**
Problem: How can a problem be decomposed into tasks that can execute concurrently? Context: In some cases the problem will naturally break down into a collection of independent (or nearly independent tasks). In this case it is easiest to start with the task decomposition pattern. In other cases, tasks are difficult to isolate, and decomposition of data becomes a better starting point. It's not always clear, and both decompositions eventually need to be considered. Forces: *flexibility* – allow to adapt to different implementations *efficiency* – need enough tasks to keep PEs busy, but beware complex decomposition *simplicity* – tradeoff between complex enough for efficiency, and simple enough for debug and maintenance Solution: (1) Try to ensure that tasks are sufficiently independent so that dependency management will be a small fraction of execution time (2) Try to identify as many tasks as possible (we can always merge them later) (3) Try to ensure that a balanced load is possible (based on task granularity and dependencies)	Problem: How can a problem's data be decomposed into units that can be operated on relatively independently? Context: The designer should identify the most computationally intensive parts of the problem, the key data structures used, and how data is used during the problem solution. Start with data decomposition if: The most computationally intensive part of the problem is organized around manipulating large data structure(s) -or- Similar operations are being applied to different parts of the data structure such that different parts can be updated relatively independently Forces: flexibility, efficiency, and simplicity Solution: If task-based decomposition is already done, the data decomposition is driven by the demands of the tasks. If well defined data can be associated with tasks then this is simple. If starting with data decomposition, look at the central data structures defining the problem and see if they can be broken down into chunks that can be operated on concurrently. Examples: array-based computations – update segments of the array; recursive data structures – decompose a tree into sub-trees

Then Dependency Analysis patterns are applied to group the tasks and find dependencies related to timing or synchronization. At last Design Evaluation pattern is used to verify whether the decomposition principle is right selected

and to determine if the design is ready to proceed to the Algorithm Structure design space.

Table 3.6: The design space 'Algorithm Structure'



The patterns of to the Algorithm Structure design space generally correspond to main paradigms of BACS (with exception of Speculative Parallelism not suitable for patterning). But they are classified more strictly and reduced to a nice symmetric scheme (Tab. 3.6) provided obvious guidelines for the pattern selection.

The Finding Concurrency and Algorithm Structure design spaces focus on algorithm expression. At some point, however, algorithms must be translated into programs. The patterns in the Supporting Structures design space (Fig. 4.17) address that phase of the parallel program design process, representing an intermediate stage between the problem-oriented patterns of the Algorithm Structure design space and the specific programming mechanisms described in the Implementation Mechanisms design space.

The items in the Implementation Mechanisms design space are not presented as patterns since in many cases they map directly onto elements within particular parallel programming environments. They were included to provide a complete path from problem description to code. Instead descriptions of available implementation approaches and tools are presented.

Parallel programming issues were deeply developed since 1970s years. Some

Table 3.7: The design space 'Supporting Structures'

Program Structuring Patterns				
Correspondence to the selected algorithm structure:	'SPMD' Pattern	'Master/Worker' Pattern	'Loop Parallelism' Pattern	'Fork/Join' Pattern
Task Parallelism	****	****	****	**
Divide and Conquer	***	**	**	****
Geometric Decomposition	****	*	***	**
Recursive Data	**	*		
Pipeline	***	*		****
Event-based Coordination	**	*		****
Patterns Representing Data Structures				
'**Shared Data**' Pattern		'**Shared Queue**' Pattern		'**Distributed Array**' Pattern

Table 3.8: The design space 'Implementation Mechanisms'

Implementation Mechanisms
The '**UE Management**' Pattern
The '**Synchronization**' Pattern
The '**Communication**' Pattern

theoretical models were introduced and multiple toolkits and libraries were implemented. Their number was estimated as about 250 in the middle of 1990s [9]. Modern approaches like PLPP or Object-Oriented Parallel System [38] generalize the long time experience of High Performance Computing. They help to find concurrency and to develop parallel programs on the base of a small number of ready detailed patterns.

3.8 Massive Parallel Data Processing

I/O bottlenecks have always been a major issue in Computer Science. As early as 1967, Amdahl addressed the issue of storage and computation efficiency. Almost forty years later, this lack of performances is confirmed in [36] and this trend is likely to continue.

Data intensive applications have imposed specific constraints due to cluster

architectures and their different access patterns. First studies of [6] and [30] characterized parallel I/O accesses and observed recurrence forms and determined patterns: huge amounts of small and non-contiguous requests. The proposed solutions to reduce congestion issues and improve I/O access performances can be classified in two categories: Parallel File systems and Parallel I/O Libraries. The former tries to find the best trade-offs between parallel I/O and efficient exploitation of hardware capabilities whereas the latter is focused on parallel I/O aspects and portability constraints. [24] overview both the approaches. Interesting I/O framework for grid systems is proposed in [33].

An execution time model for data-intensive parallel computations

The massive dataset processing by each of N synchronous parallel processes is presented as an alternation of calculations and I/O operations [39]. Other S processes or devices wait for I/O requests from the calculation processes, put them in the infinite queue and execute in the FIFO order. Communication overheads not related to I/O are negligibly small. The time of i steps execution

$$T_N = \max_{n=1..N} \left(t_0^n + \omega_1^n + \tau_1^n + t_1^n + \omega_2^n + \tau_2^n + t_2^n + ... + \omega_i^n + \tau_i^n + t_i^n\right) ,$$

where calculation time for the i-th step of n-th computing process is t_i^n, necessary I/O operation time is τ_i^n and related waiting time is ω_i^n. The model behavior depends on the proportion of computations to I/O. The next two extremes are examined:

- *weak load* when the I/O processes have time to serve the queue before beginning the next I/O step ($\forall\, i\ \sum_n \tau_i^n << t_{i+1}^n$) and
- *full load* when the queue raises at each step ($\forall\, i\ \sum_n \tau_i^n \geq t_{i+1}^n$).

Let $c_N = \mu\left(\sum_i t_i^n\right)$ is the average computation time, and $d_N = \mu\left(\sum_i \tau_i^n\right)$ is the average I/O time.

For the **partitioning** data distribution policy under weak load:

$$\mu\left(T_N\right) = \frac{1}{2}c_N + d_N + \sqrt{\frac{1}{4}c_N^2 + d_N^2 \frac{\mu\left(\tau^2\right)}{2\mu\left(\tau\right)^2}}. \tag{3.2}$$

For the **sharing** or **duplication** data distribution policy under weak load:

$$\mu(T_N) = c_N + d_N + (N-1)\,\tau^S_{\max}. \tag{3.3}$$

In the case of full load regardless of the data distribution policy:

$$\mu(T_N) = Nd_N + t_{const}, \quad \text{where} \quad t_{const} = t^1_0 + t^N_i. \tag{3.4}$$

Experiments corroborate an assertion the program execution time bottom close to the intersection of weak load and full load diagrams: for example see Fig. 3.11 of Duplex Wave Migration [19] calculation time.

The first set of plots (A, B) compares the performance of migration with pre-calculated time cubes with the theory. The second set (C, D) demonstrates use of the weak and full load diagram crosspoint as an estimate for the computational time diagram cusp that is a best fit for the number of computing processes. Despite a big variation of the experimental diagrams in the right branches because of caching and communicational effects the really useful left branches and cusp points are predicted with high precision on the sequential run results.

Data compression effect on speedup

Under assumption of big data sets $d_N >> t_{const}$ and $d_N >> \tau^S_{\max}$. For efficient parallel calculations $N >> 1$. In such conditions at the execution time bottom $c_N/d_N \approx N-1$.

The equation works in two directions. For a 'black box' program it allows to estimate resource distribution for the best N experimentally found by multiple runs. For open source or internally developed program the execution time bottom allows to approximate the best N by statistic analysis of few runs.

Let analyze the performance changes caused by use of a compression procedure. Denote compression average velocity $v = q_N/d_N$ and average rate $r > 1$. Compression isoefficiency can be estimated as:

$$E = \left(\frac{k-1}{k} + \frac{c_N}{d_N}\right) \Big/ \left(v + \frac{c_N}{d_N}\right).$$

The bigger compression isoefficiency the higher calculation acceleration is:

- compression accelerates calculations for any number of PUs if $E \geq 1$,

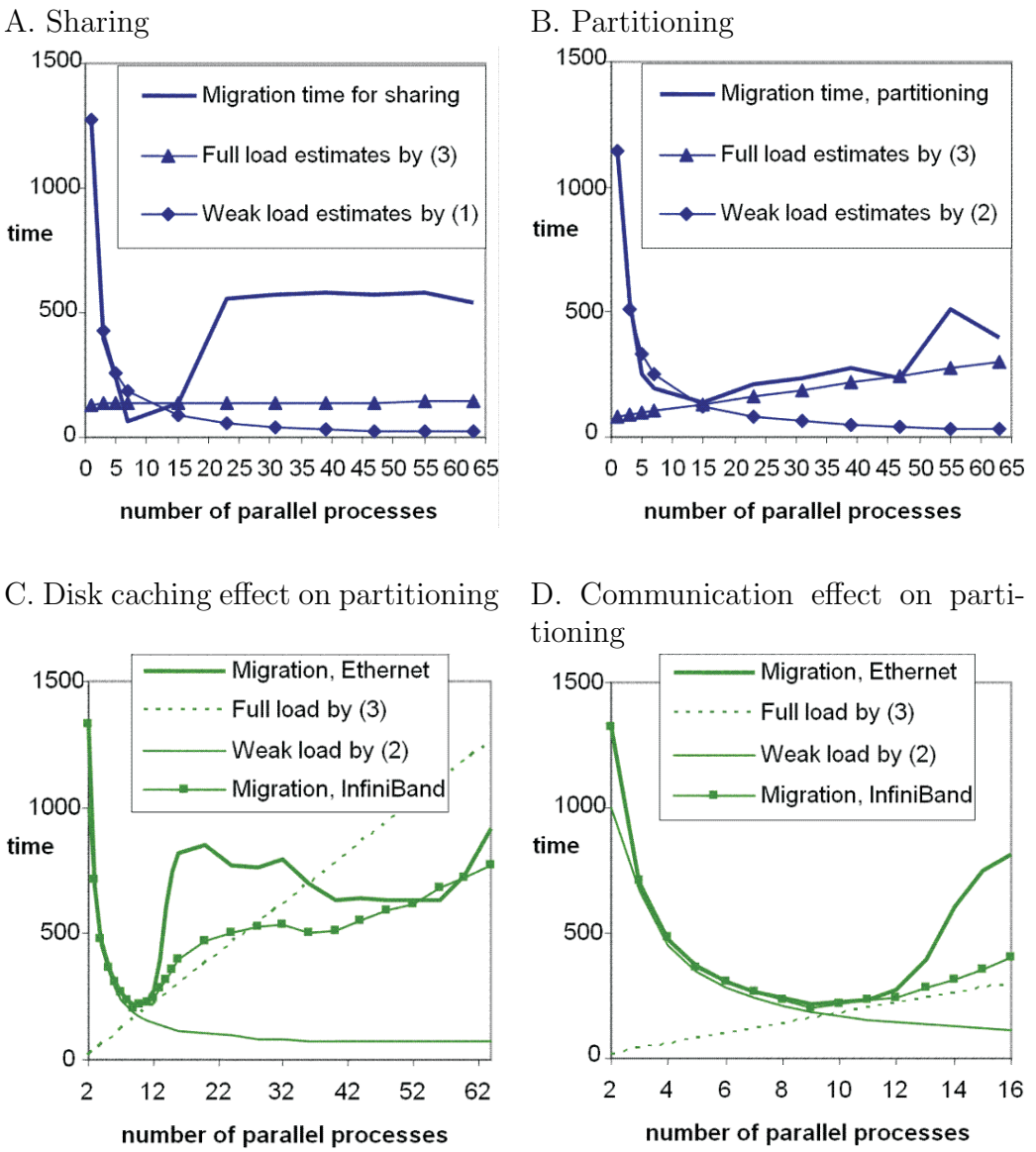


Figure 3.11: Massive data processing experiments

- compression impedes calculations for a cluster of $N_{\max}$PUs if $E < N/N_{\max}$,
- if $E < 1$compression is not efficient for number of PUs less than $E \cdot N$.

In the case of known best N, E can be rewritten as $E = \frac{N-\frac{1}{k}}{N+v-1}$. Big data

sets are very common for the industry. Despite advances in file storage hardware development and success of parallel file systems I/O still is a common bottleneck for parallel computations. In such cases application of more PUs can even impede the whole calculations. Data compression decreases both the network and storage load. As result the computation acceleration can exceed anticipated magnitude. Besides, compression provides conditions to increase the cluster PU number without overheating the storage and so multiplies performance effect. Because of such double effect data compression can be generally recommended for acceleration of cluster computations for seismic data processing.

Simple estimates of this section help to compare and select compression algorithms for particular hardware-software complex using some recordable information about already passed jobs. Their application together with formulas of performance forecasting can help administrators in balancing load of a cluster when multiple applications run simultaneously.

3.9 Conclusion

This chapter briefly discussed current (2010) state of art in the area of Multiprocessor Computing Structures. Only the most valuable and prospective (from the author's viewpoint) representatives of each hardware/software architecture are described. Some closely related issues of performance/scalability analysis and parallel programming are presented to help understand the general considerations put in background of the modern HPC systems.

Bibliography

[1] S.G. Akl: The design and analysis of parallel algorithms, Prentice-Hall, 1989.

[2] G.M. Amdahl: Validity of the single processor approach to achieving large scale computing capabilities. In AFIPS Conf. Proceedings, pages 483-485, 1967.

[3] M. Baker, R. Buyya, and D. Laforenza: Grids and Grid technologies for wide-area distributed computing, http://www.buyya.com/papers/gridtech.pdf

[4] http://www.browndeertechnology.com/stdcl.html#man_pages

[5] R.H. Bisseling: Parallel scientific computation: a structured approach using BSP and MPI, Oxford Press, 2004.

[6] P.E. Crandall, R.A. Aydt, A.A. Chien, D.A. Reed: Input/output characteristics of scalable parallel applications, Proc. of Supercomputing '95, San Diego, 1995.

[7] N. Carriero, D. Gelernter: How to Write Parallel Programs: A First Course, MIT Press, 1990.

[8] http://www.ex.ua/get/1255235 (in Russian)

[9] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L.A. Torczon: White Sourcebook of Parallel Computing, Morgan Kaufmann, 2003.

[10] I. Foster: Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering, Addison-Wesley Longman Publishing, 1995.

[11] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns; Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994.

[12] A. Gupta, V. Kumar: Performance properties of large scale parallel systems. Journal of Parallel and Distributed Computing, 19, pp. 234-244, 1993.

[13] A. Gupta, V. Kumar, A.H. Sameh: Performance and scalability of preconditioned conjugate gradient methods on parallel computers. Proc. of the 6th SIAM Conf. on Parallel Processing for Scientific Computing, pp. 664-674, 1993.

[14] P.N. Glaskowsky: NVIDIA's Fermi: The First Complete GPU Computing Architecture, http://www.nvidia.co.uk/content/PDF/fermi_white_papers /P.Glaskowsky_NVIDIA's_Fermi-The_First_Complete_GPU_ Architecture.pdf

[15] J.L. Gustafson, G.R. Montry, R.E. Benner: Development of parallel methods for a 1024-processor hypercube. SIAM Journal on Scientific and Statistical Computing, 9(4), pp. 609-638, 1988.

[16] J.L. Gustafson: Reevaluating Amdahl's law. Communications of the ACM, 31(5), pp. 532-533, 1988.

[17] H.P. Hofstee: Power efficient processor architecture and the Cell processor, Proc. of the 11th Int. Symp. on High-Performance Computer Architecture (HPCA-11), 2005.

[18] P.G. Howard: GPU Computing: More exciting times for HPC! http://www.microway.com/pdfs/microway_GPU_whitepaper_2009-05.pdf

[19] A.C. Kostyukevych, N.A. Marmalevsky, Z.V. Gornyak, Y.B. Roganov, V.V. Mershchy, B. Link, I.U. Khromova: Study of vertical heterogeneities by using duplex wave migration, Seismic exploration technology, 1, pp. 3-14, 2008.

[20] A.H. Karp, H.P. Flatt: Measuring parallel processor performance. Communications of the ACM, 33(5), pp. 539-543, 1990.

[21] V. Kumar, A. Gupta: Analysis of scalability of parallel algorithms and architectures: a survey, Proc. of the 5th int. conf. on Supercomputing, pp. 396-405, Cologne, West Germany, 1991.

[22] V. Kumar, A. Grama, A. Gupta, G. Karypis: Introduction to Parallel Computing: Design and Analysis of Algorithms. Benjamin/Cummings, Redwood City, CA, 1994.

[23] V. Kumar, V.N. Rao: Parallel depth-first search, part II: Analysis. International Journal of Parallel Programming, 16(6), pp. 501-519, 1987.

[24] A. Lebre, G. Huard, Y. Denneulin, P. Sowa: I/O Scheduling Service for Multi-Application Clusters, IEEE Int. Conf. on Volume, Barcelona, pp. 21-23, 2006.

[25] S. MacDonald, J. Schaeffer, D. Szafron: Pattern-based object-oriented parallel programming, Proc. of ISCOPE'97, LNCS Vol 1343, pp. 267-274, 1997.

[26] J.R. McGraw, T.S. Axelrod: Exploiting Multiprocessors: Issues and Options, IEEE Software, 5(5), pp. 7-25, 1988.

[27] C. Moler: Another look at Amdahl's law. Technical Report TN-02-0587-0288, Intel Scientific Computers, 1987.

[28] http://www.mpi-forum.org/

[29] T.G. Mattson, B.A. Sanders, B.L. Massingill: A Pattern Language for Parallel Programming, Addison Wesley Software Patterns Series, 2004,

[30] N. Nieuwejaar, D. Kotz, A. Purakayastha, C.S. Ellis, M. Best: File-access characteristics of parallel scientific workloads, IEEE Transactions on Parallel and Distributed Systems, 7(10), pp. 1075-1089, 1996.

[31] http://developer.download.nvidia.com/OpenCL/NVIDIA_OpenCL_JumpStart_Guide.pdf

[32] http://www.khronos.org/opencl/

[33] R. Oldfield, D. Kotz: Armada: A parallel I/O framework for computational grids, in Future Generation Computing Systems (FGCS),18(4), pp.501-523, 2002.

[34] http://openmp.org/wp/

[35] http://www.cise.ufl.edu/research/ParallelPatterns

[36] O.L. Perevozchikova, V.G. Tulchinsky, R.A. Iushchenko: Building and optimization of parallel computers for massive dataset processing, Cybernetics and System Analysis, 4, pp. 117-129, 2006.

[37] X.-H. Sun and L.M. Ni: Another view of parallel speedup, Proceedings of Supercomputing'90, pp. 324-333, 1990.

[38] E. Sonoda, G. Travieso: The OOPS framework: high level abstractions for the development of parallel scientific applications. Proceedings of OOPSLA'06, Portland, Oregon, USA, pp. 659-660, 2006.

[39] V.G. Tulchinsky, A.K. Iushchenko, R.A. Iushchenko: Parallel Computing Optimization for Processing the Seismic Data, 70th EAGE Conference & Exhibition, Rome, Extended Abstract # P056, 2008.

[40] http://en.wikipedia.org

[41] P.H. Worley: The effect of time constraints on scaled speedup. SIAM Journal on Scientific and Statistical Computing, 11(5), pp. 838-858, 1990.

Part II

Numarical modelling

Chapter 4

3D BEM Froward Problem for Diffusive Optical Tomography

T. Grzywacz, J. Sikora, S. Wójtowicz, P. Komada

4.1 Introduction

In recent years Optical Tomography (OT) has emerged as a highly active and viable field of research, due to advances both in measurement technology and in theoretical and practical understanding of the nature of the image of reconstruction problem. For recent reviews see [32, 50, 23, 5, 3, 21, 12]. An increasingly active topic within this field is the development of an efficient and accurate method for calculating the intensity of light internal to, and on the boundary of an object under experimental investigation, sometimes referred to as the *Forward Problem*. Existing methods are either deterministic, based on the solutions to governing equations, or stochastic based on simulations of the individual scattering and absorption events undertaken by each photon. The former includes analytical expressions based on Green functions [3], and numerical methods based on Finite Difference method (FDM) or Finite Element Methods (FEM) [6, 44, 43, 34, 31]. However, a generally applicable model of the forward problem in three-dimensional space is still not a fully solved problem.

In this chapter we introduce another standard technique for solution of Partial Differential Equations (PDE) in general geometries: the Boundary Element Method (BEM), which has received substantial attention in numerical modelling of fields [26, 15, 33, 16, 19, 48, 29]. The advantages and disadvantages of BEM are well known [10, 11, 13, 20] and they will not be repeated here, but instead we will concentrate on some specific features useful in OT. Recently BEM has been used in Diffusing-wave spectroscopy for determining the correlation function for different boundary conditions and source properties in a cone-plate geometry[48], but has received very little attention in OT.

Like Finite Element Method (FEM), the Boundary Element Method (BEM) provides a general numerical tool for the solution of complex engineering problems. In the last decades, the range of its applications has remarkably been enlarged. This chapter is dealing with the application of BEM to Optical Tomography, a new – to our best knowledge – field of application.

Nevertheless, the BEM still demands an explicit expression of a fundamental solution, which is only known in simple cases. Therefore, in Optical Tomography BEM is restricted to a diffusion approximation of transport equation.

But hopefully this restriction can be lifted in future, as just recently an alternative BEM-formulation, based on the Fourier transform, has been released [18]. The new formulation only needs the Fourier transform of the fundamental solution, which can be constructed, in contrast to the fundamental solution itself, for all linear and homogeneous differential operators by a simple matrix inversion of the transformed differential operator. Hence, the realm of applications of BEM can be extended remarkably.

In recent Optical Tomography applications, researcher's attention is focused on three-dimensional problems. They are a lot more difficult than those defined in the two-dimensional space. Mainly due to the geometry which demands a sophisticated discretization with enormously big number of unknowns. Such problems are named **'Large Scale Problems'**.

BEM is characterized by the boundary-only property of the algorithm. This property reduces the number of unknowns in BEM as compared to those in methods of the domain type such as Finite Difference Method (FDM) or Finite Element Method (FEM). However, the reduced number of unknowns does not necessarily lead to improved efficiency, because BEM generally produces a full asymmetric matrix of coefficients, while the matrices for FDM or FEM are usually sparse and very often symmetric.

Because of this drawback, BEM has so far been considered to be less efficient than these domain type competitors in large scale problems. However, the situation is changing with the recent breakthrough introduced by the so-called "fast-BEMs"' based on techniques such as multiple methods [14], panel clustering, the use of wavelet bases, etc.

The fast BEMs can compute potential functions at the all collocation points with $O(N) - O(N(\log N)^m)$ $(m \geq 0)$ operations in problems with N unknowns.

This is a dramatic improvement over the conventional BEMs which have $O(N^2)$ number of operations. Development of the fast BEM is certain to further enhance the status of BEM as a solver of large scale problems.

4.2 Singular and nearly singular integrals

In three dimensional boundary element analysis, computation of integrals is an important aspect since it governs the accuracy of the analysis and also because it usually takes the substantiable part of the CPU time.

The integrals which determine the influence matrices, the internal field and its gradients contain nearly singular kernels of order $1/R^\alpha$ $(\alpha = 1, 2, 3, ...)$ where R is the distance between the source point and the integration point on the boundary element [22].

For planar elements, analytical integration may be possible [33]. However, it is becoming increasingly important, in practical boundary element codes, to use curved elements, such as the isoparametric elements, to model general curved surfaces [46]. Since analytical integration is not possible for general isoparametric curved elements, one has to rely on numerical integration.

When the distance between the source point and the element over which the integration is performed is sufficiently large, compared to the element size, the standard Gauss-Legendre quadrature formula works efficiently.

However, when the source is actually on the element, the kernel becomes singular and the straight forward application of the Gauss-Legendre quadrature formula breaks down. These integrals will be called singular integrals. Singular integrals occur when calculating the diagonals of the coefficient matrix.

When the source is not on the element, but very close to the element, although the kernel is regular in the mathematical sense, the value of the kernel changes rapidly in the neighborhood of the source point. In such case the standard Gauss-Legendre quadrature formula is not practical, since it would require a huge number of integration points to achieve the required accuracy.

These integrals will be called **nearly singular integrals**. Nearly singular integrals occur in practice when calculating influence matrices for thin structures, where distances between different elements can be very small compared to the element size. Such situation is very common for scull or CSF layer modelling in Impedance or Optical Tomography. They also occur when calculating the field or its derivatives at the internal point, very close to the boundary element. Numerous research works have already been published on this subject, for example [1, 19, 22, 27, 28, 15, 35], and they may be classified as in Table 4.1. Let briefly review the methods which using the kernel

Table 4.1: Methods of integration for singular and nearly singular integrals in three dimensional Boundary Element Method

No.	Singular integrals	No.	Nearly singular integrals
I	**Analytical (for planar elements only)**		
II	**Numerical**		
1	Weighted Gauss	1	Element subdivision
2	Singularity subtraction and Taylor expansion	2	Variable transformation
		2a	Double exponential transformation
3	Variable transformation	2b	Cubic transformation
4	Coordinate transformation		
4a	Triangle to quadrilateral	3	Coordinate transformation
4b	Polar coordinates	3a	Polar coordinates
5	Finite part integrals		and modification

$1/R$ as the weight function for generating the Gauss integration points. The singularity subtraction with Taylor expansion method [1] expands the singular kernel by the local parametric coordinates. The main terms containing the singularity are subtracted and integrated analytically and the remaining well behaved terms are integrated by Gaussian quadrature.

Then there are the coordinate transformation methods. The first type is the method of transforming a triangular region into a quadrilateral region, so that the node corresponding to the singularity is expanded to an edge of the quadrilateral, so that the singularity is weakened. This method is used in the

following chapters (see for example 4.4.2). The second type is the method of using polar coordinates (r, Θ) around the source point in the parameter space [13]. This introduces a Jacobian which cancels the singularity $1/R$ type. For higher singularities of order $1/R^2$, which appears in elastostatic (out of the scope of this book), the method for calculating finite part integrals may be used.

Nearly singular integrals turn out to be more difficult and expensive to calculate compared to singular integrals. They are becoming more and more important in practical boundary element codes, since the ability and efficiency to calculate nearly singular integrals governs the code's versatility in treating objects containing thin structures (scull or CSF layer of the human head). The reader interested in this particular problem may consult [22] where a new quadrature scheme for the accurate and efficient evaluation of these nearly singular integrals is presented.

4.3 Governing equations

Let consider the Poisson's equation in three-dimensional space:

$$\nabla^2 \Phi(\mathbf{r}) = b\,, \tag{4.1}$$

where the Φ stands for the arbitrary potential function like temperature or electric potential.

On the surface of the volume the Robin boundary conditions are imposed:

$$\frac{\partial \Phi(\mathbf{r})}{\partial n} = m_R \Phi(\mathbf{r}) + n_R\,, \tag{4.2}$$

where m_R and n_R are known Robin boundary condition coefficients.

The fundamental solution for 3D space is:

$$G(|\mathbf{r} - \mathbf{r}'|) = \frac{1}{4\pi R}\,, \tag{4.3}$$

where $R = |\mathbf{r} - \mathbf{r}'|$ is a distance between $\mathbf{r}$ and $\mathbf{r}'$, given by

$$R = |\mathbf{r} - \mathbf{r}'| = \sqrt{(x - x')^2 + (y - y')^2 + (z - z')^2}\,. \tag{4.4}$$

We can use Green's theorem [3] in its second form to derive an integral equation applicable on the surface:

$$\Phi(\mathbf{r}) + \int_{\Gamma} \frac{\partial G(|\mathbf{r}-\mathbf{r}'|)}{\partial n} \Phi(\mathbf{r}') d\Gamma(\mathbf{r}') = \int_{\Gamma} G(|\mathbf{r}-\mathbf{r}'|) \frac{\partial \Phi(\mathbf{r}')}{\partial n} d\Gamma(\mathbf{r}') + + \int_{\Omega} b\, G(|\mathbf{r}-\mathbf{r}'|) d\Omega(\mathbf{r}') \,. \tag{4.5}$$

To make this equation a truly 'boundary-only' equation, we move the interior load point $\mathbf{r}$ to the boundary which results in the following modification of the previous equation:

$$c(\mathbf{r})\Phi(\mathbf{r}) + \int_{\Gamma} \frac{\partial G(|\mathbf{r}-\mathbf{r}'|)}{\partial n} \Phi(\mathbf{r}') d\Gamma(\mathbf{r}') = \int_{\Gamma} G(|\mathbf{r}-\mathbf{r}'|) \frac{\partial \Phi(\mathbf{r}')}{\partial n} d\Gamma(\mathbf{r}') + + \int_{\Omega} b\, G(|\mathbf{r}-\mathbf{r}'|) d\Omega(\mathbf{r}') \,. \tag{4.6}$$

The function $c(\mathbf{r})$ can be calculated by surrounding the boundary point $\mathbf{r}$ by a small sphere of radius ε and taking each term of Eq. (4.6) in the limit as $\varepsilon \to 0$.

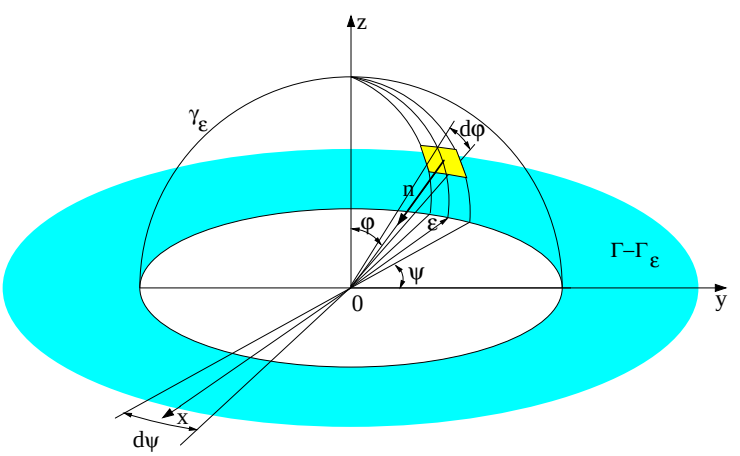


Figure 4.1: Hemisphere around the boundary point for 3D problems

However, as shown in previous sections for the two-dimensional problems, the term $c(\mathbf{r})$ does not need to be calculated explicitly, and can be obtained indirectly by utilizing some simple physical considerations [10]. In particular,

we have $c(\mathbf{r}) = 1/2$ when the observation point lies on a smooth surface, which is the case considered in a frame of this chapter.

To calculate the kernel of the first integral in Eq. (4.6), the Green's function is differentiated with respect to the unit normal at the point $\mathbf{r}'$, as follows:

$$\frac{\partial G(|\mathbf{r}-\mathbf{r}'|)}{\partial n} = \frac{\partial G}{\partial R}\left(\frac{\partial R}{\partial n}\right) = \frac{\partial G}{\partial R}\left[\frac{\partial R}{\partial x}\left(\frac{\partial x}{\partial n}\right) + \frac{\partial R}{\partial y}\left(\frac{\partial y}{\partial n}\right) + \frac{\partial R}{\partial z}\left(\frac{\partial z}{\partial n}\right)\right], \quad (4.7)$$

where the derivatives of the coordinates x, y and z with respect to the unit outward normal $\mathbf{n}$ in point $\mathbf{r}'$ are the components of the outward normal as follows:

$$n_x = \frac{\partial x}{\partial n}, \qquad n_y = \frac{\partial y}{\partial n}, \qquad n_z = \frac{\partial z}{\partial n}, \quad (4.8)$$

and

$$\frac{\partial R}{\partial x} = \frac{x'-x}{R}, \qquad \frac{\partial R}{\partial y} = \frac{y'-y}{R}, \qquad \frac{\partial R}{\partial z} = \frac{z'-z}{R}. \quad (4.9)$$

Therefore, the first kernel can be written as follows:

$$\frac{\partial G(|\mathbf{r}-\mathbf{r}'|)}{\partial n} = \frac{-1}{4\pi R^3}\left[(x'-x)n_x + (y'-y)n_y + (z'-z)n_z\right]. \quad (4.10)$$

To solve the three-dimensional problem numerically, the surface has to be discretized into elements. The zero linear and quadratic order elements were used.

The numerical implementation of three-dimensional problems follows the similar procedure to that of two-dimensional problems [45].

4.3.1 Jacobian

To study boundary elements which are two-dimensional structures placed in the 3D space, first we need to define the way in which we can pass from the xyz global Cartesian system to the ξ_1, ξ_2, ξ_3 system defined over the element, where ξ_1, ξ_2 are oblique coordinates and ξ_3 is in the direction of the normal. The transformation for a given function u is related through the following:

$$\begin{bmatrix} \frac{\partial u}{\partial \xi_1} \\ \frac{\partial u}{\partial \xi_2} \\ \frac{\partial u}{\partial \xi_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi_1} & \frac{\partial y}{\partial \xi_1} & \frac{\partial z}{\partial \xi_1} \\ \frac{\partial x}{\partial \xi_2} & \frac{\partial y}{\partial \xi_2} & \frac{\partial z}{\partial \xi_2} \\ \frac{\partial x}{\partial \xi_3} & \frac{\partial y}{\partial \xi_3} & \frac{\partial z}{\partial \xi_3} \end{bmatrix} \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \\ \frac{\partial u}{\partial z} \end{bmatrix}, \quad (4.11)$$

where the square matrix is the Jacobian matrix (or Jacoby matrix).

Transformation of this type allows us to describe differentials of surface in the Cartesian system in terms of the curvilinear coordinates. A differential of area will be given by:

$$d\Gamma = |\mathbf{n}|\, d\xi_1 d\xi_2 = \left| \frac{\partial \mathbf{r}}{\partial \xi_1} \times \frac{\partial \mathbf{r}}{\partial \xi_2} \right| d\xi_1 d\xi_2 = \sqrt{{n_x}^2 + {n_y}^2 + {n_z}^2}\, d\xi_1 d\xi_2 \,, \quad (4.12)$$

where

$$n_x = \frac{\partial y}{\partial \xi_1}\frac{\partial z}{\partial \xi_2} - \frac{\partial y}{\partial \xi_2}\frac{\partial z}{\partial \xi_1}, \quad n_y = \frac{\partial z}{\partial \xi_1}\frac{\partial x}{\partial \xi_2} - \frac{\partial z}{\partial \xi_2}\frac{\partial x}{\partial \xi_1}, \quad n_z = \frac{\partial x}{\partial \xi_1}\frac{\partial y}{\partial \xi_2} - \frac{\partial x}{\partial \xi_2}\frac{\partial y}{\partial \xi_1} \quad (4.13)$$

where x, y, z are the coordinates of the points $\mathbf{r}$ or $\mathbf{r}'$ and ξ_i define the local curvilinear coordinate system.

This mapping introduces the Jacobian[1] J proportional to the magnitude of the area of the mapped boundary element [10].

4.3.2 Integration of non-singular integrals over the triangle

After discretization of the boundary surface onto boundary elements the integral equation Eq. (4.6) will take the form:

$$c(\mathbf{r})\Phi(\mathbf{r}) + \sum_{j=0}^{M-1} \int_{\Gamma_j} \frac{\partial G(|\mathbf{r}-\mathbf{r}'|)}{\partial n} \Phi(\mathbf{r}')\, d\Gamma_j(\mathbf{r}') =$$

$$= \sum_{j=0}^{M-1} \int_{\Gamma_j} G(|\mathbf{r}-\mathbf{r}'|) \frac{\partial \Phi(\mathbf{r}')}{\partial n} d\Gamma_j(\mathbf{r}') + \int_{\Omega} b\, G(|\mathbf{r}-\mathbf{r}'|) d\Omega(\mathbf{r}') \,, \quad (4.14)$$

where Γ_j is the j-th boundary triangle and M is the number of nodes of the particular element.

[1] Jacobian is shorthand for determinant of Jacoby matrix.

And next can be written in terms of local coordinates ξ_1, ξ_2:

$$
\begin{aligned}
& c(\mathbf{r})\Phi_i(\mathbf{r}) + \sum_{j=0}^{M-1} \Phi_j(\mathbf{r}') \int_{\xi_1=0}^{\xi_1=1} \int_{\xi_2=0}^{\xi_2=1-\xi_1} \frac{\partial G(|\mathbf{r}-\mathbf{r}'|)}{\partial n} N_j J_j(\xi_1,\xi_2) d\xi_1 d\xi_2 = \\
= & \sum_{j=0}^{M-1} \frac{\partial \Phi_j(\mathbf{r}')}{\partial n} \int_{\xi_1=0}^{\xi_1=1} \int_{\xi_2=0}^{\xi_2=1-\xi_1} G(|\mathbf{r}-\mathbf{r}'|) N_j J_j(\xi_1,\xi_2) d\xi_1 d\xi_2 + \\
+ & \int_{\Omega} b\, G(|\mathbf{r}-\mathbf{r}'|) d\Omega(\mathbf{r}')\,,
\end{aligned}
\tag{4.15}
$$

where $N_j = 1$ for the zero order triangle.

The numerical integration rules discussed in this section, perform well when the distance from the source point is relatively far from the element being integrated. Two-dimensional integrals over the triangle can be evaluated by application of the following equation:

$$
\int_0^1 \int_0^{1-\xi_1} f(\xi_1,\xi_2) d\xi_1 d\xi_2 = \sum_{i=0}^{g-1} w_i f(\xi_{1i},\xi_{2i})\,, \tag{4.16}
$$

where g is the total number of Gaussian integration points.

The abscissas and associated weights are given in Table 4.2 [13, 56].

4.4 Second-order interpolation functions

As with many numerical methods, the accuracy of the solution obtained with the BEM, increases with the number of discretization points and/or surface elements at the cost of increased computation time and memory requirements. In this section following [19], it will be shown how the accuracy of the BEM could be improved if second-order interpolation functions for the potential Φ and $\frac{\partial \Phi}{\partial n}$ variation over the surface elements and for the shape of these surface elements are introduced.

Table 4.2: Gauss points and weights over the triangle

n	i	ξ_1	ξ_2	w_i
4	1	1./3.	1./3.	-9./32.
	2	0.6	0.2	25./96.
	3	0.2	0.6	25./96.
	4	0.2	0.2	25./96.
7	1	0.3333333333333333	0.3333333333333333	0.1125000000000000
	2	0.0597158717897698	0.4701420641051151	0.0661970763942531
	3	0.4701420641051151	0.0597158717897698	0.0661970763942531
	4	0.4701420641051151	0.4701420641051151	0.0661970763942531
	5	0.7974269853530873	0.1012865073234563	0.0629695902724136
	6	0.1012865073234563	0.7974269853530873	0.0629695902724136
	7	0.1012865073234563	0.1012865073234563	0.0629695902724136
12	1	0.06308901449150223	0.87382197101699554	0.025422453185103408
	2	0.87382197101699554	0.06308901449150223	0.025422453185103408
	3	0.06308901449150223	0.06308901449150223	0.025422453185103408
	4	0.24928674517091042	0.50142650965817920	0.058393137863189683
	5	0.50142650965817920	0.24928674517091042	0.058393137863189683
	6	0.24928674517091042	0.24928674517091042	0.058393137863189683
	7	0.31035245103378440	0.63650249912139870	0.041425537809186787
	8	0.63650249912139870	0.31035245103378440	0.041425537809186787
	9	0.05314504984481695	0.63650249912139870	0.041425537809186787
	10	0.63650249912139870	0.05314504984481695	0.041425537809186787
	11	0.05314504984481695	0.31035245103378440	0.041425537809186787
	12	0.31035245103378440	0.05314504984481695	0.041425537809186787

4.4.1 Triangular boundary elements

Let focus our attention on six nodes triangular isoparametric element presented in Fig. 4.2. The basis interpolation functions (so called the shape functions) are given by the following formulas:

$$\begin{aligned} N_0(\xi_1,\xi_2) &= -\xi_3(1-2\xi_3), & N_1(\xi_1,\xi_2) &= 4\xi_1\xi_3, \\ N_2(\xi_1,\xi_2) &= -\xi_1(1-2\xi_1), & N_3(\xi_1,\xi_2) &= 4\xi_1\xi_2, \\ N_4(\xi_1,\xi_2) &= -\xi_2(1-2\xi_2), & N_5(\xi_1,\xi_2) &= 4\xi_2\xi_3, \end{aligned} \tag{4.17}$$

where $\xi_3 = 1 - \xi_1 - \xi_2$.

The first derivatives of the standard interpolation functions with respect to

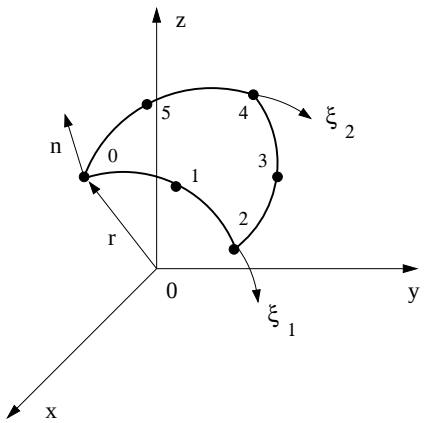


Figure 4.2: Global and local coordinates of the six nodes triangle

the ξ_1 and ξ_2 are given by:

$$\begin{aligned}
\frac{\partial N_0(\xi_1,\xi_2)}{\partial \xi_1} &= 1-4\xi_3, & \frac{\partial N_0(\xi_1,\xi_2)}{\partial \xi_2} &= 1-4\xi_3,\\
\frac{\partial N_1(\xi_1,\xi_2)}{\partial \xi_1} &= 4(\xi_3-\xi_1), & \frac{\partial N_1(\xi_1,\xi_2)}{\partial \xi_2} &= -4\xi_1,\\
\frac{\partial N_2(\xi_1,\xi_2)}{\partial \xi_1} &= -1+4\xi_1, & \frac{\partial N_2(\xi_1,\xi_2)}{\partial \xi_2} &= 0,\\
\frac{\partial N_3(\xi_1,\xi_2)}{\partial \xi_1} &= 4\xi_2, & \frac{\partial N_3(\xi_1,\xi_2)}{\partial \xi_2} &= 4\xi_1,\\
\frac{\partial N_4(\xi_1,\xi_2)}{\partial \xi_1} &= 0, & \frac{\partial N_4(\xi_1,\xi_2)}{\partial \xi_2} &= -1+4\xi_2,\\
\frac{\partial N_5(\xi_1,\xi_2)}{\partial \xi_1} &= -4\xi_2, & \frac{\partial N_5(\xi_1,\xi_2)}{\partial \xi_2} &= 4(\xi_3-\xi_2).
\end{aligned} \tag{4.18}$$

In contrast with the three-node triangular element, the quadratic variation in ξ_1, ξ_2 of the functions $N_0,\ldots,N_5$ allows the creation of curved surfaces with curved edges.

4.4.2 Numerical integration of singular integrals

There are two main approaches for singular integration: integration by regularization and by subtraction and series expansion. Subtraction and series expansion method of calculating the singular integrals is described by Hall

in the monograph [20] and for optics by Ripoll in [39]. The regularization method is described in [11], and is a convenient way to handle higher order elements such as the isoparametric types considered here. However, a detailed comparison of the two methods is outside the scope of this book.

In the following sections the mapping procedures, which are essential for the regularizing method, will be presented. Singular points are denoted by circle surrounding a particular node.

The standard integration rules gives an exact results except when the load point is near to or coincident with one of the nodes of the element. The singularity occurs in this case, so that special treatment of the integration is required. Two of the most effective methods will be described. These are the regularization method and the subtraction/expansion method. As it was for 2D case the exact integration is not possible for the quadratic boundary elements.

In the following figures the mapping procedure, when the singular point (surrounded by a circle) is presented in the nodes 0, 1 or 2. At first, three-dimensional Cartesian space is mapped into a local two-dimensional Cartesian space by interpolation functions.

$$x = \sum_{i=0}^{5} N_i(\xi_1, \xi_2) x_i, \quad y = \sum_{i=0}^{5} N_i(\xi_1, \xi_2) y_i, \quad z = \sum_{i=0}^{5} N_i(\xi_1, \xi_2) z_i. \quad (4.19)$$

If the singularity is in the vertex nodes then the element in local coordinates system ξ_1, ξ_2 is mapped into a standard square (see Fig. 4.3), but when singularity is in one of the mid-side nodes than triangle is divided into two subtriangles and then such triangles are mapped into a standard squares (subelement coordinates)(see Fig. 4.3). The transformation from local elements to subelements coordinates is given by the following expressions:

$$\xi_1 = \sum_{i=0}^{2} M_i(\eta_1, \eta_2) \xi_{1i}, \qquad \xi_2 = \sum_{i=0}^{2} M_i(\eta_1, \eta_2) \xi_{2i}, \quad (4.20)$$

where η_1 and η_2 are the local coordinate of the subelement and the shape functions are given by:

$$M_0(\eta_1, \eta_2) = \frac{(1+\eta_1)(1-\eta_2)}{4}, \quad M_1(\eta_1, \eta_2) = \frac{(1+\eta_1)(1+\eta_2)}{4},$$
$$M_2(\eta_1, \eta_2) = \frac{1-\eta_1}{2}. \quad (4.21)$$

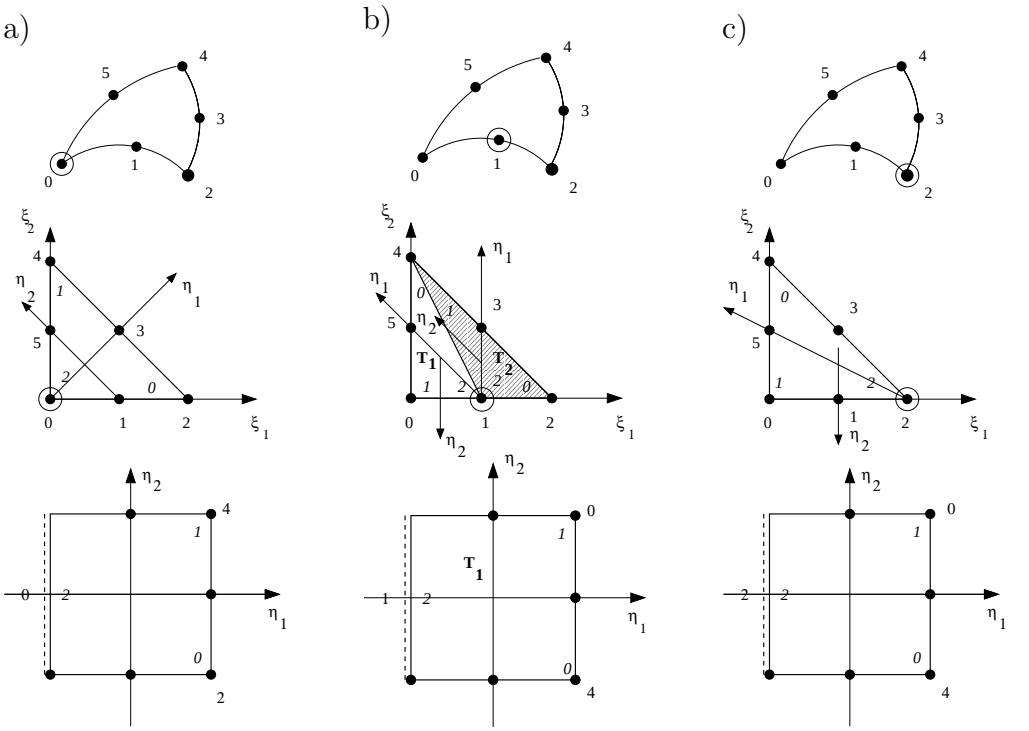


Figure 4.3: The singular point (circled) in: a) the node 0, b) the node 1, c) the node 2

To integrate numerically an arbitrary function over the standard square, the conventional Gaussian quadrature scheme can be easily carried out there [11]. The regularization method transforms triangular domain into a square, in that way introduces a further Jacobian which cancels out the singularity of the integrand.

Node 0

Based on Eq. (4.20) the transformation is:

$$\xi_1 = \frac{(1+\eta_1)(1-\eta_2)}{4}, \qquad \xi_2 = \frac{(1+\eta_1)(1+\eta_2)}{4}. \tag{4.22}$$

A regularizing Jacobian $J_r(\eta_1, \eta_2)$, will be associated with the change of variables which is given by:

$$J_r = \begin{vmatrix} \frac{\partial \xi_1}{\partial \eta_1} & \frac{\partial \xi_1}{\partial \eta_2} \\ \frac{\partial \xi_2}{\partial \eta_1} & \frac{\partial \xi_2}{\partial \eta_2} \end{vmatrix} = \frac{1+\eta_1}{8}. \tag{4.23}$$

Node 1

For subtriangle T_1:

$$\xi_1 = \frac{1-\eta_1}{4}, \qquad \xi_2 = \frac{(1+\eta_1)(1-\eta_2)}{4}, \qquad J_{r_{T_1}} = \frac{1+\eta_1}{16}. \tag{4.24}$$

For subtriangle T_2:

$$\xi_1 = \frac{2-(1+\eta_1)\eta_2}{4}, \qquad \xi_2 = \frac{(1+\eta_1)(1+\eta_2)}{4}, \qquad J_{r_{T_2}} = \frac{1+\eta_1}{16}. \tag{4.25}$$

Node 2

$$\xi_1 = \frac{1-\eta_1}{2}, \qquad \xi_2 = \frac{(1+\eta_1)(1-\eta_2)}{4}, \qquad J_r = \frac{1+\eta_1}{8}. \tag{4.26}$$

The mapping procedure when the singular point is in the nodes 3, 4 or 5.

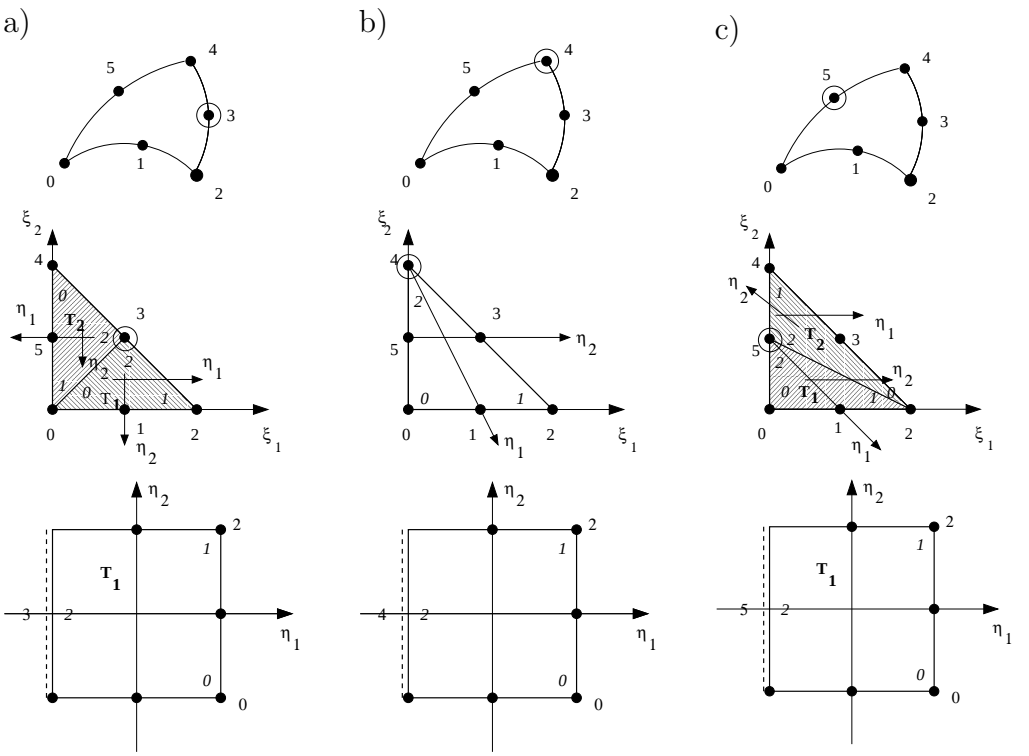


Figure 4.4: The singular point (circled) in: a) the node 3, b) the node 4, c) the node 5

Node 3

For subtriangle T_1:

$$\xi_1 = \frac{2 + (1 + \eta_1)\eta_2}{4}, \quad \xi_2 = \frac{1 - \eta_1}{4}, \qquad J_{r_{T_1}} = \frac{1 + \eta_1}{16}. \tag{4.27}$$

For subtriangle T_2:

$$\xi_1 = \frac{1 - \eta_1}{4}, \qquad \xi_2 = \frac{2 - (1 + \eta_1)\eta_2}{4}, \quad J_{r_{T_2}} = \frac{1 + \eta_1}{16}. \tag{4.28}$$

Node 4

$$\xi_1 = \frac{(1 + \eta_1)(1 + \eta_2)}{4}, \quad \xi_2 = \frac{1 - \eta_1}{2}, \qquad J_r = \frac{1 + \eta_1}{8}. \tag{4.29}$$

Node 5

For subtriangle T_1:

$$\xi_1 = \frac{(1 + \eta_1)(1 + \eta_2)}{4}, \quad \xi_2 = \frac{1 - \eta_1}{4}, \qquad J_{r_{T_1}} = \frac{1 + \eta_1}{16}. \tag{4.30}$$

For subtriangle T_2:

$$\xi_1 = \frac{(1 + \eta_1)(1 - \eta_2)}{4}, \quad \xi_2 = \frac{2 + (1 + \eta_1)\eta_2}{4}, \quad J_{r_{T_2}} = \frac{1 + \eta_1}{16}. \tag{4.31}$$

In the Fig. 4.5 it is illustrated how splitting and mapping procedure concentrates the Gaussian points around the singularity node.

4.4.3 More precise numerical integration of singular integrals

For the isoparametric triangular quadratic element the singular integration procedure was already presented in section 4.4.2 for Laplace equation. In case of diffusion equation the situation is more complicated, because in order

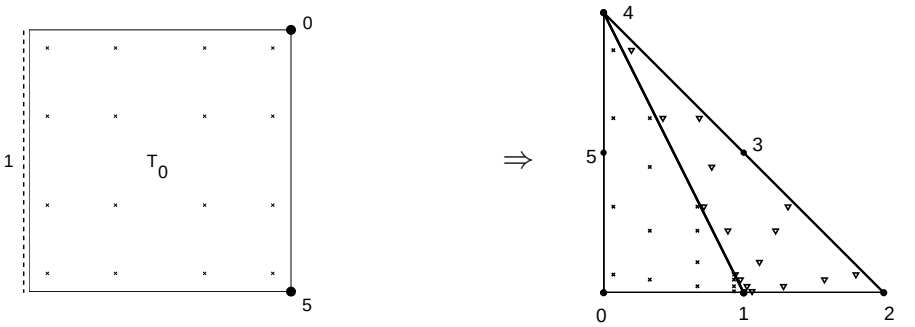


Figure 4.5: The Gaussian quadrature points used on the square mapped back to the flat triangle space (ξ_1, ξ_2) when the singulaity is located in the node no. 1

to achieve the same level of accuracy, better integration procedure of singular integrands is needed.

That is why a new mapping procedure will be shown. This procedure is similar to the previous one only for the first two nodes. The rest nodes can be treated in a similar way.

To gain sufficient precision of calculations, each element containing a singularity is divided not into two sub-elements (see Fig. 4.3) but into four sub-triangles T_0, T_1, T_2 and T_3. Those sub-elements containing the singular point, for example T_0 in Fig. 4.6 and sub-triangles T_0, T_1 and T_2 in the same figure are mapped again into a square (coordinates η_1, η_2). Sub-triangles which do not contain the singular point are numerically integrated using for example the seven point Gaussian quadrature rule for triangles (see Table 4.2) [13, 55, 56].

The second transformation introduces Jacobian J_r or $J_{r_{T_0}}$, $J_{r_{T_1}}$, $J_{r_{T_2}}$ (see Fig. 4.6), according to the singularity position. Fig. 4.7 illustrates how the splitting and mapping procedures concentrate the Gaussian quadrature points around the singularity node. The mathematical expressions are given below.

The isoparametric triangular element from the x, y, z global coordinates system is mapped to the local coordinates system ξ_1, ξ_2 by Eq. (4.19).

First let consider the singularity at node 0 and than at node 1:

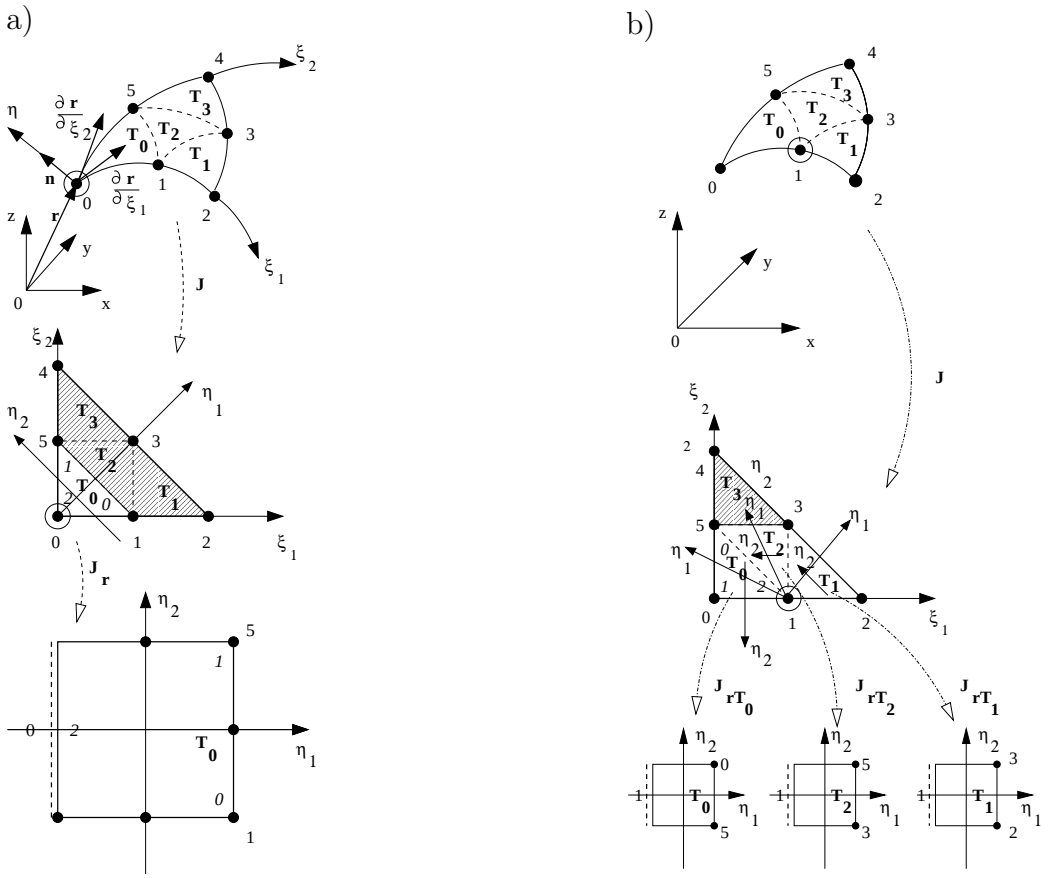


Figure 4.6: Isoparametric triangle subdivision in case when the singular point is located in: a) the node 0, b) the node 1

Node 0 (Fig. 4.6)

The transformation from a sub-triangle T_1 to a standard square is:

$$\xi_1 = \frac{(1+\eta_1)(1-\eta_2)}{8}, \qquad \xi_2 = \frac{(1+\eta_1)(1+\eta_2)}{8}. \tag{4.32}$$

The Jacobian $J_r(\eta_1, \eta_2)$, will be associated with the change of variables which is given by:

$$J_r = \begin{vmatrix} \frac{\partial \xi_1}{\partial \eta_1} & \frac{\partial \xi_1}{\partial \eta_2} \\ \frac{\partial \xi_2}{\partial \eta_1} & \frac{\partial \xi_2}{\partial \eta_2} \end{vmatrix} = \frac{1+\eta_1}{32}. \tag{4.33}$$

Figure 4.7: The Gaussian point concentration around the singularity located at the node 1

Node 1 (Fig. 4.6)

For sub-triangle T_1:

$$\xi_1 = \frac{1-\eta_1}{4}, \qquad \xi_2 = \frac{(1+\eta_1)(1-\eta_2)}{8}, \quad J_{r_{T_1}} = \frac{1+\eta_1}{32}. \quad (4.34)$$

For sub-triangle T_2:

$$\xi_1 = \frac{(1+\eta_1)(3-\eta_2)}{8} + \frac{1-\eta_1}{4}, \quad \xi_2 = \frac{(1+\eta_1)(1+\eta_2)}{8}, \quad J_{r_{T_2}} = \frac{1+\eta_1}{32}. \quad (4.35)$$

For sub-triangle T_3:

$$\xi_1 = \frac{(1+\eta_1)(1-\eta_2)}{8} + \frac{1-\eta_1}{4}, \quad \xi_2 = \frac{1+\eta_1}{4}, \qquad J_{r_{T_3}} = \frac{1+\eta_1}{32}. \quad (4.36)$$

The similar procedure can be used for the next nodes – $2, 3, 4$ and 5, using the transformation from a local coordinates system of the particular element ξ_1, ξ_2 to sub-element coordinates η_1, η_2 given by Eq. (4.20) and Eq. (4.21).

4.4.4 Quadrilateral boundary elements

Let consider eight nodes quadrilateral isoparametric element presented in Fig. 4.8.

The shape functions are given by the following formulas:

$$
\begin{aligned}
N_0(\xi_1,\xi_2) &= -(1-\xi_1)(1-\xi_2)(1+\xi_1+\xi_2)/4, & N_1(\xi_1,\xi_2) &= (1-\xi_1^2)(1-\xi_2)/2,\\
N_2(\xi_1,\xi_2) &= -(1+\xi_1)(1-\xi_2)(1-\xi_1+\xi_2)/4, & N_3(\xi_1,\xi_2) &= (1+\xi_1)(1-\xi_2^2)/2,\\
N_4(\xi_1,\xi_2) &= -(1+\xi_1)(1+\xi_2)(1-\xi_1-\xi_2)/4, & N_5(\xi_1,\xi_2) &= (1-\xi_1^2)(1+\xi_2)/2,\\
N_6(\xi_1,\xi_2) &= -(1-\xi_1)(1+\xi_2)(1+\xi_1-\xi_2)/4, & N_7(\xi_1,\xi_2) &= (1-\xi_1)(1-\xi_2^2)/2.
\end{aligned}
$$

The first derivatives of the standard interpolation functions with respect to the ξ_1 and ξ_2 are given by:

$$
\begin{aligned}
\frac{\partial N_0(\xi_1,\xi_2)}{\partial \xi_1} &= (1-\xi_2)(2\xi_1+\xi_2)/4, & \frac{\partial N_1(\xi_1,\xi_2)}{\partial \xi_1} &= -\xi_1(1-\xi_2),\\
\frac{\partial N_2(\xi_1,\xi_2)}{\partial \xi_1} &= (1-\xi_2)(2\xi_1-\xi_2)/4, & \frac{\partial N_3(\xi_1,\xi_2)}{\partial \xi_1} &= (1-\xi_2^2)/2,\\
\frac{\partial N_4(\xi_1,\xi_2)}{\partial \xi_1} &= (1+\xi_2)(2\xi_1+\xi_2)/4, & \frac{\partial N_5(\xi_1,\xi_2)}{\partial \xi_1} &= -\xi_1(1+\xi_2),\\
\frac{\partial N_6(\xi_1,\xi_2)}{\partial \xi_1} &= (1+\xi_2)(2\xi_1-\xi_2)/4, & \frac{\partial N_7(\xi_1,\xi_2)}{\partial \xi_1} &= -(1-\xi_2^2)/2,
\end{aligned}
\tag{4.37}
$$

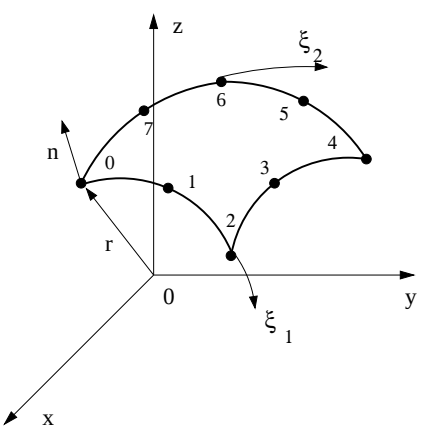


Figure 4.8: Local coordinates of the quadrilateral boundary element

$$\begin{aligned}
\frac{\partial N_0(\xi_1,\xi_2)}{\partial \xi_2} &= (1-\xi_1)(2\xi_2+\xi_1)/4, & \frac{\partial N_1(\xi_1,\xi_2)}{\partial \xi_2} &= -(1-\xi_1^2)/2,\\
\frac{\partial N_2(\xi_1,\xi_2)}{\partial \xi_2} &= (1+\xi_1)(2\xi_2-\xi_1)/4, & \frac{\partial N_3(\xi_1,\xi_2)}{\partial \xi_2} &= -\xi_2(1+\xi_1),\\
\frac{\partial N_4(\xi_1,\xi_2)}{\partial \xi_2} &= (1+\xi_1)(2\xi_2+\xi_1)/4, & \frac{\partial N_5(\xi_1,\xi_2)}{\partial \xi_2} &= (1-\xi_1^2)/2,\\
\frac{\partial N_6(\xi_1,\xi_2)}{\partial \xi_2} &= (1-\xi_1)(2\xi_2-\xi_1)/4, & \frac{\partial N_7(\xi_1,\xi_2)}{\partial \xi_2} &= -\xi_2(1-\xi_1).
\end{aligned} \tag{4.38}$$

4.4.5 Integration of non-singular integrals over the square

Three-dimensional boundary element analysis involves the integration of certain functions over the surface of boundary elements. For quadratic elements we can characterize the generic integration task by:

$$I = \int_{-1}^{+1}\int_{-1}^{+1} f(\xi_1,\xi_2)d\xi_1 d\xi_2 = \sum_{i=1}^{n}\left(\sum_{j=1}^{n} f(\xi_{1j},\xi_{2i})w_j\right)w_i\,. \tag{4.39}$$

Values of the Gaussian integration points and weights coefficients are listed in the Table 4.3.

Table 4.3: Gauss points and weights

n	i	ξ_1	ξ_2	w_i
3	1	-0.77459666924148337704	-0.77459666924148337704	0.55555555555555555556
	2	0.0	0.0	0.88888888888888888889
	3	+0.77459666924148337704	+0.77459666924148337704	0.55555555555555555556
4	1	-0.86113631159495257522	-0.86113631159495257522	0.34785484513745385737
	2	-0.33998104358485626480	-0.33998104358485626480	0.65214515486254614263
	3	+0.33998104358485626480	+0.33998104358485626480	0.65214515486254614263
	4	+0.86113631159495257522	+0.86113631159495257522	0.34785484513745385737
6	1	-0.93246951420315202781	-0.93246951420315202781	0.17132449237917034504
	2	-0.66120938646626451366	-0.66120938646626451366	0.36076157304813860757
	3	-0.23861918608319690863	-0.23861918608319690863	0.46791393457269104739
	4	+0.23861918608319690863	+0.23861918608319690863	0.46791393457269104739
	5	+0.66120938646626451366	+0.66120938646626451366	0.36076157304813860757
	6	+0.93246951420315202781	+0.93246951420315202781	0.17132449237917034504

4.4.6 Integration of singular integrals over the square

The standard integration rules (see Table 4.3) give an exact result except when the load point is near to or coincident with one of the nodes of the element. The singularity occurs in this case, so a special treatment of the integration is required. Two of the most effective methods will be described. These are the regularization method and the subtraction/expansion method [49]. As it was for 2D case, the exact integration is not possible for the quadratic boundary elements.

In the following figures the mapping procedure, when the singular point (surrounded by a circle) is in the nodes 0, 1 or 2, is presented. At first, three-dimensional Cartesian space is mapped into a local two-dimensional Cartesian space by interpolation functions.

$$x = \sum_{i=0}^{7} N_i(\xi_1, \xi_2) x_i, \quad y = \sum_{i=0}^{7} N_i(\xi_1, \xi_2) y_i, \quad z = \sum_{i=0}^{7} N_i(\xi_1, \xi_2) z_i. \tag{4.40}$$

Then the standard square element in local coordinates system ξ_1, ξ_2 is divided into two or three triangular subelements, depending on the location of the singular point, as it is shown in Fig. 4.9. Lastly, those sub-triangles are mapped into the standard square again, so the conventional Gaussian quadrature scheme can be easily carried out there.

The regularization method using transformation of a triangular domain into a square and in that way introduces a further Jacobian which cancels out the singularity of the integrand.

Node 0

For triangle T_1:

$$\xi_1 = \eta_1, \quad \xi_2 = \frac{-1 + \eta_1 + (1 + \eta_1)\eta_2}{2}. \tag{4.41}$$

For triangle T_2:

$$\xi_1 = \frac{-1 + \eta_1 - (1 + \eta_1)\eta_2}{2}, \quad \xi_2 = \eta_1. \tag{4.42}$$

A regularizing Jacobian $J_{r_{T_1}} = J_{r_{T_2}} = J_r(\eta_1, \eta_2)$, is associated with the

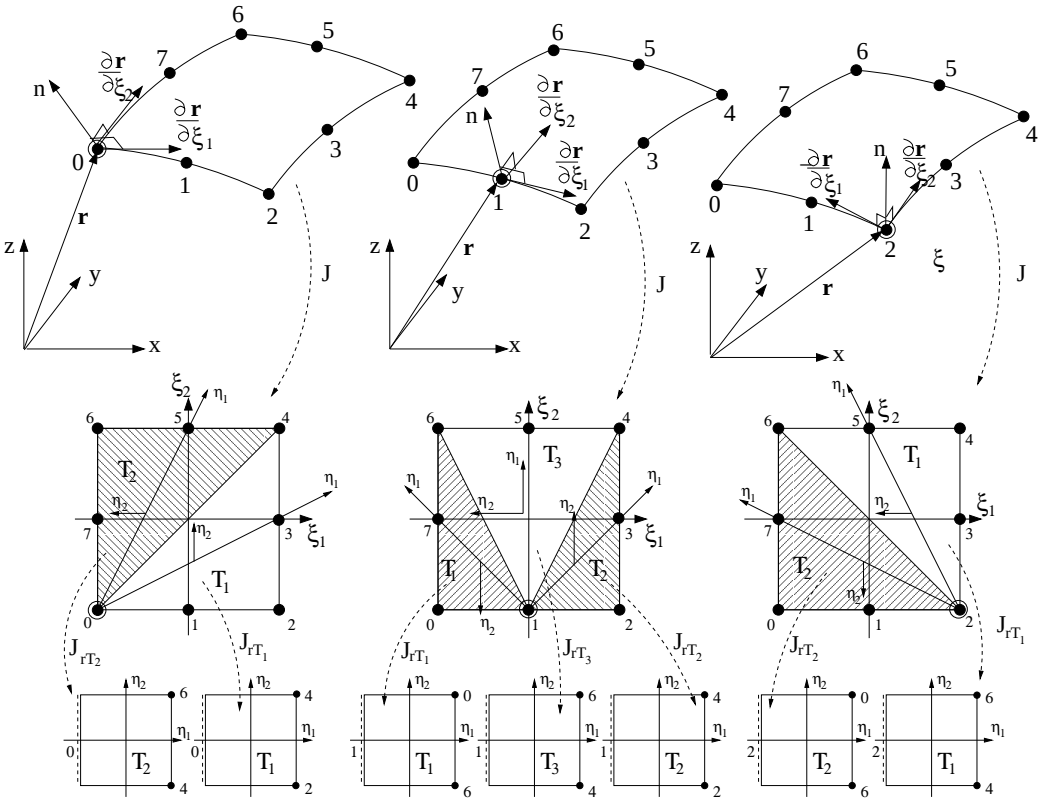


Figure 4.9: The singular point in the node 0 (left), the singular point in the node 1 (middle) and the singular point in the node 2 (right)

change of variables and is given by

$$J_r = \left| \begin{array}{cc} \frac{\partial \xi_1}{\partial \eta_1} & \frac{\partial \xi_1}{\partial \eta_2} \\ \frac{\partial \xi_2}{\partial \eta_1} & \frac{\partial \xi_2}{\partial \eta_2} \end{array} \right| = \frac{1+\eta_1}{2}. \tag{4.43}$$

Node 1

For triangle T_1:

$$\xi_1 = -\frac{1+\eta_1}{2}, \qquad \xi_2 = \frac{-1+\eta_1-(1+\eta_1)\eta_2}{2}, \qquad J_{r_{T_1}} = \frac{1+\eta_1}{4}. \tag{4.44}$$

For triangle T_2:

$$\xi_1 = \frac{1+\eta_1}{2}, \qquad \xi_2 = \frac{-1+\eta_1+(1+\eta_1)\eta_2}{2}, \qquad J_{r_{T_2}} = \frac{1+\eta_1}{4}. \tag{4.45}$$

For triangle T_3:

$$\xi_1 = -\frac{(1+\eta_1)\eta_2}{2}, \quad \xi_2 = \eta_1, \qquad J_{r_{T_3}} = \frac{1+\eta_1}{2}. \tag{4.46}$$

Node 2

For triangle T_1:

$$\xi_1 = \frac{1-\eta_1-(1+\eta_1)\eta_2}{2}, \qquad \xi_2 = \eta_1. \tag{4.47}$$

For triangle T_2:

$$\xi_1 = -\eta_1, \qquad \xi_2 = \frac{-1+\eta_1-(1+\eta_1)\eta_2}{2}. \tag{4.48}$$

As it was in case of the node 0 the regularizing Jacobian $J_r(\eta_1, \eta_2)$ for triangle T_1 and for the triangle T_2 is the same and is given by:

$$J_r = \frac{1+\eta_1}{2}. \tag{4.49}$$

The mapping procedure when the singular point is placed in the nodes 3, 4 or 5 are presented below.

Node 3

For triangle T_1:

$$\xi_1 = \frac{1-\eta_1+(1+\eta_1)\eta_2}{2}, \quad \xi_2 = -\frac{1+\eta_1}{2}, \quad J_{r_{T_1}} = \frac{1+\eta_1}{4}. \tag{4.50}$$

For triangle T_2:

$$\xi_1 = \frac{1-\eta_1-(1+\eta_1)\eta_2}{2}, \quad \xi_2 = \frac{1+\eta_1}{2}, \qquad J_{r_{T_2}} = \frac{1+\eta_1}{4}. \tag{4.51}$$

For triangle T_3:

$$\xi_1 = -\eta_1, \qquad \xi_2 = -\frac{(1+\eta_1)\eta_2}{2}, \quad J_{r_{T_3}} = \frac{1+\eta_1}{2}. \tag{4.52}$$

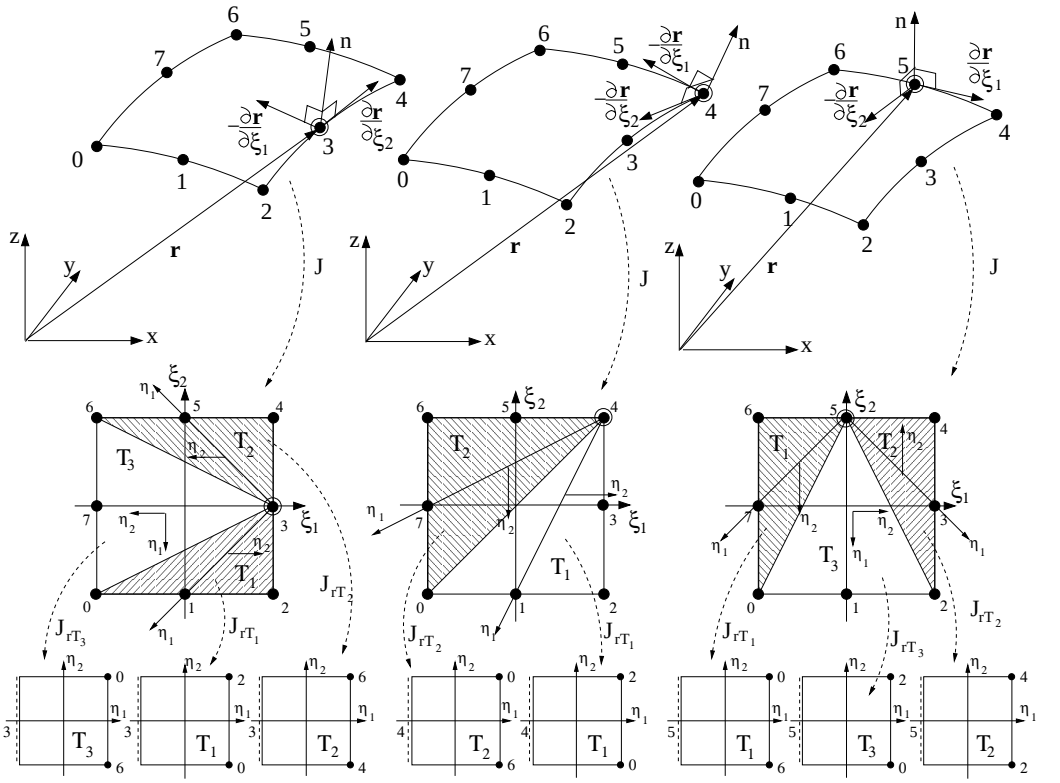


Figure 4.10: Singularity in the node 3 (left), in the node 4 (middle) and in the node 5 (right)

Node 4

For triangle T_1:

$$\xi_1 = \frac{1 - \eta_1 + (1 + \eta_1)\eta_2}{2}, \qquad \xi_2 = -\eta_1. \tag{4.53}$$

For triangle T_2:

$$\xi_1 = -\eta_1, \qquad \xi_2 = \frac{1 - \eta_1 - (1 + \eta_1)\eta_2}{2}. \tag{4.54}$$

A regularizing Jacobian $J_r(\eta_1, \eta_2)$ for both sub-triangles is given by:

$$J_r = \frac{1 + \eta_1}{2}. \tag{4.55}$$

Node 5

For triangle T_1:

$$\xi_1 = -\frac{1+\eta_1}{2}, \qquad \xi_2 = \frac{1-\eta_1-(1+\eta_1)\eta_2}{2}, \qquad J_{r_{T_1}} = \frac{1+\eta_1}{4}. \tag{4.56}$$

For triangle T_2:

$$\xi_1 = \frac{1+\eta_1}{2}, \qquad \xi_2 = \frac{1-\eta_1+(1+\eta_1)\eta_2}{2}, \qquad J_{r_{T_2}} = \frac{1+\eta_1}{4}. \tag{4.57}$$

For triangle T_3:

$$\xi_1 = \frac{(1+\eta_1)\eta_2}{2}, \qquad \xi_2 = -\eta_1, \qquad J_{r_{T_3}} = \frac{1+\eta_1}{2}. \tag{4.58}$$

Node 6

For triangle T_1:

$$\xi_1 = \frac{-1+\eta_1+(1+\eta_1)\eta_2}{2}, \qquad \xi_2 = -\eta_1. \tag{4.59}$$

For triangle T_2:

$$\xi_1 = \eta_1, \qquad \xi_2 = \frac{1-\eta_1+(1+\eta_1)\eta_2}{2}. \tag{4.60}$$

A regularizing Jacobian $J_r(\eta_1, \eta_2)$, is associated with the change of variables and is given by:

$$J_r = \frac{1+\eta_1}{2}. \tag{4.61}$$

Node 7

For triangle T_1:

$$\xi_1 = \frac{-1+\eta_1+(1+\eta_1)\eta_2}{2}, \qquad \xi_2 = -\frac{1+\eta_1}{2}, \qquad J_{r_{T_1}} = \frac{1+\eta_1}{4}. \tag{4.62}$$

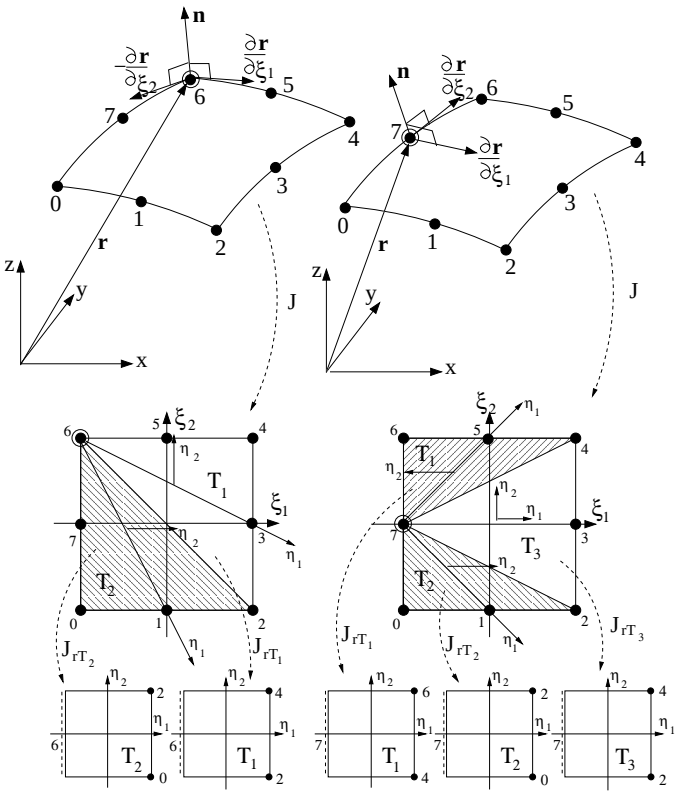


Figure 4.11: Singularity in the node 6 and in the node 7

For triangle T_2:

$$\xi_1 = \frac{-1+\eta_1-(1+\eta_1)\eta_2}{2}, \qquad \xi_2 = \frac{1+\eta_1}{2}, \qquad J_{r_{T_2}} = \frac{1+\eta_1}{4}. \tag{4.63}$$

For triangle T_3:

$$\xi_1 = \eta_1, \qquad \xi_2 = \frac{(1+\eta_1)\eta_2}{2}, \quad J_{r_{T_3}} = \frac{1+\eta_1}{2}. \tag{4.64}$$

In the Fig. 4.12a and in the Fig. 4.12b it is illustrated how splitting and mapping procedure concentrates the Gaussian points around the singularity nodes in first two cases.

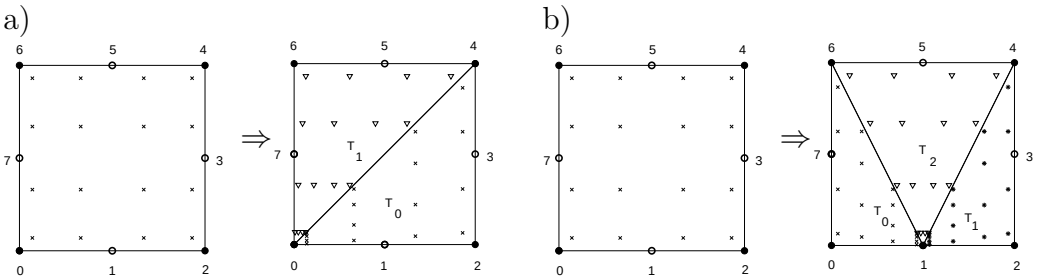


Figure 4.12: The Gaussian points concentration around: a) the singular node 0, b) the singular node 1

4.5 Treatment of Boundary Conditions

When the boundary conditions are imposed on the whole boundary, the number of unknowns is reduced from $2N$ to N. We can distinguish the following cases:

1. Dirichlet boundary conditions,
2. Neumann boundary conditions,
3. Robin boundary conditions,
4. Mixed boundary conditions.

All of those cases will be described in the following sections.

4.5.1 Dirichlet boundary conditions

For Dirichlet boundary conditions vector $\mathbf{\Phi} = \mathbf{\Phi}_D$ containing discrete values of photon density function $\mathbf{\Phi}$, is specified and the vector $\frac{\partial \mathbf{\Phi}}{\partial n}$ is unknown. Then the matrix form of integral equation $\mathbf{A\Phi} = \mathbf{B}\frac{\partial \mathbf{\Phi}}{\partial n} + \mathbf{q}$ can be rearranged to the following form:

$$\mathbf{B}\frac{\partial \mathbf{\Phi}}{\partial n} = \mathbf{A\Phi}_D - \mathbf{q} = \mathbf{C}_D \,. \tag{4.65}$$

4.5.2 Neumann boundary conditions

In this case the vector $\frac{\partial \mathbf{\Phi}}{\partial n} = (\frac{\partial \mathbf{\Phi}}{\partial n})_N$ is known and vector $\mathbf{\Phi}$ is unknown:

$$\mathbf{A\Phi} = (\mathbf{B}\frac{\partial \mathbf{\Phi}}{\partial n})_N + \mathbf{q} = \mathbf{C}_N \,. \tag{4.66}$$

4.5.3 Robin boundary conditions

Robin boundary conditions (see Eq. (4.2)) express linear dependency of both vectors $\mathbf{\Phi}$ and $\frac{\partial \mathbf{\Phi}}{\partial n}$. So the procedure is as follows: define $\frac{\partial \mathbf{\Phi}}{\partial n}$ as a function of vector $\mathbf{\Phi}$ and then introduce to matrix form of integral equation:

$$(\mathbf{A} + \frac{1}{2\alpha D}\mathbf{B})\mathbf{\Phi} = \mathbf{q} = \mathbf{C}_R \,. \tag{4.67}$$

4.5.4 Mixed boundary conditions

When on some part of the boundary surface the Dirichlet boundary conditions and on the rest of the surface the Robin boundary conditions are imposed, then the rearranging process is a little more complicated. Let assume that first m_1 rows of vector $\mathbf{\Phi}$ are the Dirichlet boundary conditions and the remain part $m - m_1$ are of the Robin type.

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{\Phi}_D \\ \mathbf{\Phi} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix} \begin{bmatrix} \frac{\partial \mathbf{\Phi}}{\partial n} \\ -\frac{1}{2\alpha D}\mathbf{\Phi} \end{bmatrix} + \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \end{bmatrix} . \tag{4.68}$$

After some mathematics final system of equations has the following form:

$$\begin{bmatrix} (\mathbf{A}_{12} + \frac{1}{2\alpha D}\mathbf{B}_{12}) & -\mathbf{B}_{11} \\ (\mathbf{A}_{22} + \frac{1}{2\alpha D}\mathbf{B}_{22}) & -\mathbf{B}_{21} \end{bmatrix} \begin{bmatrix} \mathbf{\Phi} \\ \frac{\partial \mathbf{\Phi}}{\partial n} \end{bmatrix} = \begin{bmatrix} -\mathbf{A}_{11}\mathbf{\Phi}_D + \mathbf{q}_1 \\ -\mathbf{A}_{21}\mathbf{\Phi}_D + \mathbf{q}_2 \end{bmatrix} . \tag{4.69}$$

4.6 Non-homogeneity

Basically the BEM method is designed for the solution in the homogeneous areas. However very often we have to find out the solution inside the regions which are spatially homogeneous. Then each region is considered separately

and the solution is "sewed" (see Eq. (4.70)) on an interface or on the interfaces if we have more than two subregions. Such approach is general and describes 2D and 3D problems. Let introduce the superscripts (i) denoting the i-th region.

The photon density and the current of photons along the nodes on the interface must therefore satisfy the following conditions:

$$\Phi_{\Gamma_i}^{(i-1)} = \Phi_{\Gamma_i}^{(i)}, \qquad D^{(i-1)} \frac{\partial \Phi^{(i-1)}}{\partial n}\bigg|_{\Gamma_i} = - D^{(i)} \frac{\partial \Phi^{(i)}}{\partial n}\bigg|_{\Gamma_i}. \quad (4.70)$$

The system of equations describing the solution in the structure consisting of n subregions is:

$$\begin{aligned} c(\mathbf{r})\Phi^{(1)}(\mathbf{r}) + \int_{\Gamma} \frac{\partial G^{(1)}(|\mathbf{r}-\mathbf{r}'|,\omega)}{\partial n} \Phi^{(1)}(\mathbf{r}') d\Gamma(\mathbf{r}') = \\ = \int_{\Gamma} G^{(1)}(|\mathbf{r}-\mathbf{r}'|,\omega) \frac{\partial \Phi^{(1)}(\mathbf{r}')}{\partial n} d\Gamma(\mathbf{r}') - \sum_{s=1}^{ns} Q_s G^{(1)}(|\mathbf{r}_s - \mathbf{r}|,\omega), \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ c(\mathbf{r})\Phi^{(n-1)}(\mathbf{r}) + \int_{\Gamma} \frac{\partial G^{(n-1)}(|\mathbf{r}-\mathbf{r}'|,\omega)}{\partial n} \Phi^{(n-1)}(\mathbf{r}') d\Gamma(\mathbf{r}') = \\ = \int_{\Gamma} G^{(n-1)}(|\mathbf{r}-\mathbf{r}'|,\omega) \frac{\partial \Phi^{(n-1)}(\mathbf{r}')}{\partial n} d\Gamma(\mathbf{r}'), \\ c(\mathbf{r})\Phi^{(n)}(\mathbf{r}) + \int_{\Gamma} \frac{\partial G^{(n)}(|\mathbf{r}-\mathbf{r}'|,\omega)}{\partial n} \Phi^{(n)}(\mathbf{r}') d\Gamma(\mathbf{r}') = \\ = \int_{\Gamma} G^{(n)}(|\mathbf{r}-\mathbf{r}'|,\omega) \frac{\partial \Phi^{(n)}(\mathbf{r}')}{\partial n} d\Gamma(\mathbf{r}'). \end{aligned} \quad (4.71)$$

If we limit ourselves to regions consisted of homogeneous subregions (layers) enclosed one inside the others (see Fig. 4.13), then the boundary might be divided to external Γ_i and internal one Γ_{i+1}. The set of integral equations

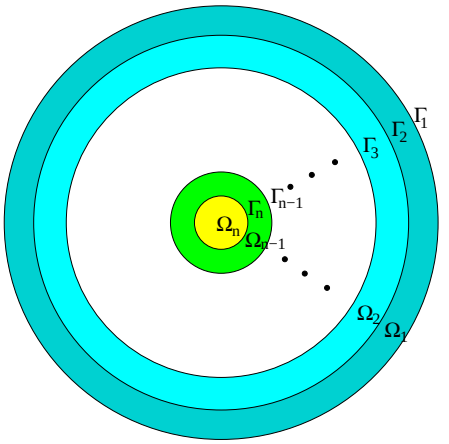


Figure 4.13: Cross-section of the region under consideration as a set of concentric spheres

(4.71) may be presented in the matrix form:

$$\begin{bmatrix} \mathbf{A}_{11}^{(1)} & \mathbf{A}_{12}^{(1)} & \dots & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{21}^{(1)} & \mathbf{A}_{22}^{(1)} & \dots & \dots & \mathbf{0} & \mathbf{0} \\ \hline \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \hline \mathbf{0} & \mathbf{0} & \dots & \mathbf{A}_{11}^{(n-1)} & \mathbf{A}_{12}^{(n-1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{A}_{21}^{(n-1)} & \mathbf{A}_{22}^{(n-1)} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{A}_{11}^{(n)} \end{bmatrix} \begin{bmatrix} \mathbf{\Phi}_{\Gamma_1}^{(1)} \\ \mathbf{\Phi}_{\Gamma_2}^{(1)} \\ \hline \vdots \\ \hline \mathbf{\Phi}_{\Gamma_{n-1}}^{(n-1)} \\ \mathbf{\Phi}_{\Gamma_n}^{(n-1)} \\ \hline \mathbf{\Phi}_{\Gamma_n}^{(n)} \end{bmatrix} =$$

$$= \begin{bmatrix} \mathbf{B}_{11}^{(1)} & \mathbf{B}_{12}^{(1)} & \dots & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{B}_{21}^{(1)} & \mathbf{B}_{22}^{(1)} & \dots & \dots & \mathbf{0} & \mathbf{0} \\ \hline \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \hline \mathbf{0} & \mathbf{0} & \dots & \mathbf{B}_{11}^{(n-1)} & \mathbf{B}_{12}^{(n-1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{B}_{21}^{(n-1)} & \mathbf{B}_{22}^{(n-1)} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{B}_{11}^{(n)} \end{bmatrix} \begin{bmatrix} \mathbf{J}_{\mathbf{n}\Gamma_1}^{(1)} \\ \mathbf{J}_{\mathbf{n}\Gamma_2}^{(1)} \\ \hline \vdots \\ \hline \mathbf{J}_{\mathbf{n}\Gamma_{n-1}}^{(n-1)} \\ \mathbf{J}_{\mathbf{n}\Gamma_n}^{(n-1)} \\ \hline \mathbf{J}_{\mathbf{n}\Gamma_n}^{(n)} \end{bmatrix} + \begin{bmatrix} \mathbf{q_1}^{(1)} \\ \mathbf{q_2}^{(1)} \\ \hline \vdots \\ \hline \mathbf{0} \\ \mathbf{0} \\ \hline \mathbf{0} \end{bmatrix}, \tag{4.72}$$

where $\mathbf{J}_{\mathbf{n}\Gamma_i=\Gamma_1\cup\Gamma_2}^{(i)}$ means $\left.\frac{\partial \mathbf{\Phi}^{(i)}}{\partial n}\right|_{\Gamma_i=\Gamma_1\cup\Gamma_2} = m_R \mathbf{\Phi}_{\Gamma_i=\Gamma_1\cup\Gamma_2}^{(i)}$. Usually in DOT vector $\mathbf{n}_R = 0$ for Robin boundary conditions.

Horizontal and vertical lines separate coefficients of the matrices of the same subregions. The final subregion contrary to the previous subregions poses

one sphere boundary only.

$$\left[\begin{array}{cc|c|cc|c} \mathbf{A}_{11}^{(1)} & \mathbf{A}_{12}^{(1)} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{21}^{(1)} & \mathbf{A}_{22}^{(1)} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \hline \mathbf{0} & \mathbf{0} & \dots & \mathbf{A}_{11}^{(n-1)} & \mathbf{A}_{12}^{(n-1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{A}_{21}^{(n-1)} & \mathbf{A}_{22}^{(n-1)} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{A}_{11}^{(n)} \end{array}\right] \left[\begin{array}{c} \mathbf{\Phi}_{\Gamma_1}^{(1)} \\ \mathbf{\Phi}_{\Gamma_2}^{(1)} \\ \hline \vdots \\ \hline \mathbf{\Phi}_{\Gamma_{n-1}}^{(n-1)} \\ \mathbf{\Phi}_{\Gamma_n}^{(n-1)} \\ \hline \mathbf{\Phi}_{\Gamma_n}^{(n)} \end{array}\right] =$$

$$= \left[\begin{array}{cc|c|cc|c} \mathbf{B}_{11}^{\prime(1)} & \mathbf{B}_{12}^{(1)} & \dots & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{B}_{21}^{\prime(1)} & \mathbf{B}_{22}^{(1)} & \dots & \dots & \mathbf{0} & \mathbf{0} \\ \hline \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \hline \mathbf{0} & \mathbf{0} & \dots & \mathbf{B}_{11}^{(n-1)} & \mathbf{B}_{12}^{(n-1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{B}_{21}^{(n-1)} & \mathbf{B}_{22}^{(n-1)} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{B}_{11}^{(n)} \end{array}\right] \left[\begin{array}{c} \mathbf{\Phi}_{\Gamma_1}^{(1)} \\ \boldsymbol{J}_{\boldsymbol{n}\Gamma_2}^{(1)} \\ \hline \vdots \\ \hline \mathbf{J}_{\mathbf{n}\Gamma_{n-1}}^{(n-1)} \\ \mathbf{J}_{\mathbf{n}\Gamma_n}^{(n-1)} \\ \hline \mathbf{J}_{\mathbf{n}\Gamma_n}^{(n)} \end{array}\right] + \left[\begin{array}{c} \mathbf{q_1}^{(1)} \\ \mathbf{q_2}^{(1)} \\ \hline \vdots \\ \hline \mathbf{0} \\ \mathbf{0} \\ \hline \mathbf{0} \end{array}\right], \tag{4.73}$$

where $\mathbf{B}_{11}^{\prime(1)}$ and $\mathbf{B}_{21}^{\prime(1)}$ are modified by Robin boundary conditions.

Both side of equations (4.73) poses unknowns. That is why we need to rearrange it using interface boundary conditions (4.70) for $i = 1, 2, \dots, n-1$. As the nodes potential collected in vector $\mathbf{\Phi}$ is continuous on the interface and $\mathbf{J}_{\mathbf{n}\Gamma_2}^{(2)} = -\mathbf{J}_{\mathbf{n}\Gamma_2}^{(1)}$ the upper indices will be omitted. In order to focuss our attention let us assume that number of subregions is $n = 4$. Not loosing generality of consideration it would be more easy to understand process of rearranging matrix of coefficients. Finally we get the following system of equations:

$$\left[\begin{array}{cc|cc|cc|c} \mathbf{A}_{11}^{(1)} - \mathbf{B}_{11}^{\prime(1)} & \mathbf{A}_{12}^{(1)} & -\mathbf{B}_{12}^{(1)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{21}^{(1)} - \mathbf{B}_{21}^{\prime(1)} & \mathbf{A}_{22}^{(1)} & -\mathbf{B}_{22}^{(1)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{A}_{11}^{(2)} & +\mathbf{B}_{11}^{(2)} & \mathbf{A}_{12}^{(2)} & -\mathbf{B}_{12}^{(2)} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{21}^{(2)} & +\mathbf{B}_{21}^{(2)} & \mathbf{A}_{22}^{(2)} & -\mathbf{B}_{22}^{(2)} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{A}_{11}^{(3)} & +\mathbf{B}_{11}^{(3)} & \mathbf{A}_{12}^{(3)} & -\mathbf{B}_{12}^{(3)} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{A}_{21}^{(3)} & +\mathbf{B}_{21}^{(3)} & \mathbf{A}_{22}^{(3)} & -\mathbf{B}_{22}^{(3)} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{A}^{(4)} & \mathbf{B}^{(4)} \end{array}\right] \left[\begin{array}{c} \mathbf{\Phi}_{\Gamma_1} \\ \mathbf{\Phi}_{\Gamma_2} \\ \hline \mathbf{J}_{\mathbf{n}\Gamma_2}^{(1)} \\ \mathbf{\Phi}_{\Gamma_3} \\ \hline \mathbf{J}_{\mathbf{n}\Gamma_3}^{(2)} \\ \mathbf{\Phi}_{\Gamma_4} \\ \hline \mathbf{J}_{\mathbf{n}\Gamma_4}^{(3)} \end{array}\right] = \left[\begin{array}{c} \mathbf{q_1}^{(1)} \\ \mathbf{q_2}^{(1)} \\ \hline \mathbf{0} \\ \mathbf{0} \\ \hline \mathbf{0} \\ \mathbf{0} \\ \hline \mathbf{0} \end{array}\right].$$

Extending our consideration to n subregions we will get the final form of set

of equations:

$$\left[\begin{array}{c|c|c|c|c|c|c|c} \mathbf{A}_{11}^{(1)}-\mathbf{B}_{11}^{\prime(1)} & \mathbf{A}_{12}^{(1)} & -\mathbf{B}_{12}^{(1)} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{21}^{(1)}-\mathbf{B}_{21}^{\prime(1)} & \mathbf{A}_{22}^{(1)} & -\mathbf{B}_{22}^{(1)} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{A}_{11}^{(2)} & +\mathbf{B}_{11}^{(2)} & \mathbf{A}_{12}^{(2)} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{21}^{(2)} & +\mathbf{B}_{21}^{(2)} & \mathbf{A}_{22}^{(2)} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & +\mathbf{B}_{11}^{(n-1)} \mathbf{A}_{12}^{(n-1)} & -\mathbf{B}_{12}^{(n-1)} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & +\mathbf{B}_{21}^{(n-1)} \mathbf{A}_{22}^{(n-1)} & -\mathbf{B}_{22}^{(n-1)} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \quad \mathbf{A}^{(n)} & \mathbf{B}^{(n)} \end{array}\right] \left[\begin{array}{c} \mathbf{\Phi}_{\Gamma_1} \\ \hline \mathbf{\Phi}_{\Gamma_2} \\ \hline \mathbf{J}_{\mathbf{n}\Gamma_2}^{(1)} \\ \hline \mathbf{\Phi}_{\Gamma_3} \\ \hline \vdots \\ \hline \mathbf{J}_{\mathbf{n}\Gamma_{n-1}}^{(n-2)} \\ \hline \mathbf{\Phi}_{\Gamma_n} \\ \hline \mathbf{J}_{\mathbf{n}\Gamma_n}^{(n-1)} \end{array}\right] = \left[\begin{array}{c} \mathbf{q_1}^{(1)} \\ \hline \mathbf{q_2}^{(1)} \\ \hline \mathbf{0} \\ \hline \mathbf{0} \\ \hline \vdots \\ \hline \mathbf{0} \\ \hline \mathbf{0} \\ \hline \mathbf{0} \end{array}\right].$$

4.7 Index mismatched diffusive/diffusive interfaces

In a previous section we have considered the common case when the refractive indices are equal to each other and are constant throughout both media. However, in practice, existing refractive index[2] mismatch causes some problems and the boundary conditions Eq. (4.70) can not be imposed [39, 17].

For refracting index mismatched ($n_0 \neq n_1$) along diffusive/diffusive interfaces the following boundary conditions have to be imposed:

$$\begin{aligned} \Phi_{\Gamma_i{}^-} - \left(\frac{n_1}{n_0}\right)^2 \Phi_{\Gamma_i{}^+} &= C_n J_n \,, \\ J_n = -D_0 \left.\frac{\partial \Phi_{\Gamma_i{}^+}}{\partial n}\right|_{\Gamma_i} &= -\left(-D_1 \left.\frac{\partial \Phi_{\Gamma_i{}^-}}{\partial n}\right|_{\Gamma_i}\right), \end{aligned} \tag{4.74}$$

where the total flux at the boundary must be continuous i.e., $J_n = J_{n\Gamma_1^+} = J_{n\Gamma_1^-}$ and where: $C_n = \frac{2-\mathcal{R}_J^{1\to 0}-\mathcal{R}_J^{0\to 1}}{\mathcal{R}_\Phi^{1\to 0}}$.

In case when refracting index $n_0 = n_1$ one gets $\mathcal{R}_J^{1\to 0} = 1$ and $\mathcal{R}_\Phi^{1\to 0} = 0.5$ and $C_n = 0$ obtaining again conditions defined by Eq. (4.70). As it is stated

[2]The speed of all electromagnetic radiation in vacuum is the same, approximately $3 \cdot 10^8$ m/s, and is denoted by c. Therefore, if v is the phase velocity of radiation of a specific frequency in a specific material, the refractive index is given by $n = \frac{c}{v}$.

in [39] the most common range of refractive indices in biological media is $1.3 < n < 1.5$ and therefore maximum expected value for $n_1 > n_0$ is $C_n \sim 5$.

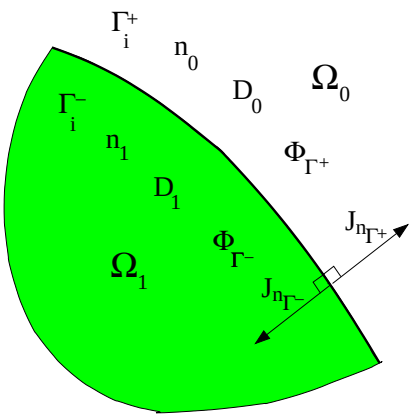


Figure 4.14: Interface between diffusive regions with different refractive indices

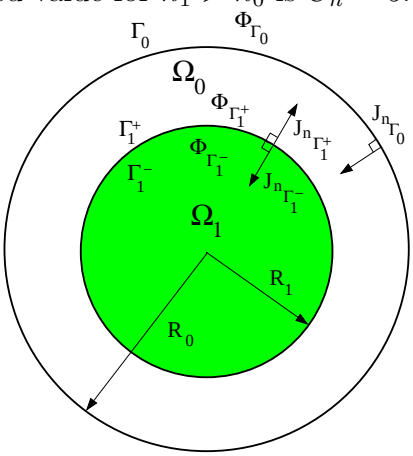


Figure 4.15: Cross-section of spherical region with the interface

In such cases it is suggested to implement approximate interface conditions which are valid as long as we can consider $\left(\frac{n_1}{n_0}\right)^2 \Phi_{\Gamma_i+} >> C_n J_n$ and this condition holds when the refractive indices are within the range mentioned above.

4.7.1 Approximate interface conditions

Let start with a simplified version of the boundary/interface conditions (Eq. (4.74)). Then the approximate interface conditions will be expressed as follows:

$$\begin{aligned} \Phi_{\Gamma_i-} &\simeq \left(\frac{n_1}{n_0}\right)^2 \Phi_{\Gamma_i+}\,, \\ -D_0 \left.\frac{\partial \Phi_{\Gamma_i+}}{\partial n}\right|_{\Gamma_i} &= -\left(-D_1 \left.\frac{\partial \Phi_{\Gamma_i-}}{\partial n}\right|_{\Gamma_i}\right)\,. \end{aligned} \tag{4.75}$$

Let focus our attention on the spherical region with internal interface as shown in Fig. 4.15. The system of equations describing the solution in the structure consisting of two subregions is:

1. An equation for **subregion Ω_0** bounded by surface Γ_0 and surface Γ_1^+ is:

$$c(\mathbf{r})\Phi^{(0)}(\mathbf{r}) + \int_{\Gamma} \frac{\partial G^{(0)}(|\mathbf{r}-\mathbf{r}'|,\omega)}{\partial n}\Phi^{(0)}(\mathbf{r}')d\Gamma(\mathbf{r}') = \tag{4.76}$$

$$= \int_{\Gamma} G^{(0)}(|\mathbf{r}-\mathbf{r}'|,\omega)\frac{\partial \Phi^{(0)}(\mathbf{r}')}{\partial n}d\Gamma(\mathbf{r}') - \sum_{s=1}^{ns} Q_s G^{(0)}(|\mathbf{r}_s-\mathbf{r}|,\omega)\,,$$

where $\Phi^{(0)}$ consists of unknowns on the boundary Γ_0 – Φ_{Γ_0} and on the interface Γ_1^+ – $\Phi_{\Gamma_1^+}$ (see Fig. 4.15),
for surface Γ_0 Robin boundary conditions holds:

$$\Phi_{\Gamma_0} - mJ_{n_{\Gamma_0}} = 0\,, \tag{4.77}$$

where $m = 2A$ [39, 44]. The coefficient A can be calculated as follows:

$$A = \frac{2/(1-R_0) - 1 + |\cos\Theta_c|^3}{1 - |\cos\Theta_c|^2}\,, \tag{4.78}$$

where $\Theta_c = \arcsin(1/n)$ and $R_0 = (n-1)^2/(n+1)^2$. For the value of refractive index $n = 1.333$ we get $A = 2.3645$.

2. The second equation for **subregion Ω_1** bounded by surface Γ_1^- is:

$$c(\mathbf{r})\Phi_{\Gamma_1^-}(\mathbf{r}) + \int_{\Gamma_1} \frac{\partial G^{(1)}(|\mathbf{r}-\mathbf{r}'|,\omega)}{\partial n}\Phi_{\Gamma_1^-}(\mathbf{r}')d\Gamma(\mathbf{r}') =$$

$$= \int_{\Gamma_1} G^{(1)}(|\mathbf{r}-\mathbf{r}'|,\omega)\frac{\partial \Phi_{\Gamma_1^-}(\mathbf{r}')}{\partial n}d\Gamma(\mathbf{r}') \quad , \tag{4.79}$$

and for the surface Γ_1 we impose the approximate interface boundary conditions (see Eq. (4.75)).

$$\Phi_{\Gamma_1^-} \simeq \left(\frac{n_1}{n_0}\right)^2 \Phi_{\Gamma_1^+} \qquad \text{and} \qquad J_{n_{\Gamma_1^-}} = -J_{n_{\Gamma_1^+}}\,. \tag{4.80}$$

The above two equations we may rewrite in the following matrix form replacing function Φ and J by the vectors containing their discrete nodal values:

$$\begin{bmatrix} \mathbf{A}_{11}^{(0)} & \mathbf{A}_{12}^{(0)} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{21}^{(0)} & \mathbf{A}_{22}^{(0)} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_{33}^{(0)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{A}_{44}^{(1)} \end{bmatrix} \begin{bmatrix} \mathbf{\Phi}_{\Gamma_0} \\ \mathbf{\Phi}_{\Gamma_1^+} \\ \mathbf{\Phi}_{\Gamma_0} \\ \mathbf{\Phi}_{\Gamma_1^-} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{11}^{(0)} & \mathbf{B}_{12}^{(0)} & \mathbf{0} & \mathbf{0} \\ \mathbf{B}_{21}^{(0)} & \mathbf{B}_{22}^{(0)} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}_{33}^{(0)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{B}_{44}^{(1)} \end{bmatrix} \begin{bmatrix} \mathbf{J}_{n\Gamma_0} \\ \mathbf{J}_{n\Gamma_1^+} \\ \mathbf{J}_{n\Gamma_0} \\ \mathbf{J}_{n\Gamma_1^-} \end{bmatrix} + \begin{bmatrix} \mathbf{q_1}^{(0)} \\ \mathbf{q_2}^{(0)} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}.$$

Reducing the number of unknowns using approximate interface conditions and replacing $\frac{\partial\Phi^{(0)}}{\partial n}$ by J_n we get:

$$c(\mathbf{r})\Phi^{(0)}(\mathbf{r}) + \int\limits_{\Gamma} \frac{\partial G^{(0)}(|\mathbf{r}-\mathbf{r}'|,\omega)}{\partial n}\Phi^{(0)}(\mathbf{r}')d\Gamma(\mathbf{r}') \;=$$

$$= -\frac{1}{D_0}\int\limits_{\Gamma} G^{(0)}(|\mathbf{r}-\mathbf{r}'|,\omega)J_{n\Gamma_0}d\Gamma(\mathbf{r}') - \sum_{s=1}^{ns} Q_s G^{(0)}(|\mathbf{r}_s-\mathbf{r}|,\omega),$$

$$\Phi_{\Gamma_0} - mJ_{n_{\Gamma_0}} = 0,$$

$$c(\mathbf{r})\left(\frac{n_1}{n_0}\right)^2\Phi_{\Gamma_1^+}(\mathbf{r}) + \int\limits_{\Gamma} \frac{\partial G^{(1)}(|\mathbf{r}-\mathbf{r}'|,\omega)}{\partial n}\left(\frac{n_1}{n_0}\right)^2\Phi_{\Gamma_1^+}(\mathbf{r}')d\Gamma(\mathbf{r}') \;=$$

$$= -\frac{1}{D_1}\int\limits_{\Gamma} G^{(1)}(|\mathbf{r}-\mathbf{r}'|,\omega)\left(-J_{n\Gamma_1^+}\right)d\Gamma(\mathbf{r}')\,. \qquad (4.81)$$

Now, eliminating the unknowns $\mathsf{J}_{n_{\Gamma_0}}$ the system of equations in a matrix form is as follows:

$$\begin{bmatrix} \mathbf{A}_{11}^{(0)} & \mathbf{A}_{12}^{(0)} & \mathbf{0} \\ \mathbf{A}_{21}^{(0)} & \mathbf{A}_{22}^{(0)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \left(\frac{n_1}{n_0}\right)^2\mathbf{A}_{44}^{(1)} \end{bmatrix}\begin{bmatrix} \mathbf{\Phi}_{\Gamma_0} \\ \mathbf{\Phi}_{\Gamma_1^+} \\ \mathbf{\Phi}_{\Gamma_1^+} \end{bmatrix} = \begin{bmatrix} \mathbf{B'}_{11}^{(0)} & \mathbf{B}_{12}^{(0)} & \mathbf{0} \\ \mathbf{B'}_{21}^{(0)} & \mathbf{B}_{22}^{(0)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}_{44}^{(1)} \end{bmatrix}\begin{bmatrix} \mathbf{\Phi}_{\Gamma_0} \\ \boldsymbol{J}_{\boldsymbol{n}\Gamma_1^+} \\ \boldsymbol{J}_{\boldsymbol{n}\Gamma_1^+} \end{bmatrix} + \begin{bmatrix} \mathbf{q_1}^{(0)} \\ \mathbf{q_2}^{(0)} \\ \mathbf{0} \end{bmatrix},$$

where $\mathbf{B'}_{11}^{(0)} = \mathbf{B}_{11}^{(0)}\left(\mathbf{B}_{33}^{(0)}\right)^{-1}\mathbf{A}_{33}^{(0)}$.

Now, we have only three group, of unknowns: $\mathbf{\Phi}_{\Gamma_0}$, $\mathbf{\Phi}_{\Gamma_1^+}$ and $\mathbf{J}_{\mathbf{n}\Gamma_1^+}$. Transferring unknowns to the left hand side of the system we will finally get:

$$\begin{bmatrix} \mathbf{A}_{11}^{(0)} & \mathbf{A}_{12}^{(0)} & -\mathbf{B}_{12}^{(0)} \\ \mathbf{A}_{21}^{(0)} & \mathbf{A}_{22}^{(0)} & -\mathbf{B}_{22}^{(0)} \\ \mathbf{0} & \left(\frac{n_1}{n_0}\right)^2\mathbf{A}_{44}^{(1)} & -\mathbf{B}_{44}^{(1)} \end{bmatrix}\begin{bmatrix} \mathbf{\Phi}_{\Gamma_0} \\ \mathbf{\Phi}_{\Gamma_1^+} \\ \mathbf{J}_{\mathbf{n}\Gamma_1^+} \end{bmatrix} = \begin{bmatrix} \mathbf{q_1}^{(0)} \\ \mathbf{q_2}^{(0)} \\ \mathbf{0} \end{bmatrix}. \qquad (4.82)$$

Let consider two concentric spheres which cross-section is shown in Fig. 4.15. Their dimensions are: $R_0 = 25$ mm and $R_0 = 22.5$ mm. The amplitude and phase distribution on the perimeter of the region are presented in the Fig. 4.16. For approximate interface conditions, when the coefficient $\left(\frac{n_1}{n_0}\right)$ increases the phase shift decreases significantly (see Fig. 4.16) but amplitude

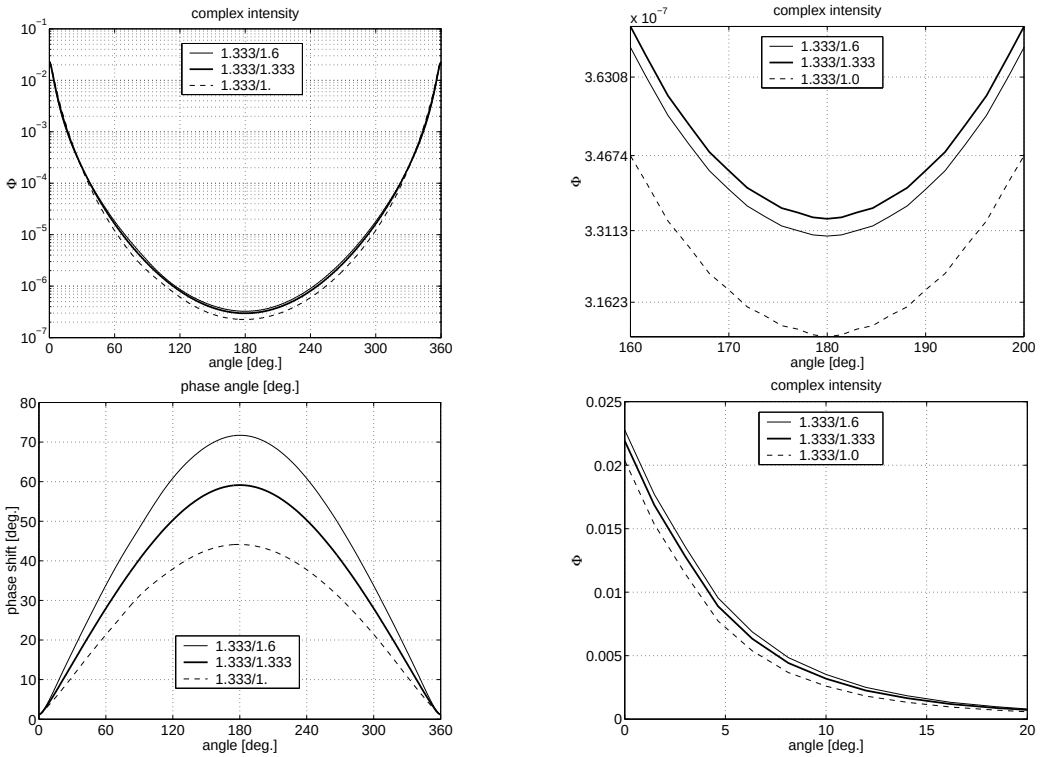


Figure 4.16: Left column of the figures-approximate interface conditions results for refractive index mismatch case $\left(\frac{n_1}{n_0}\right) = 1.33/1.6 = 0.83$, refractive index match case $\left(\frac{n_1}{n_0}\right) = 1.33/1.33 = 1.0$ and refractive index mismatch case $\left(\frac{n_1}{n_0}\right) = 1.33/1. = 1.33$; the right column of the figures-in linear scale presents enlarged amplitude distribution to show that graphs on top left are close each to other especially in logarithmic scale but are not the same

remains almost unchanged for all cases. That seems to be not justified from physical point of view. That is why we will implement the complete boundary conditions Eq. (4.74).

4.7.2 Complete interface conditions

If we consider complete or saltus [39] interface boundary conditions then $\Phi_{\Gamma_1^-}$ will read:

$$\Phi_{\Gamma_1^-} = \left(\frac{n_1}{n_0}\right)^2 \Phi_{\Gamma_1^+} - C_n J_{n_{\Gamma_1^+}} \,. \tag{4.83}$$

Now, introducing Eq. (4.83) into Eq. (4.79) we will get:

$$
\begin{aligned}
c(\mathbf{r})\Phi^{(0)}(\mathbf{r}) + \int_{\Gamma} \frac{\partial G^{(0)}(|\mathbf{r}-\mathbf{r}'|,\omega)}{\partial n}\Phi^{(0)}(\mathbf{r}')d\Gamma(\mathbf{r}') &= \\
= -\frac{1}{D_0}\int_{\Gamma} G^{(0)}(|\mathbf{r}-\mathbf{r}'|,\omega)J_{n\Gamma_0}d\Gamma(\mathbf{r}') - \sum_{s=1}^{ns} Q_s G^{(0)}(|\mathbf{r}_s-\mathbf{r}|,\omega), & \\
\Phi_{\Gamma_0} + mJ_{n_{\Gamma_0}} &= 0, \\
c(\mathbf{r})\left(\frac{n_1}{n_0}\right)^2\Phi_{\Gamma_1^+}(\mathbf{r}) + \int_{\Gamma} \frac{\partial G^{(1)}(|\mathbf{r}-\mathbf{r}'|,\omega)}{\partial n}\left(\frac{n_1}{n_0}\right)^2\Phi_{\Gamma_1^+}(\mathbf{r}')d\Gamma(\mathbf{r}') &= \\
= c(\mathbf{r})C_n J_{n\Gamma_1^+} + \int_{\Gamma} \frac{\partial G^{(1)}(|\mathbf{r}-\mathbf{r}'|,\omega)}{\partial n}C_n J_{n\Gamma_1^+}d\Gamma(\mathbf{r}') &+ \\
+\frac{1}{D_1}\int_{\Gamma} G^{(1)}(|\mathbf{r}-\mathbf{r}'|,\omega)J_{n\Gamma_1^+}d\Gamma(\mathbf{r}'). & \qquad (4.84)
\end{aligned}
$$

In a matrix form, the system of integral equations Eq. (4.84) is:

$$
\begin{bmatrix}
\mathbf{A}_{11}^{(0)} & \mathbf{A}_{12}^{(0)} & -\mathbf{B}_{12}^{(0)} \\
\mathbf{A}_{21}^{(0)} & \mathbf{A}_{22}^{(0)} & -\mathbf{B}_{22}^{(0)} \\
\mathbf{0} & \left(\frac{n_1}{n_0}\right)^2\mathbf{A}_{44}^{(1)} & -\mathbf{B'}_{44}^{(1)} - \mathbf{B}_{44}^{(1)}
\end{bmatrix}
\begin{bmatrix}
\mathbf{\Phi}_{\Gamma_0} \\ \mathbf{\Phi}_{\Gamma_1^+} \\ \mathbf{J}_{\mathbf{n}\Gamma_1^+}
\end{bmatrix}
=
\begin{bmatrix}
\mathbf{q_1}^{(0)} \\ \mathbf{q_2}^{(0)} \\ \mathbf{0}
\end{bmatrix}, \qquad (4.85)
$$

where the new submatrix $\mathbf{B'}_{44}^{(1)}$ of the above system is calculated from discretized form of the following term (see Eq. (4.84)):

$$
c(\mathbf{r})C_n + \int_{\Gamma} \frac{\partial G^{(1)}(|\mathbf{r}-\mathbf{r}'|,\omega)}{\partial n}C_n d\Gamma(\mathbf{r}')
$$

.

Numerically, solving the integral equations Eq. (4.84) requires discretization of the boundary curve or the boundary surface. In case of curved boundary, the limit of validity of saltus condition expressed in Eq. (4.74), should be studied, since this expression was originally derived for a locally plane interface. In the work [39] it was proved that the error committed by using

the expression $\mathcal{R}_{\Phi,J}^{j\to k}$ for a plane interface is never higher than 0.5%. The Author of [39] for practical applications has used values of the refractive index n_1 between 1.0 and 3.0. Considering our example we can conclude that we have big enough curvature radius and small enough boundary elements (the segment is not bigger than 10^o), to achieve satisfactory results. Some

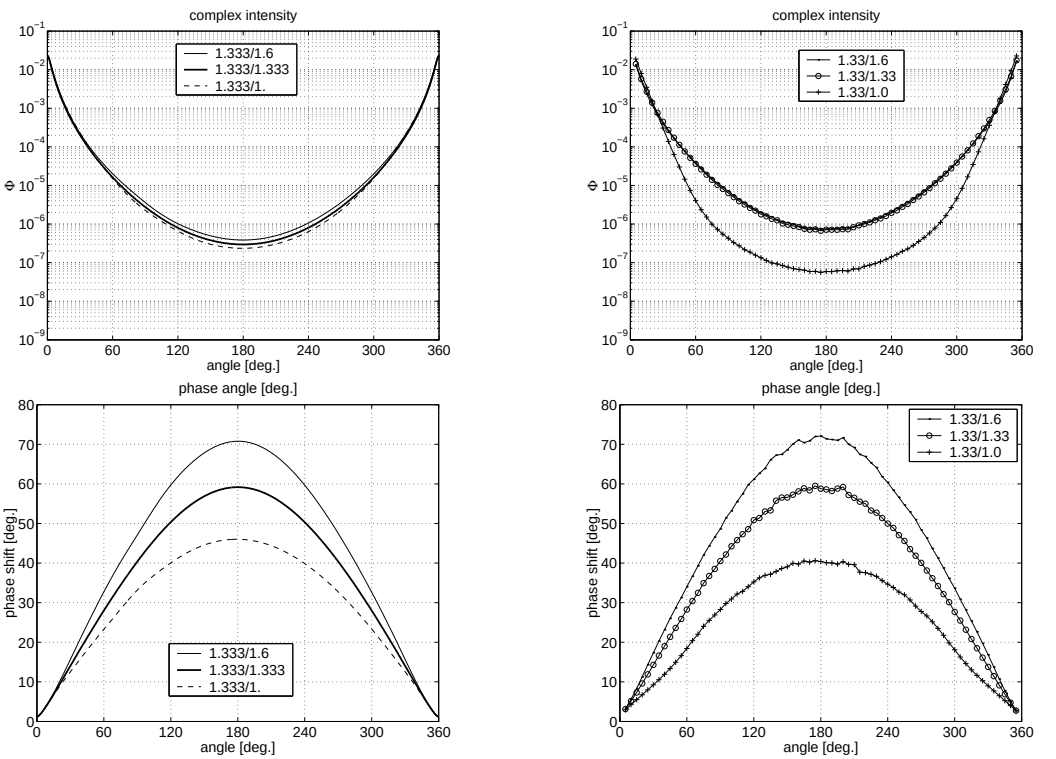


Figure 4.17: BEM with saltus interface conditions results for refractive index mismatch ($C_n = 0.2$)-left column and FMC results-right column

typical values of C_n are shown in [39], where we see that in cases in which $n_1 < n_0$, we obtain values of $C_n < 2.5$. The most relevant case corresponds to $n_0 = 1.33$, since it is the most common value in biological media.

For calculation we have assumed that $C_n = 0.2$. The results are very close to the results presented in Fig. 4.16. In order to emphasis the differences for different refractive indices all cases were put in one figure.

For comparison, the results achieved with the aid of the Frequency Monte-Carlo method were also presented (Fig. 4.17 top and bottom right). As we can see in the Fig. 4.17 the BEM and the FMC results for the cases $\frac{1.33}{1.6}$ and $\frac{1.33}{1.33}$ are really close for the amplitude and the phase shift. Unfortunately the

case $\frac{1.33}{1.0}$ demand further validation as the results seem to be not correct.

4.8 Numerical examples

All examples presented in this section are selected in such a way to make sure that boundary element method will provide a reliable result in impedance or optical tomography.

For Electrical Impedance Tomography (EIT) and its industrial applications we have to deal with regions possessing sharp edges and corners. For Diffuse Optical Tomography (DOT) mostly we have to deal with medical applications where the regions possess rather smooth surfaces without sharp edges and corners.

That is why we have also studied close placed surfaces of spheres as well as inhomogeneous regions with refractive index match and mismatch in case of DOT. Whenever that was possible, results were compared with analytical solution [3] or with the other numerical methods like the Finite Element [3, 4] or Monte-Carlo [5]. The main attention is focused on 3D cases but more interesting 2D examples were also considered (see the previous sections).

4.8.1 Two concentric spheres

Two concentric spheres can be recognized as a simplified numerical model of the CSF layer of the baby's head. That is why so much attention will be devoted to this particular region configuration. We will consider the spherical object of standard dimensions of 25 mm and the internal embedded sphere of 24 mm simulating the baby's head.Our intention is to show haw difficult such region is for the boundary element method. This investigation we will start first with Laplace equation next, the more difficult problem of diffusion equation will be considered. The errors of the solution on the boundary as well as inside the structure will be controlled.

The region was discretised by 1536 isoparametric 6 nodes triangular boundary elements, that gives 3074 nodes (see Fig. 4.18). Solution for the internal function and its relative error is shown in Fig. 4.19.

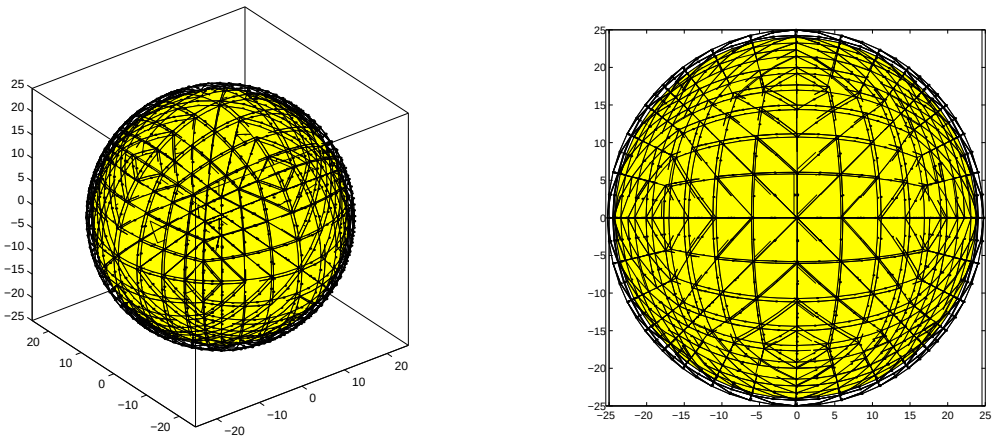


Figure 4.18: The outer and inner shell meshes (left) and external surface discretization by 6 nodes isoparametric triangles

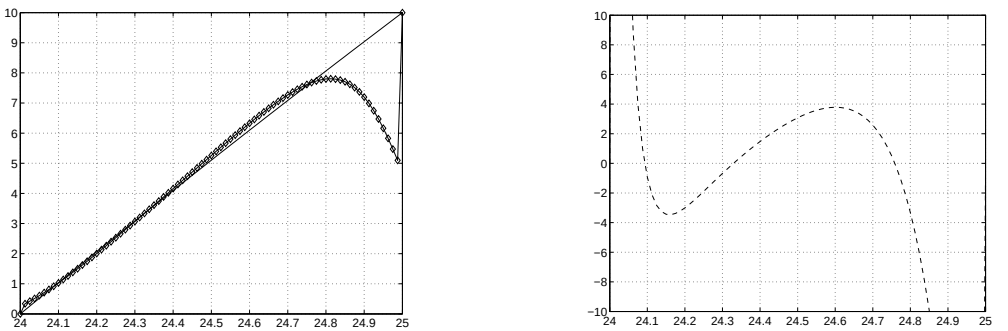


Figure 4.19: Analytical and numerical solution (solid and dotted line respectively) on the left and relative error distribution for potential function Φ inside the region (right)

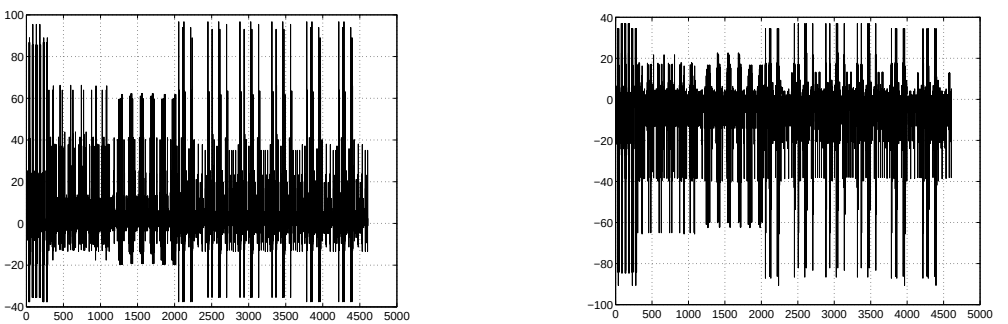


Figure 4.20: Relative error distribution for the $\frac{d\Phi}{dn}$ on the external sphere (left) and relative error distribution for the $\frac{d\Phi}{dn}$ on the internal sphere (right)

Comparing results we can see that the potential distribution however insufficiently precise gives the hope for a satisfactory result if discretization would

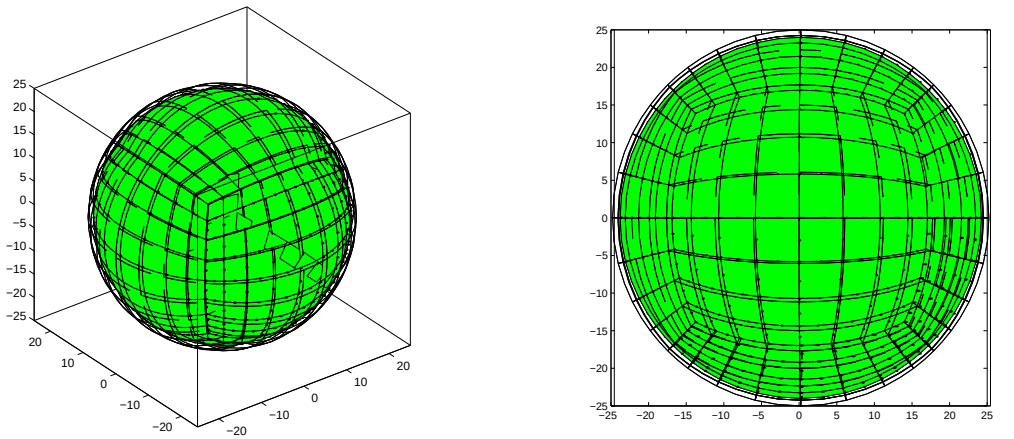


Figure 4.21: The outer and inner shell meshes (left) and external surface for coarse discretization by quadrilateral boundary element

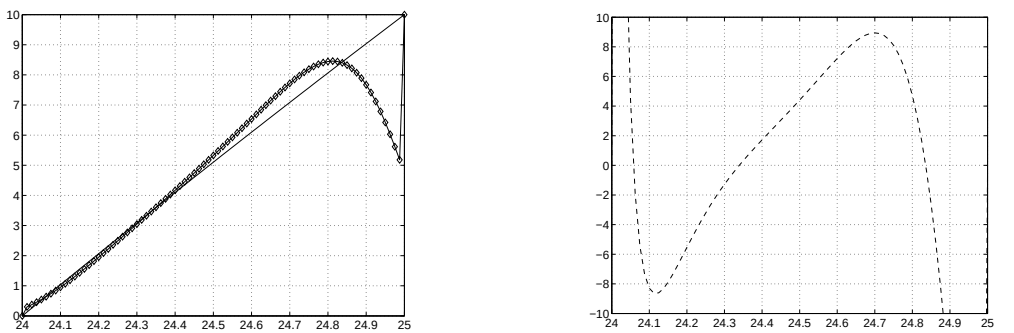


Figure 4.22: Analytical and numerical solution (solid and dotted line respectively) (left) and relative error distribution for potential function Φ inside the region

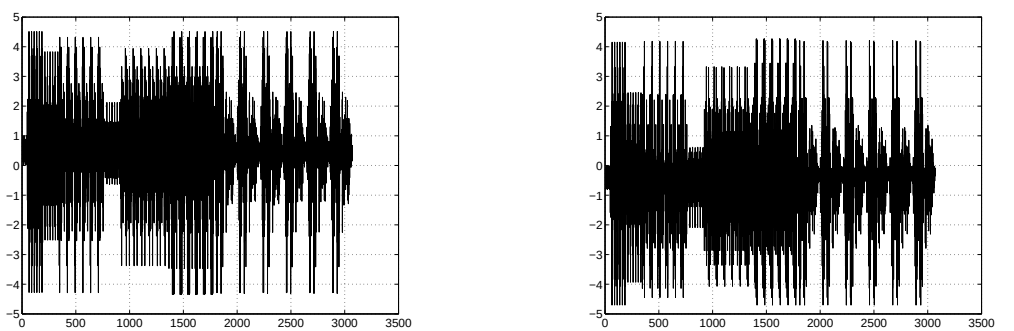


Figure 4.23: Relative error distribution for the $\frac{d\Phi}{dn}$ on the external sphere (left) and relative error distribution for the $\frac{d\Phi}{dn}$ on the internal sphere

be properly dense. The relative error distributions for the surface values are presented in the Fig. 4.20. Looking at the results for the isoparametric 6

nodes triangular boundary elements surprisingly high relative error occurs for a certain nodes (see Fig. 4.20).

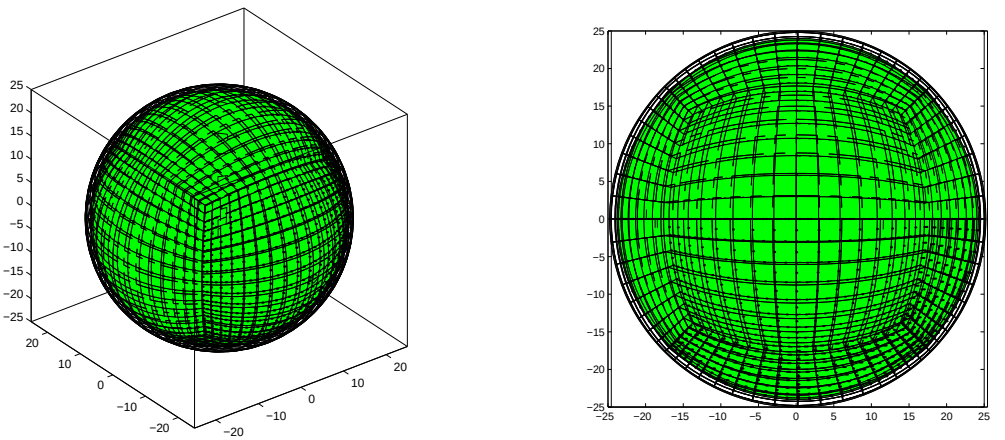


Figure 4.24: The outer and inner shell meshes (left) and external surface for dense discretization by quadrilateral boundary element

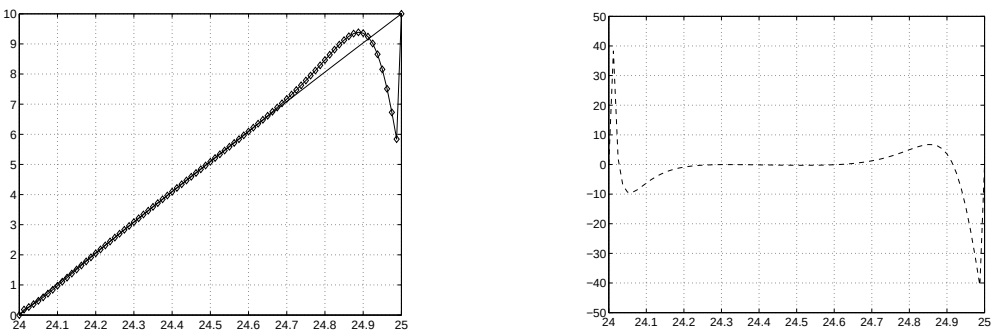


Figure 4.25: Analytical and numerical solution (solid and dotted line respectively) on the left and relative error distribution for potential function Φ inside the region on the right

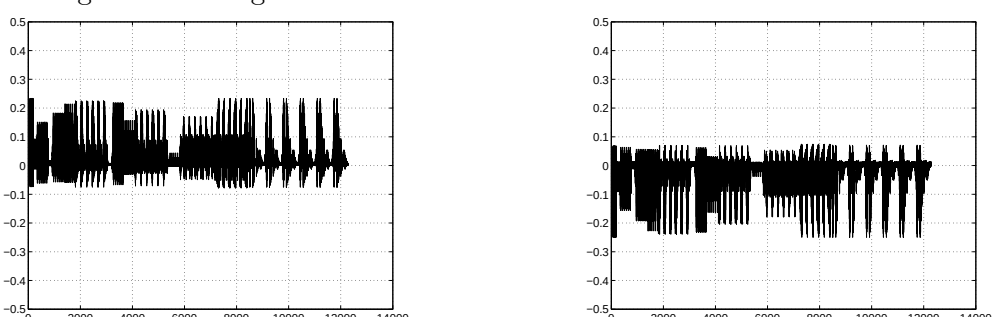


Figure 4.26: Relative error distribution for the $\frac{d\Phi}{dn}$ on the external sphere (left) and relative error distribution for the $\frac{d\Phi}{dn}$ on the internal sphere (right)

It is interesting how the precision of thin layers calculation depends on the kind of the boundary elements. Now the region under consideration was discretised by 768 isoparametric 8 nodes quadrilateral isoparametric boundary elements. That gives 2308 nodes. Discretisation of the region is presented in Fig. 4.21.

The internal field solution and the relative error are shown in the Fig. 4.22.

The relative error distributions for the surface values are presented in the Fig. 4.23. Results are still unsatisfactory, so in order to investigate the influence of the discretization on the precision of the solution, this region was discretised by 3072 isoparametric 8 nodes quadrilateral boundary elements (see Fig. 4.24). That gives 9220 nodes in total.

The relative error distributions for the surface values are presented in the figures Fig. 4.25 and Fig. 4.26. This time the relative error for the surface quantities dropped even below 0.3%.

4.9 Diffusion model for light transport in the frequency domain

Let consider a domain Ω with boundary Γ . Light transport in scattering tissue is commonly described by the diffusion approximation to the transport equation [3], a second order elliptic partial differential equation:

$$(\nabla \cdot D\nabla - \mu_a + i\omega/c)\Phi(\mathbf{r},\omega) = q_0(\mathbf{r},\omega) \ \forall \quad \mathbf{r} \in \Omega \setminus \Gamma\,, \tag{4.86}$$

where Φ stands for photon density, diffusion coefficient $D = \frac{1}{3(\mu_a+\mu_s')}$, μ_a is an absorbing and μ_s' is reduced scattering coefficient, the speed of light $c(\mathbf{r}) = c_0/\nu(\mathbf{r})$, where $\nu(\mathbf{r})$ is the refractive index and c_0 is the speed of light in a vacuum, and q_0 is a source of light with modulation frequency ω.

Boundary conditions are usually taken as Robin type [44]:

$$\Phi(\mathbf{r},\omega) + 2\alpha D\frac{\partial\Phi(\mathbf{r},\omega)}{\partial n} = 0, \qquad \forall \quad \mathbf{r} \in \Gamma\,, \tag{4.87}$$

where coefficient α depends on refractive index.

In the case where scattering and absorption are homogeneous, Eq. (4.86) reduces to a Helmholtz equation with complex wave number $k = \sqrt{\frac{\mu_a}{D} - i\frac{\omega}{cD}}$:

$$\nabla^2\Phi(\mathbf{r},\omega) - k^2\Phi(\mathbf{r},\omega) = -\frac{q_0(\mathbf{r},\omega)}{D} \quad \forall \quad \mathbf{r} \in \Omega \setminus \Gamma\,, \tag{4.88}$$

with the same boundary condition as those expressed by Eq. (4.87).

For the diffusion equation the fundamental solution is [3]:

$$G(|\mathbf{r}-\mathbf{r}'|,\omega) = \frac{1}{4\pi|\mathbf{r}-\mathbf{r}'|}e^{-k|\mathbf{r}-\mathbf{r}'|}\,. \tag{4.89}$$

The normal derivative of the Green function in a direction $\mathbf{n}$ can be written:

$$\mathbf{n}\cdot\nabla G = \mathbf{n}\cdot\boldsymbol{\rho}\left(\frac{-1}{4\pi|\mathbf{r}-\mathbf{r}'|^2} - \frac{k}{4\pi|\mathbf{r}-\mathbf{r}'|}\right)e^{-k|\mathbf{r}-\mathbf{r}'|}\,, \tag{4.90}$$

where $\boldsymbol{\rho} = \frac{\mathbf{r}-\mathbf{r}'}{|\mathbf{r}-\mathbf{r}'|}$.

The Boundary Element Method (BEM) proceeds by applying Green theorem in its second form [10] to derive an integral equation applicable on the surface:

$$\begin{aligned} c(\mathbf{r})\Phi(\mathbf{r}) + \int\limits_{\Gamma-\Gamma_\varepsilon} \frac{\partial G(|\mathbf{r}-\mathbf{r}'|,\omega)}{\partial n}\Phi(\mathbf{r}')d\Gamma(\mathbf{r}') = \\ = \int\limits_{\Gamma-\Gamma_\varepsilon} G(|\mathbf{r}-\mathbf{r}'|,\omega)\frac{\partial\Phi(\mathbf{r}')}{\partial n}d\Gamma(\mathbf{r}') - \int\limits_{\Omega} q_0 G(|\mathbf{r}-\mathbf{r}'|,\omega)d\Omega\,. \end{aligned} \tag{4.91}$$

The only difference between Eq. (4.6) and Eq. (4.91) is that the photon density function Φ and current photon function $\frac{\partial\Phi}{\partial n}$ are complex and we have complex valued Green functions (Eq. (4.89)).

In DOT, concentrated sources are frequently used and are very simple to handle in BEM. They are a special case for which the function q_0 at the internal point $\mathbf{r}_s$ becomes:

$$q_0 = Q_s\delta(\mathbf{r}_s)\,, \tag{4.92}$$

where Q_s is the magnitude of the source and $\delta(\mathbf{r}_s)$ is a Dirac delta function whose integral is equal to 1 over any volume containing the singularity point $\mathbf{r}_s$ and zero elsewhere.

Assuming that a number, n_s of these functions exists one can write:

$$c(\mathbf{r})\Phi(\mathbf{r}) + \int_{\Gamma} \frac{\partial G(|\mathbf{r}-\mathbf{r}'|,\omega)}{\partial n}\Phi(\mathbf{r}')d\Gamma(\mathbf{r}') =$$

$$= \int_{\Gamma} G(|\mathbf{r}-\mathbf{r}'|,\omega)\frac{\partial \Phi(\mathbf{r}')}{\partial n}d\Gamma(\mathbf{r}') - \sum_{s=0}^{ns-1} Q_s G(|\mathbf{r}_s-\mathbf{r}|,\omega)\,. \tag{4.93}$$

4.10 Results for 3D space

To test the influence of the discretization in the BEM on the accuracy of the method, we will use the homogeneous spherical region. To discretize the surface, two different boundary elements were used: isoparametric six nodes triangular and isoparametric eight nodes quadrilateral elements. In a gray color scale the logarithm of the solution on the surface of the sphere is visualized (see for example Fig. 4.27 and Fig. 4.28).

As a result of the solution of equation (4.86) with Robin boundary condition (4.87), the amplitude and the phase shift of the photon density Φ on the circumference of the cross-section of the sphere have been plotted, together with an analytical solution taken from [3].

Calculations have been performed for the cases of a point source in a sphere 50 mm in diameter. The source was located in a distance $r_d = 1/\mu_s'$ mm from the surface, and a refractive index of 1.0 was assumed, so the speed of light was $c = 0.3$ mm ps^{-1}. Values for μ_a of $0.025\,\mathrm{mm}^{-1}$ and for μ_s' of $2.0\,\mathrm{mm}^{-1}$ were used, which are representative for brain tissue [3]. The modulation frequency was equal to 200 MHz. In each case the equation (4.67) was solved with $\alpha = 1$. In order to achieve the solution of this problem, the GMRES [42] numerical solver was used.

To discretize the surface, two different boundary elements were used: isoparametric quadratic triangular boundary element and in the next sections the quadrilateral eight nodes boundary element. An important factor in mesh construction is to make the size of a distance between the nodes of elements

no bigger than the diffusion length[3]:

$$L_d = \sqrt{\frac{D}{\mu_a}} = \frac{1}{k}\bigg|_{\omega=0} . \tag{4.94}$$

Since this quantity is very small especially for near-infrared (**NIR**) light in the wavelength range 650 to 1200 nm, the diffusion equation is difficult to solve numerically.

For the case of isoparametric quadratic triangular elements the sphere surface had been discretized by 768 triangular boundary elements, which gave rise to 1538 nodes (see Fig. 4.27).

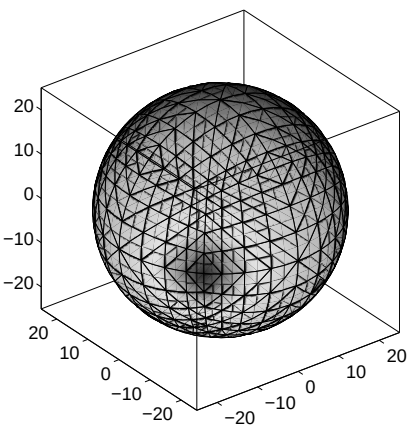


Figure 4.27: The sphere surface discretized by quadratic (six nodes) triangular elements

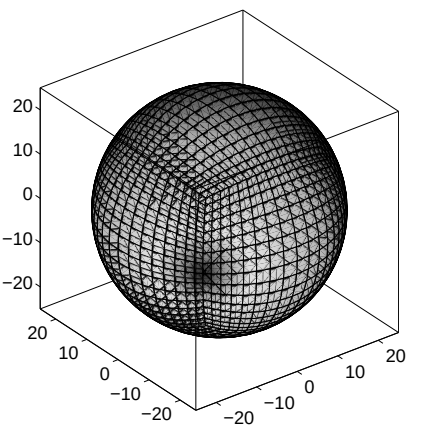


Figure 4.28: The sphere surface discretized by quadratic quadrilateral elements

Amplitude and a phase shift distribution of Φ against angle Θ for the equatorial plane of the sphere are presented in Fig. 4.29. Conformity with the analytical solution is very good, so it is hard to distinguish both curves.

Similar results were achieved using quadrilateral boundary element described in the previous section. This time, the surface of the sphere was discretized by 1536 elements with 4610 nodes (Fig. 4.28) providing better results, because the discretization is twice as dense as for triangular mesh.

[3]The diffusion length is a distance at which the photon density Φ decreases by a factor of e, which derives directly from (4.89) when $\omega = 0$.

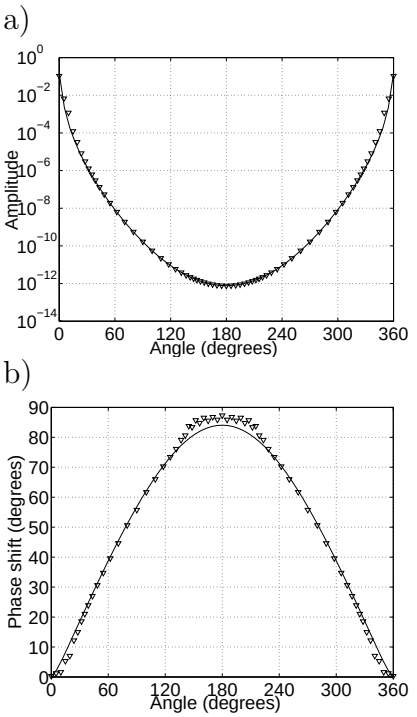


Figure 4.29: Comparison of the photon density results for a isoparametric triangular quadratic discretization ($\triangledown$) with analytical solution (solid line) a) log amplitude b) phase shift as a function of angle Θ

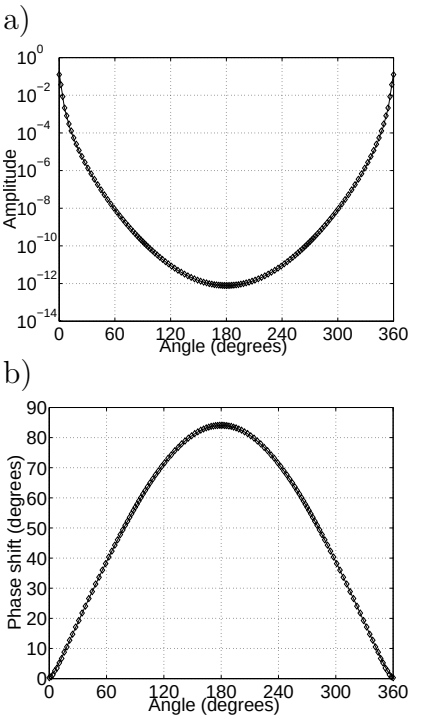


Figure 4.30: Comparison of the photon density results for a quadrilateral discretization (diamond) with analytical solution (solid line) a) log amplitude, b) phase shift as a function of angle Θ

This is particularly true for the phase shift, where for triangular discretization we can observe small oscillations (Fig. 4.29), disappearing for more dense discretization.

Amplitude and a phase shift distribution of Φ against the angle Θ for all two considered cases are presented in the Fig. 4.29 and Fig. 4.30. The analytical solution is a solid line but numerical is a line marked with triangles or diamonds for triangular or quadrilateral discretization, respectively.

In order to validate the numerical results it were compared with the analytical solution [3], however it is unknown how precise the analytical method is. That is why to estimate the precision of the numerical method we need some measure of the accuracy.

Sometimes authors dealing with the Boundary Element Method express opinion that only simplest boundary element provide acceptable from practical point of view results. That is why we decided to include some results which compare results achieved by 3 nodes flat triangle discretization and 6 nodes isoparametric triangle discretization of the same region with the same number of nodes. It does't mean the same number of unknowns as for the flat triangle the number of unknowns are related to the number of elements but for isoparametric 6 nodes triangle the number of unknowns are related to the number of nodes. The timings for both cases solution on a Pentium IV-1.0 GHz computer are collected in Table 4.4.

Table 4.4: Time performance for different kind of elements

type of element	*time (sec)*	*no of unknowns*
triangle const.	22.7	3072
triangle izopar.	4.8	1538

4.10.1 Validation of numerical results and measures of the accuracy

To estimate the precision of the numerical method we need some measure of the accuracy. In [30, 19] several measures of the accuracy of the calculated potential function distributions have been defined. They all compare the numerical solution against the exact one. The most widely used measure is the relative difference measure (RDM), which is defined as follows:

$$RDM = \sqrt{\frac{\int_S (\Phi - \Phi_{exact})^2 dS}{\int_S \Phi_{exact}^2 dS}}\,. \tag{4.95}$$

However, such a measure gives us the global error estimation, when in Optical Tomography we are mainly interested in local error distribution. The relative error distribution for the amplitude and the phase shift is presented in the Fig. 4.31.

For the flat triangular approximation we can observe large oscillations in the vicinity of the point source (see Fig. 4.31b), even though the discretization was quite dense, the isoparametric case gives more accurate result. That is

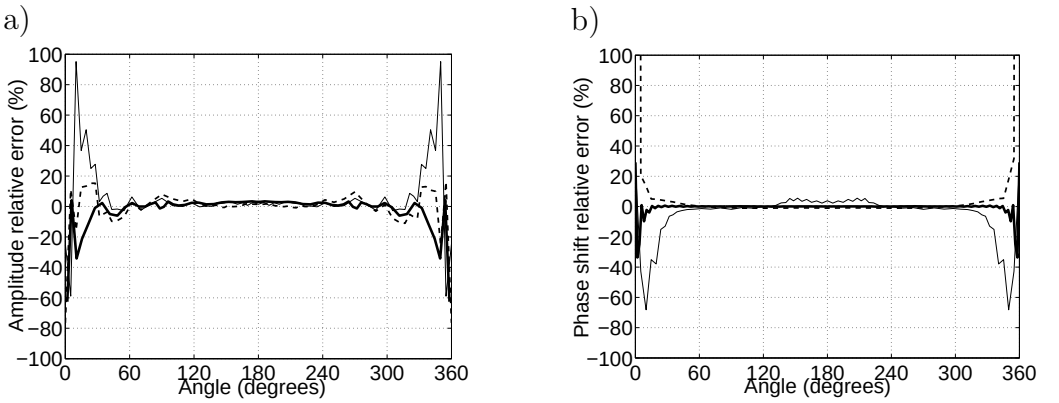


Figure 4.31: Distribution of the relative error for isoparametric triangle (solid line) and for quadrilateral 8 nodes boundary elements (thick solid line); additionally by dashed line are marked results for constant triangle: a) for the log amplitude b) for the phase shift

one of the many reason why in Diffuse Optical Tomography almost exclusively we are using isoparametric boundary elements.

4.10.2 The proximity effect

Because the purpose is to create the 3D BEM model of the baby's head, the behavior of the method in case of diffusion equation formulated for the frequency domain is very important when the surfaces become close to each other.

In the previous sections the thin layer structures were investigated in case of Laplace equation. Now we are interested in more complicated case, due to the fact that all quantities become complex.

Some of the integrals for the thin layers became nearly singular. We need to know the error which may be introduced by that kind of geometry. For this purposes we have created the structures with 1 mm, 2 mm and 3 mm distance between the surfaces in the homogeneous region, for which we know the analytical solution [3, 5]. In all three cases discretization remains the same. Comparison of the results are shown in Fig. 4.32 and Fig. 4.33.

Concluding this numerical experiment, we can say that magnitude is much more sensitive on the distance between the surfaces (gap) then the phase

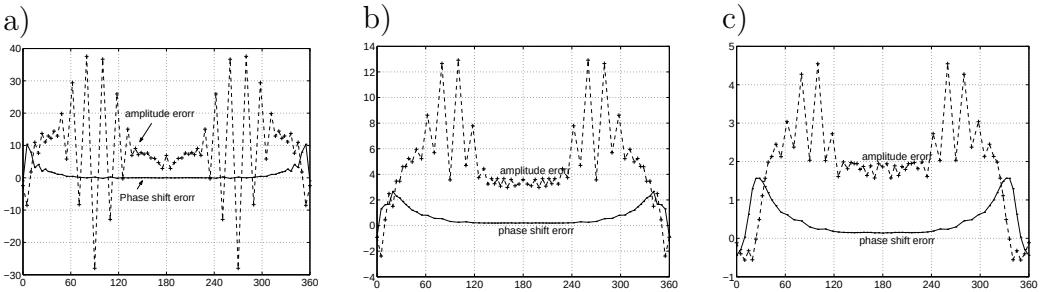


Figure 4.32: Thin layers solution for: a) 1 mm gap, b) 2 mm gap, c) 3 mm gap

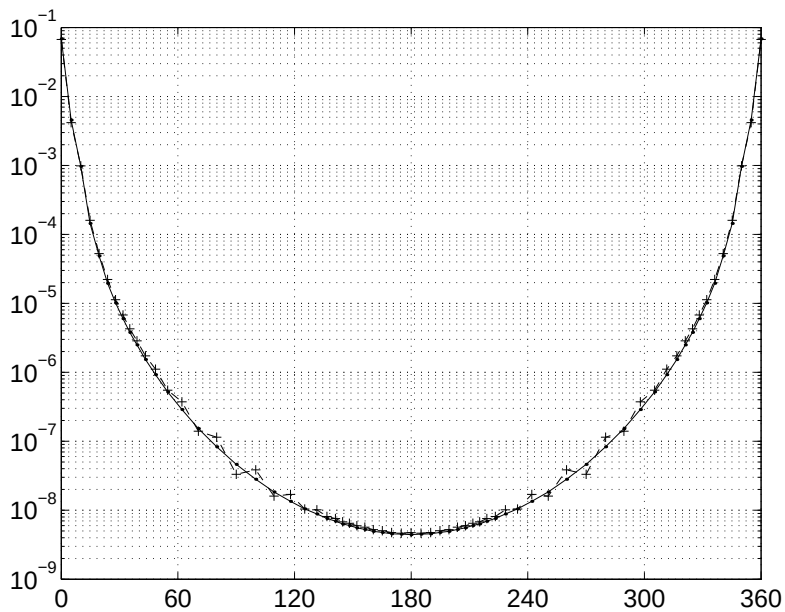


Figure 4.33: The amplitude distribution for 1mm gap (one can hardly recognize the differences between the "exact solution" and the numerical one

shift. The bigger gap the smaller relative error is. Anyway, it is possible to keep the relative error on a reasonable level if the proper discretization will be selected, as it was shown in the section 4.8.

4.11 Multilayered model of the neonatal head

Finally, the nonstructural meshes provided by David Holder co-worker were used [47, 53]. Next results for a four layer head model will be presented.

The surfaces modeled were the outer skin, the skull, the CSF layer and the brain. Those surfaces were generated from an MRI scan following tissue segmentation. The complexity of these meshes is that a volume mesh (like this) used for a finite element method (FEM) would be very difficult to construct. The optical parameters and mesh sizes are given in Table 4.5:

Table 4.5: Optical parameters and mesh discretisations for the four layer head model

	μ_a	μ_s'	nodes	elements
outer shell	0.0149	0.8	2849	1402
middle shell	0.01	1	3294	1646
inner shell	0.0178	1.25	2098	1048

In this chapter the 3D BEM numerical model for the forward problem formulated for Optical Tomography has been presented. The standard optical tomography benchmark has been solved for the diffusion equation in order to highlight the main advantages of BEM which could be useful in OT.

The results were compared with those presented in [3], showing good conformity, even though the BEM mesh was relatively coarse.

As the main advantages of BEM from the optical tomography applications point of view, the following may be mentioned:

- an easy and precise way to represent a source point (see Eq. (4.93)),
- more easy to generate a good quality 3D boundary mesh than 3D volume one,
- adaptive mesh refinement as easy as for 2D FEM problems,
- BEM guarantees high and a constant precision of the solution inside the whole region under consideration for both quantities: Φ and the gradient of Φ.

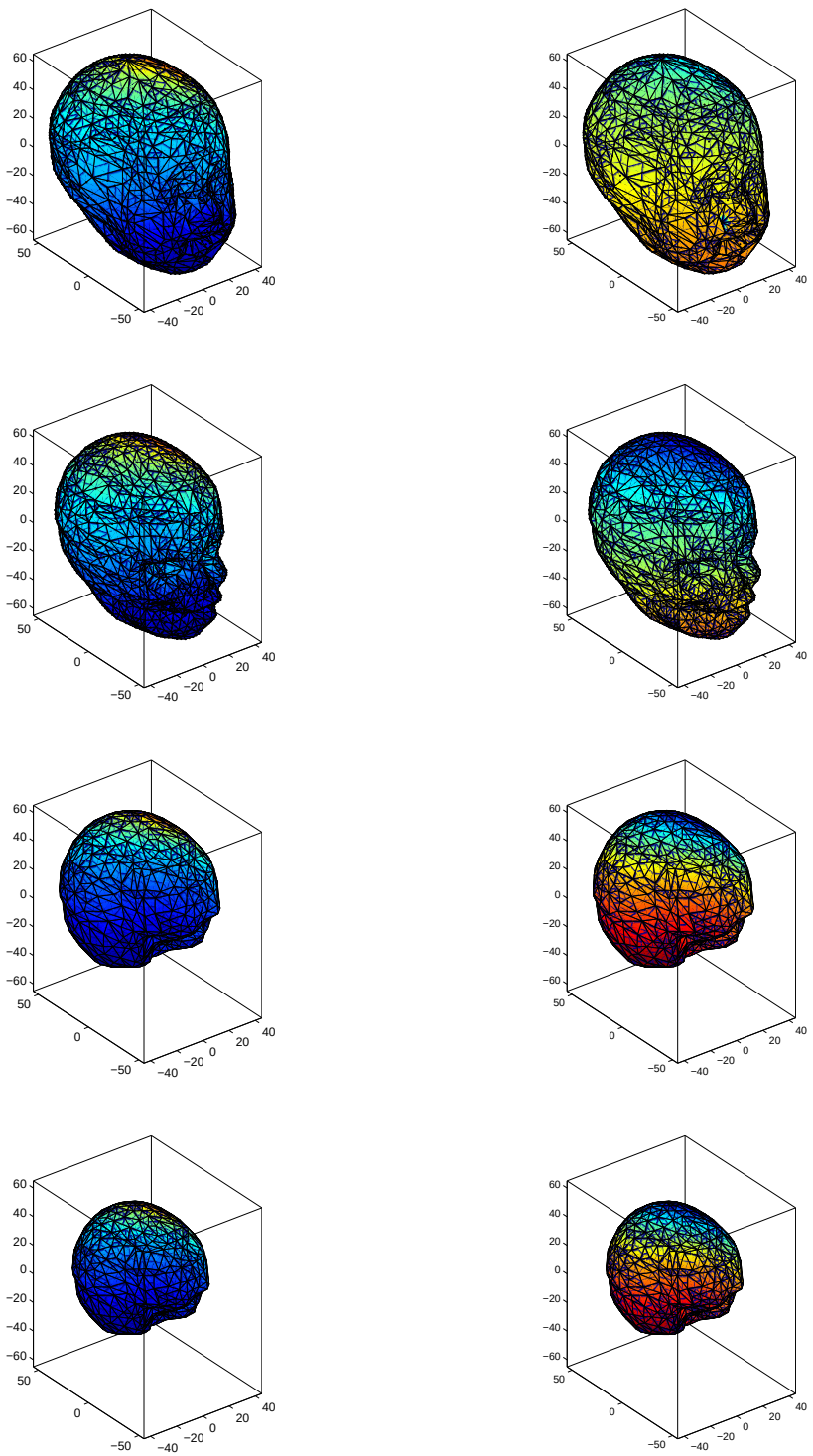


Figure 4.34: Four layer baby head model without a void gap; on the left amplitude, on the right phase

4.12 Light propagation in diffusive media with non-scattering regions

The main purpose of this chapter is to develop an efficient method for mathematical modelling the baby's head with clear layers. Previously such a model was created using almost exclusively the FEM, see for example [4, 9, 8, 7, 25, 38, 37, 39, 41, 40]. In those 2D or 3D models the region under consideration was treated as spatially homogeneous, in order to avoid too many finite elements.

So, why do not apply the BEM, which in this particular case provide almost the same ability, regarding discretization, and has advantages over FEM, as mentioned before?

Fig. 4.35 illustrate two types of void and the hollow case and the thin layer will be discussed in this chapter. These have been chosen as they are analogous to anatomical features one might expect to see in brain, the hollow regions to the ventricles and the thin layer being analogous to the Sub-Arachnoid Space (SAS), as illustrated in Fig. 4.36.

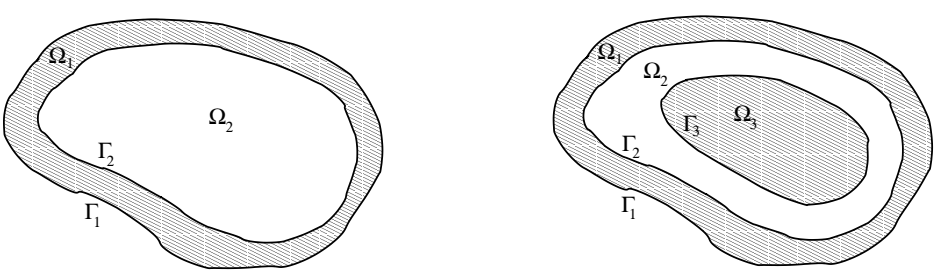


Figure 4.35: A diagram illustrating two types of void inclusion a hollow region (left) and a thin layer region (right)

4.12.1 Governing equations for non-scattering sphere embedded in a diffusive spherical region

Let consider a spherical domain Ω with boundary Γ . The Boundary Element Method (BEM) proceeds by applying Green theorem in its second form [10] to derive an integral equation applicable on the surface $\Gamma = \Gamma_1 \cup \Gamma_2$ (see

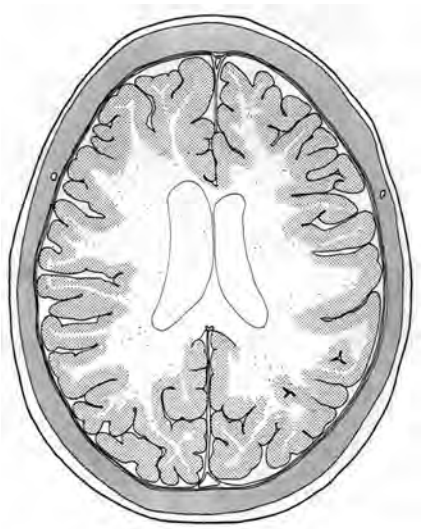


Figure 4.36: A diagram illustrating the biological analogies of the void types

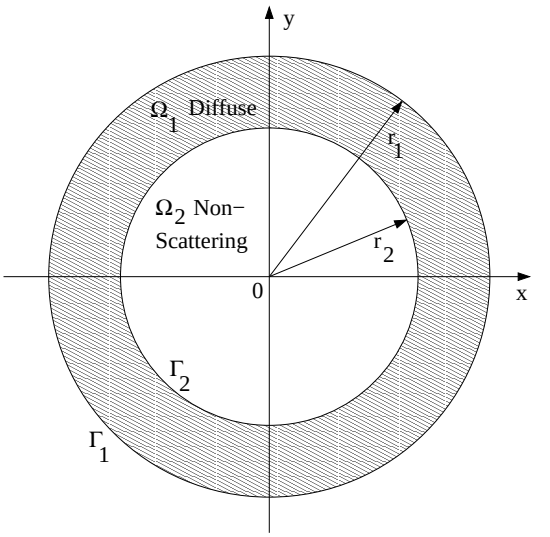


Figure 4.37: The cross-section of the region under consideration with a non-scattering sub-region

Fig. 4.37):

$$c(\mathbf{r})\Phi(\mathbf{r}) \quad + \quad \int_{\Gamma} \frac{\partial G(|\mathbf{r}-\mathbf{r}'|)}{\partial n}\Phi(\mathbf{r}')d\Gamma(\mathbf{r}') = \tag{4.96}$$

$$= \quad - \quad \int_{\Gamma} G(|\mathbf{r}-\mathbf{r}'|)\frac{J_n(\mathbf{r}')}{D}d\Gamma(\mathbf{r}') - \int_{\Omega} q_0 G(|\mathbf{r}-\mathbf{r}'|)d\Omega$$

$$\forall \qquad \mathbf{r} \in \Gamma_1 \cup \Gamma_2, \ \ \forall \quad \mathbf{r}' \in \Gamma_1 \cup \Gamma_2\,,$$

where: $-\frac{J_n(\mathbf{r}')}{D} = \frac{\partial\Phi(\mathbf{r}')}{\partial n}$ and $c(\mathbf{r}) = 1/2$ when the observation point lies on a smooth surface, which is the case considered here. The Green function, photon density function Φ and photon's current function J_n are depended on ω, but in order to simplify the equation, this symbol was skipped.

It is worth to notice that the photon's current function J_n has an opposite direction to the outward normal unit vector $\mathbf{n}$ and to the $\frac{\partial\Phi}{\partial n}$. It means that it is directed inside the region.

Boundary conditions on the boundary Γ_1 are usually referred to the zero flux or partial flux boundary conditions [39], and by means of Fick law,

$J(\mathbf{r}) = -D\nabla\Phi$ can be rewritten as:

$$\Phi_{\Gamma_1}(\mathbf{r}) + \alpha D \frac{\partial \Phi_{\Gamma_1}(\mathbf{r})}{\partial n} = \Phi_{\Gamma_1}(\mathbf{r}) - \alpha J_{n\Gamma_1} = 0, \qquad \forall \quad \mathbf{r} \in \Gamma_1\,, \tag{4.97}$$

where coefficient $\alpha = (2 - R_J)/R_U$ depends on a refractive index mismatch. We have assumed that $R_J = 1$ and $R_U = 1/2$ so $\alpha = 2$ in case of a refractive index matched [39]. The variables Φ_{Γ_i} and $J_{n\Gamma_i}$ indicates to which part of boundary Γ they belong.

On the other part of the boundary $\Gamma = \Gamma_2$ the non-local boundary condition between the diffusive and non-scattering media is imposed [39, 41]:

$$\begin{aligned}\Phi_{\Gamma_2}(\mathbf{r}) &= \alpha J_{n\Gamma_2}(\mathbf{r}) + \\ &+ \frac{1}{\pi} \int\limits_{\Gamma_2} \left[\Phi_{\Gamma_2}(\mathbf{r}') + \frac{R_J}{R_U} J_{n\Gamma_2}(\mathbf{r}') \right] \mathcal{G}(|\mathbf{r} - \mathbf{r}'|) d\Gamma_2(\mathbf{r}') \;\; \forall \;\; \mathbf{r} \in \Gamma_2\,,\end{aligned} \tag{4.98}$$

where $\mathcal{G}$ is the radiocity kernel representing diffuse-diffuse propagation of light in free space which will be explained in details later.

We can rewrite previous equation in a standard BEM way as follows:

$$\begin{aligned}\frac{1}{2}\Phi_{\Gamma_2}(\mathbf{r}) - \int\limits_{\Gamma_2} \frac{1}{2\pi} \mathcal{G}(|\mathbf{r} - \mathbf{r}'|) \Phi_{\Gamma_2}(\mathbf{r}') d\Gamma_2(\mathbf{r}') = \\ = J_{n\Gamma_2}(\mathbf{r}) + \int\limits_{\Gamma_2} \frac{1}{2\pi} \mathcal{G}(|\mathbf{r} - \mathbf{r}'|) 2 J_{n\Gamma_2}(\mathbf{r}') d\Gamma_2(\mathbf{r}') \quad \forall \quad \mathbf{r} \in \Gamma_2\,,\end{aligned} \tag{4.99}$$

where:

$$\frac{1}{2\pi} \mathcal{G}(|\mathbf{r} - \mathbf{r}'|) = \frac{1}{2\pi} \Gamma_\omega(|\mathbf{r} - \mathbf{r}'|) \mathcal{V}(\mathbf{r} - \mathbf{r}') \cos(\Theta)\,. \tag{4.100}$$

In the above equation $\mathcal{V}(\mathbf{r} - \mathbf{r}')$ is a Boolean visibility function:

$$\mathcal{V}(\mathbf{r} - \mathbf{r}') = \begin{cases} 1 & \text{if } \mathbf{r} \text{ and } \mathbf{r}' \text{ are mutually visible,} \\ 0 & \text{otherwise.} \end{cases} \tag{4.101}$$

In the considered case (see Fig. 4.37) $\mathcal{V}(\mathbf{r} - \mathbf{r}')$ is equal to unity so can be skipped, and Γ_ω is expressed as follows:

$$\Gamma_\omega(|\mathbf{r} - \mathbf{r}'|) = \frac{e^{[-\mu_{a0} + i(\omega n_0/c)]|\mathbf{r} - \mathbf{r}'|}}{|\mathbf{r} - \mathbf{r}'|^2} \cos(\Theta')\,, \tag{4.102}$$

where:

$$\cos(\Theta) = -\mathbf{n} \cdot \frac{(\mathbf{r} - \mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|}\,, \qquad \cos(\Theta') = -\mathbf{n}' \cdot \frac{(\mathbf{r}' - \mathbf{r})}{|\mathbf{r} - \mathbf{r}'|}\,, \tag{4.103}$$

where $\mathbf{n}$ is the unit outward normal vector as indicated in the Fig. 4.38.

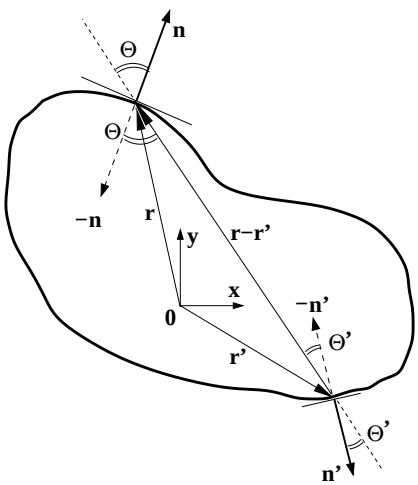


Figure 4.38: Angles illustration for 2D space

The component $\frac{\cos(\Theta)\cos(\Theta')}{\pi|\mathbf{r}-\mathbf{r}'|^2}$ is often reffered to as the 'Form Factor' (see section 4.13).

For concentrated sources for which the function q_0 is defined at the internal point $\mathbf{r}_s$ we can write:

$$q_0 = Q_s \delta(\mathbf{r}_s)\,, \tag{4.104}$$

where Q_s is the magnitude of the point source and $\delta(\mathbf{r}_s)$ is a Dirac delta function whose integral is equal to 1 over any volume containing the singularity point $\mathbf{r}_s$.

Assuming that a number n_s of these functions exists, the Eq. (4.96) , Eq. (4.97)

and Eq. (4.99) are making the system of equations describing our problem:

$$\begin{aligned}
&\frac{1}{2}\Phi_{\Gamma_i}(\mathbf{r}) + \int\limits_{\Gamma} \frac{\partial G(|\mathbf{r}-\mathbf{r}'|)}{\partial n}\Phi_{\Gamma_i}(\mathbf{r}')d\Gamma(\mathbf{r}') = \\
= \; & -\int\limits_{\Gamma} G(|\mathbf{r}-\mathbf{r}'|)\frac{J_{n\Gamma_i}(\mathbf{r}')}{D}d\Gamma(\mathbf{r}') - \sum_{s=0}^{n_s-1} G(|\mathbf{r}-\mathbf{r}_s|)Q_s \qquad (4.105)\\
\forall \quad & \mathbf{r} \in \Gamma_1 \cup \Gamma_2, \;\; \forall \quad \mathbf{r}' \in \Gamma_1 \cup \Gamma_2, \;\; \forall \quad \mathbf{r}_s \in \Omega_1 \,,
\end{aligned}$$

$$\frac{1}{2}\Phi_{\Gamma_1}(\mathbf{r}) = J_{n\Gamma_1}(\mathbf{r}) \quad \forall \quad \mathbf{r} \in \Gamma_1 \,, \qquad (4.106)$$

and:

$$\begin{aligned}
&\frac{1}{2}\Phi_{\Gamma_2}(\mathbf{r}) - \int\limits_{\Gamma_2} \frac{1}{2\pi}\mathcal{G}(|\mathbf{r}-\mathbf{r}'|)\Phi_{\Gamma_2}(\mathbf{r}')d\Gamma_2(\mathbf{r}') = \\
= J_{n\Gamma_2}(\mathbf{r}) + & \int\limits_{\Gamma_2} \frac{1}{2\pi}\mathcal{G}(|\mathbf{r}-\mathbf{r}'|)2J_{n\Gamma_2}(\mathbf{r}')d\Gamma_2(\mathbf{r}') \quad \forall \quad \mathbf{r} \in \Gamma_2 \,, \qquad (4.107)
\end{aligned}$$

where $\Phi_{\Gamma_i}(\mathbf{r}) = \Phi(\mathbf{r})|_{\Gamma_i}$, $J_{n\Gamma_i}(\mathbf{r}) = J_n(\mathbf{r})|_{\Gamma_i}$ and $i = 1, 2$.

If we assume that sphere Γ_1 and sphere Γ_2 have the same number of N unknowns then the total number of unknowns is equal to $4N$.

4.12.2 Matrix form of integral equations

Equations Eq. (4.105) to Eq. (4.107) can be presented in a matrix form:

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \\ \mathbf{A}_{31} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{42} \end{bmatrix} \begin{bmatrix} \mathbf{\Phi}_{\Gamma_1} \\ \mathbf{\Phi}_{\Gamma_2} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \\ \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_{42} \end{bmatrix} \begin{bmatrix} \mathbf{J}_{\mathbf{n}\Gamma_1} \\ \mathbf{J}_{\mathbf{n}\Gamma_2} \end{bmatrix} + \begin{bmatrix} \mathbf{q_1} \\ \mathbf{q_2} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} . \qquad (4.108)$$

By including the Robin boundary conditions (Eq. (4.106)) to Eq. (4.105), we can reduce the number of unknowns to the values $\mathbf{\Phi}_{\Gamma_1}(\mathbf{r})$, $\mathbf{\Phi}_{\Gamma_2}(\mathbf{r})$ and $\mathbf{J}_{\mathbf{n}\Gamma_2}(\mathbf{r})$ only.

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \\ \mathbf{0} & \mathbf{A}_{42} \end{bmatrix} \begin{bmatrix} \mathbf{\Phi}_{\Gamma_1} \\ \mathbf{\Phi}_{\Gamma_2} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{11}\mathbf{A}_{31} & \mathbf{B}_{12} \\ \mathbf{B}_{21}\mathbf{A}_{31} & \mathbf{B}_{22} \\ \mathbf{0} & \mathbf{B}_{42} \end{bmatrix} \begin{bmatrix} \mathbf{\Phi}_{\Gamma_1} \\ \mathbf{J}_{\mathbf{n}\Gamma_2} \end{bmatrix} + \begin{bmatrix} \mathbf{q_1} \\ \mathbf{q_2} \\ \mathbf{0} \end{bmatrix} . \quad (4.109)$$

If the first column of the right-hand matrix is transferred to the left-hand side, the unknown vector $\mathbf{J}_{\mathbf{n}\Gamma_2}$ would be included to the left hand side $\mathbf{\Phi}_{\Gamma_1}$ and $\mathbf{\Phi}_{\Gamma_2}$ unknowns vectors. Finally we come to the form:

$$\begin{bmatrix} \mathbf{A}_{11} - \mathbf{B}_{11}\mathbf{A}_{31} & \mathbf{A}_{12} & -\mathbf{B}_{12} \\ \mathbf{A}_{21} - \mathbf{B}_{21}\mathbf{A}_{31} & \mathbf{A}_{22} & -\mathbf{B}_{22} \\ \mathbf{0} & \mathbf{A}_{42} & -\mathbf{B}_{42} \end{bmatrix} \begin{bmatrix} \mathbf{\Phi}_{\Gamma_1} \\ \mathbf{\Phi}_{\Gamma_2} \\ \mathbf{J}_{\mathbf{n}\Gamma_2} \end{bmatrix} = \begin{bmatrix} \mathbf{q_1} \\ \mathbf{q_2} \\ \mathbf{0} \end{bmatrix} . \quad (4.110)$$

4.13 The Form Factor

For the Form Factor enclosed in Eq. (4.100) we will now develop analytic expressions and numerical approach for the case presented in Fig. 4.37.

$$\frac{1}{2\pi}\mathcal{G}(|\mathbf{r}-\mathbf{r}'|) = \frac{e^{\left(-\mu_{a0}+\frac{i\omega n_0}{c}\right)|\mathbf{r}-\mathbf{r}'|}}{2} \frac{\cos(\Theta)\cos(\Theta')}{\pi|\mathbf{r}-\mathbf{r}'|^2} = \frac{e^{\left(-\mu_{a0}+\frac{i\omega n_0}{c}\right)|\mathbf{r}-\mathbf{r}'|}}{2} g(u,v), \quad (4.111)$$

where $g(u,v)$ is the standard Form Factor expression; u and v are the standard parametric representation for a sphere ($u \in <\ 0, \pi >$ and $v \in <\ 0, 2\pi >$) [36]:

$$g(u,v) = \frac{\cos(\Theta)\ \cos(\Theta')}{\pi|\mathbf{r}-\mathbf{r}'|^2} \ . \quad (4.112)$$

4.13.1 The Form Factor calculated analytically

The Form Factor (4.112) for a sphere of radius r_2 (see Fig. 4.37 and consult Fig. 4.38) can be easily calculated analytically:

$$g(u,v) = \frac{\cos(\Theta)\ \cos(\Theta')}{\pi|\mathbf{r}-\mathbf{r}'|^2} = \frac{1}{4\pi r_2^2} \ . \quad (4.113)$$

The above relation significantly simplifies calculation due to the fact that integrals in Eq. (4.107) are not singular any more.

4.13.2 The Form Factor calculated numerically

In case of arbitrary shaped surfaces there is a need to calculate the form factor numerically. That means (see Eq. (4.112)) that not only both cosine functions have to be calculated numerically (Eq. (4.103), but also a visibility function.

$$\begin{aligned} \cos(\Theta) &= \mathbf{n} \cdot \frac{(\mathbf{r}-\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} = \frac{n_x|\mathbf{r}-\mathbf{r}'|_x + n_y|\mathbf{r}-\mathbf{r}'|_y + n_z|\mathbf{r}-\mathbf{r}'|_z}{|\mathbf{r}-\mathbf{r}'|}\,, \\ \cos(\Theta') &= \mathbf{n}' \cdot \frac{(\mathbf{r}'-\mathbf{r})}{|\mathbf{r}-\mathbf{r}'|} = \frac{n'_x|\mathbf{r}'-\mathbf{r}|_x + n'_y|\mathbf{r}'-\mathbf{r}|_y + n'_z|\mathbf{r}'-\mathbf{r}|_z}{|\mathbf{r}'-\mathbf{r}|}\,. \end{aligned} \tag{4.114}$$

For an isotropic second order triangle the normal vector components in BEM can be expressed by Eq. (5.9).

Regarding the numerical calculation of the visibility function we have already tried two basic approaches. The first one when the visibility function (basically the continuous function) was calculated in each Gaussian numerical integration point and the second one where was calculated only in the nodes of the boundary elements. Both approaches producing almost identical results. However, the first approach seems to be more efficient numerically, so only this approach will be exploited in the following sections.

4.14 Non-scattering gap between two diffusive regions of a spherical shape

Let study the three-dimensional configuration depicted in Fig. 4.40, where the outer sphere of fixed radius $r_1 = 25$ mm, filled with a diffusive medium of parameters $\mu'_s = 1\,\mathrm{mm}^{-1}$ and $\mu_{a1} = 0.01\,\mathrm{mm}^{-1}$ with a non-scattering gap of outer radius $r_2 = 20$ mm and inner radius $r_3 = 17$ mm between two diffusive regions. The absorption coefficient for the non-scattering region is $\mu_{a0} = 0.005\,\mathrm{mm}^{-1}$. In this case the visibility factor $\mathcal{V}(\mathbf{r}-\mathbf{r}')$ is either unity if both points $\mathbf{r}$ and $\mathbf{r}'$ can be joined by a straight line without intersecting an interface (i.e., when they are visible to each other) or zero when such a straight line does not exist. For the case of concentric spheres it may be calculated analytically.

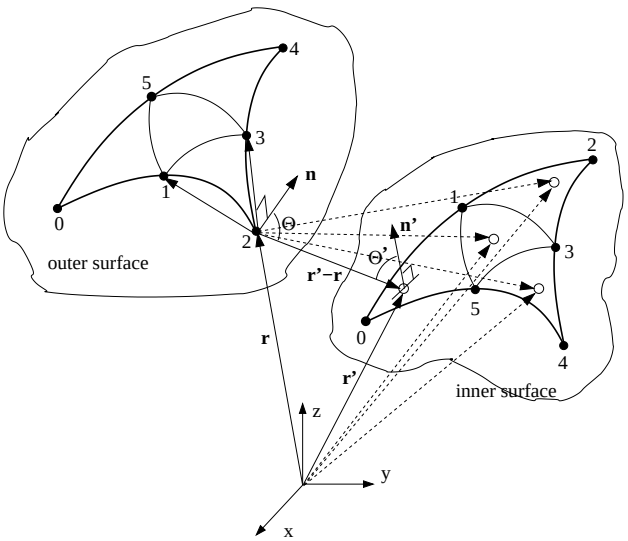


Figure 4.39: A diagram illustrating the way of the Form Factor numerical calculations (doted lines denote the node to Gaussian node visibility matrix calculation)

4.14.1 Form Factor calculated analytically

We now attempt to derive the analytical Form Factor for more interesting case then the one presented in the previous section, namely that of two concentric spheres, outer sphere radius r_2 and inner sphere radius r_3 [38]. Considering any point on the outer sphere we have two cases:

1. the visible point lies on the inner sphere,
2. the visible point lies on the outer sphere.

In case when the visible point is laying on the inner surface the analytical expression for the Form Factor becomes (see Fig. 4.41):

$$g(u,v) = \frac{\cos(\Theta)\,\cos(\Theta')}{\pi|\mathbf{r}-\mathbf{r}'|^2} = \frac{(r_2 - r_3\cos U)(r_3 - r_2\cos U)}{\pi(r_2^2 + r_3^2 - 2r_2r_3\cos U)^2}, \qquad (4.115)$$

where:

$$\cos\Theta = \frac{r_2 - r_3\cos U}{\sqrt{r_2^2 + r_3^2 - 2r_2r_3\cos U}}, \qquad \cos\Theta' = \frac{r_3 - r_2\cos U}{\sqrt{r_2^2 + r_3^2 - 2r_2r_3\cos U}} \qquad (4.116)$$

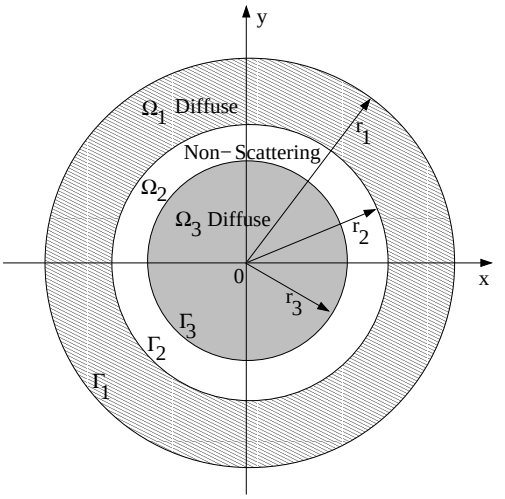


Figure 4.40: The cross-section of the region with a non-scattering gap

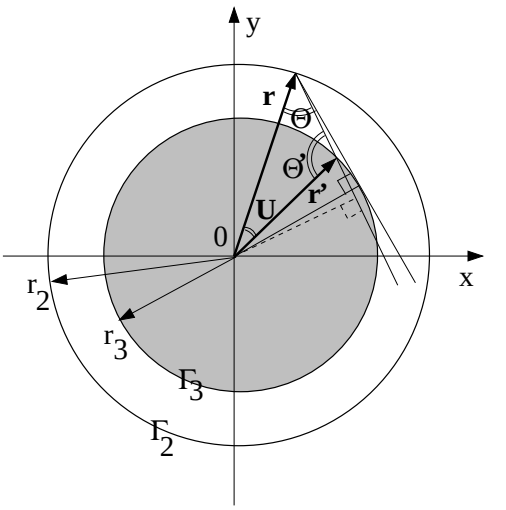


Figure 4.41: Analytical calculation of the Form Factor for the case number 1

and:

$$\cos U = \frac{\mathbf{r} \cdot \mathbf{r}'}{|\mathbf{r}||\mathbf{r}'|}. \tag{4.117}$$

But for the second case, when the visible point is on the outer surface the expression remains the same as in Eq. (4.113). The visibility matrix image is shown in Fig. 4.42.

4.14.2 Visibility function calculated analytically

If we are dealing with a regular surfaces as spheres, then the visibility factor (ref. Eq. (4.100)) may be calculated analytically. Clearly, for a hollow sphere the visibility function is equal to 1 because all points (nodes of boundary elements) can see all other points. But for the gap between a concentric spheres (see Fig. 4.40) it is not that simple. We have to consider three cases:

1. both points $\mathbf{r}$ and $\mathbf{r}'$ are on the surface Γ_2 (see Fig. 4.43-left) – are **visible** if $\alpha_{vis} < 2U_{co}$. Angle U_{co} is the visibility cut-off angle for this case,

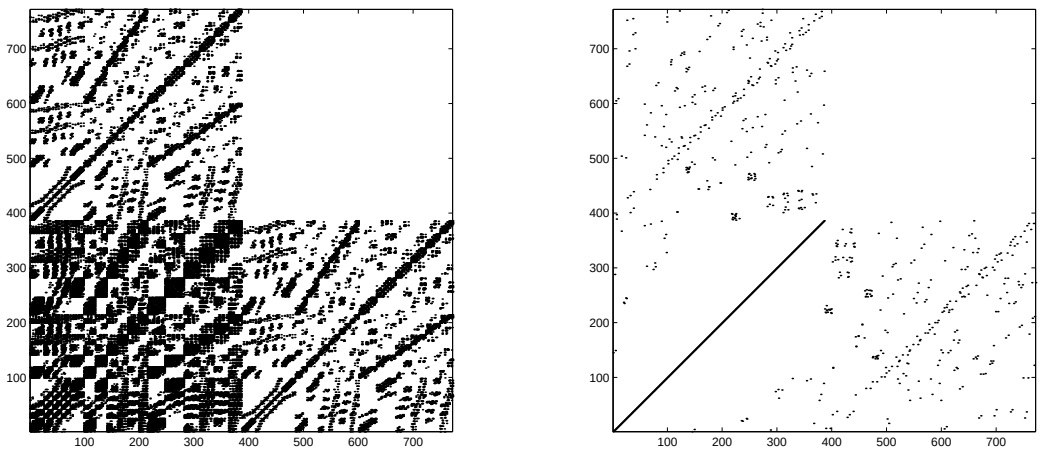


Figure 4.42: Visibility matrix image calculated analytically (left) and a difference between the matrix calculated analytically and calculated numerically (right)

2. one of the points is on the surface Γ_2 and the other on the surface Γ_3 (see Fig. 4.43-right) – are **visible** if $\alpha_{vis} < U_{co}$,

3. both points $\mathbf{r}$ and $\mathbf{r}'$ are on the surface Γ_3 – are **not visible**.

It is now possible to calculate the visibility cut-of ($U_{co}|_{\Gamma_2}$ and $U_{co}|_{\Gamma_3}$) for concentric spheres geometry. From geometry (see Fig. 4.40) we have that $U_{co}|_{\Gamma_3} = \frac{1}{2}\, U_{co}|_{\Gamma_2} = \arccos(\frac{r_3}{r_2})$, giving us a simple expression for the point to point visibility for all points on the domain boundary in this case.

The angle between two arbitrarily chosen points $\mathbf{r}$ and $\mathbf{r}'$ is equal

$$\alpha_{vis} = \arccos \frac{\mathbf{r} \cdot \mathbf{r}'}{|\mathbf{r}||\mathbf{r}'|} \quad . \tag{4.118}$$

4.14.3 Visibility function calculated numerically

Visibility function calculated numerically leads to the following basic problem. We have to find the intersection point (if it exists) between a line segment defined by two points $\mathbf{r}$ and $\mathbf{r}'$ and a planar three vertex facet. Each boundary element is subdivided by four flat facets in order to simplify the procedure (see Fig. 4.44). At the beginning we will adopt the brutal force method which rely on checking if every sub-triangle has an intersection point with the line segment defined by two points mentioned before.

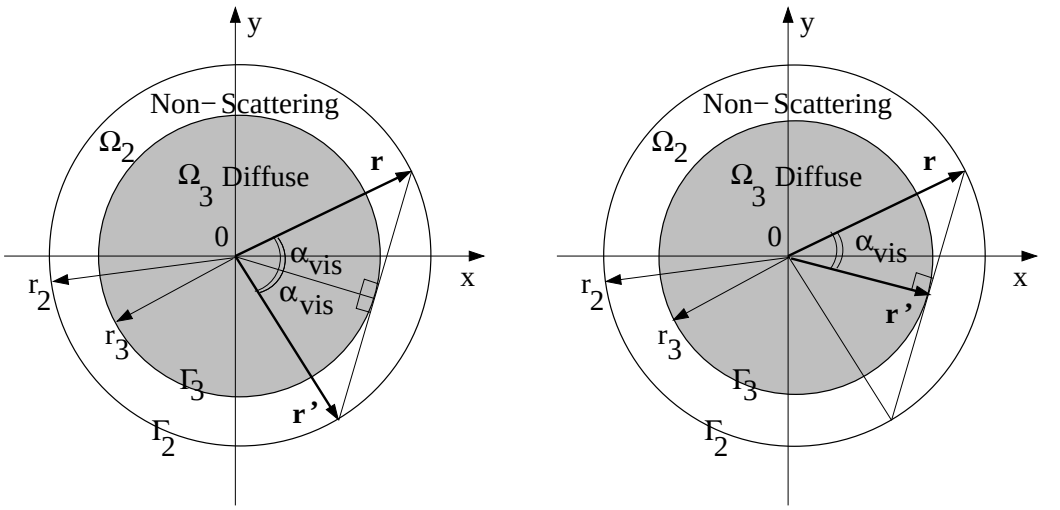


Figure 4.43: The visibility angle between two points on the Γ_2 surface (left) and between two points on Γ_1 and Γ_2 surface (right)

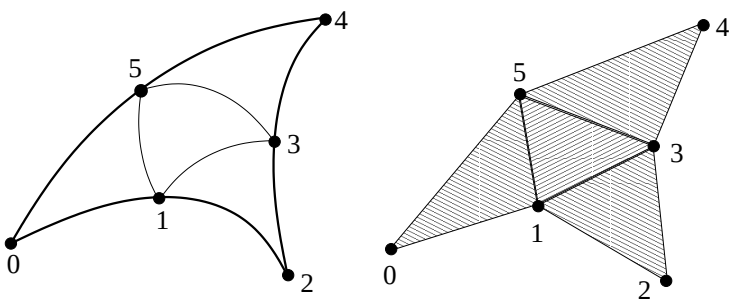


Figure 4.44: The boundary element subdivided by four flat sub-triangles

The solution involves the following steps:

1. check that the line and the plane are not parallel,
2. find the intersection of the line, on which the given line segment lies, with the plane containing the facet,
3. check that the intersection point lies along the line segment,
4. check that the intersection point lies within the sub-triangle.

The intersection point $\mathbf{r}_i$ is found by substituting the equation for the line

$\mathbf{r}_i = \mathbf{r} - u(\mathbf{r} - \mathbf{r}')$ by the general form of equation for the plane $Ax + By + Cz + D = 0$.

Note that the values of A, B, C are the components of the normal to the plane which can be found by taking the cross product of any two normalized edge vectors and then D is found by substituting one vertex into the equation for the plane.

$$\begin{aligned} A &= z_1(y_3 - y_2) + z_2(y_1 - y_3) + z_3(y_2 - y_1), \\ B &= z_1(x_2 - x_3) + z_2(x_3 - x_1) + z_3(x_1 - x_2), \\ C &= x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2), \\ D &= -x_3A - y_3B - z_3C, \end{aligned} \tag{4.119}$$

where three vertices $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{r}_3$ determine the plane.

This gives an expression for u from which the point of intersection can be found.

$$u = \frac{Ax + By + Cz + D}{A(x - x') + B(y - y') + C(z - z')}. \tag{4.120}$$

If the denominator above is equal to 0 then the line is parallel to the plane and they do not intersect. For the intersection point to lie on the line segment, u must be between 0 and 1. Those conditions allow us an early return.

And last but not least, we need to check whether or not the intersection point lies within the triangle.

If the point of intersection is inside of the boundary element than the point $\mathbf{r}$ and point $\mathbf{r}'$ are not visible.

4.14.4 Point in triangle test

Boundary elements mesh usually is quite dense so the crucial point of a test is its efficiency. We will constrain ourself to a triangular elements only due to the fact that they are the most popular and every quadrilateral boundary element can be split into two triangular elements.

A common way to check whether a point is in triangle, is to find the vectors connecting the point to each of the triangle's three vertices and sum the angles between those vectors.

If sum of the angles is 2π then the point is inside the triangle, otherwise it is not. It works, but it is very slow. We need a faster and much easier method [51].

Set of the points, let them call A, B and C forms the triangle and lines AB, BC and CA each split space in half and one of those halves is entirely outside the triangle. That is what we will take advantage of.

For a point to be inside the triangle ABC it must be below AB and left of BC and right of AC. If any one of these conditions fails we can return early. In order to test whether the point is on the correct side of the line we have

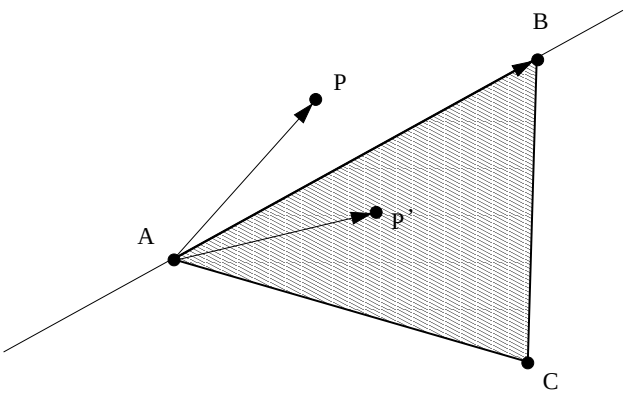


Figure 4.45: Illustration of the point in triangle test

to take the cross product of $\overrightarrow{AB}$ and $\overrightarrow{AP}$ (see Fig. 4.45), we will get a vector pointing out of the plane. On the other hand, if we take the cross product of $\overrightarrow{AB}$ and $\overrightarrow{AP'}$, we will get a vector pointing into the plane.

The question is what direction the cross product should point in? Because the triangle can be oriented in any way in the 3D space, we need some reference point – a point that we know is on a certain side of the line. In our case is just the third point C.

So, any point P where $\overrightarrow{AB}$ cross $\overrightarrow{AP}$ does not point in the same direction as $\overrightarrow{AB}$ cross $\overrightarrow{AC}$ is not inside the triangle. If the cross products do point in the same direction, than we need to test P with the other lines as well.

If the point was on the same side of AB as C and is on the same side of BC as A and on the same side of CA as B, then it is in the triangle.

We can implement this algorithm in the following way:

```
(Test Point in Triangle.)
  function SameSide(p1,p2, a,b)
    cp1 = CrossProduct(a-b, a-p1)
    cp2 = CrossProduct(a-b, a-p2)
    if DotProduct(cp1, cp2) >= 0 then return true
    else return false
  function PointTriangle(p, a,b,c)
    if SameSide(p,a, b,c) && SameSide(p,b, a,c)
    && SameSied(p,c, a,b) then return true
    else return false
```

This algorithm is simple, effective and has no square roots and the other functions like arc cosines etc.

4.14.5 Integral equations

In case of three surfaces immersed one in the other, as it is presented in Fig. 4.40, the system of equations will be more complicated:

$$\frac{1}{2}\Phi_{\Gamma_i}(\mathbf{r}) + \int\limits_{\Gamma} \frac{\partial G(|\mathbf{r}-\mathbf{r}'|)}{\partial n}\Phi_{\Gamma_i}(\mathbf{r}')d\Gamma(\mathbf{r}') =$$

$$= -\int\limits_{\Gamma} G(|\mathbf{r}-\mathbf{r}'|)\frac{J_{n\Gamma_i}(\mathbf{r}')}{D}d\Gamma(\mathbf{r}') - \sum_{s=0}^{n_s-1} G(|\mathbf{r}-\mathbf{r}_s|)Q_s \qquad (4.121)$$

$$\forall \quad \mathbf{r} \in \Gamma_1 \cup \Gamma_2, \ \forall \quad \mathbf{r}' \in \Gamma_1 \cup \Gamma_2, \ \forall \quad \mathbf{r}_s \in \Omega_1\,,$$

$$\frac{1}{2}\Phi_{\Gamma_1}(\mathbf{r}) = J_{n\Gamma_1}(\mathbf{r}) \quad \forall \quad \mathbf{r} \in \Gamma_1\,, \qquad (4.122)$$

$$\frac{1}{2}\Phi_{\Gamma_V}(\mathbf{r}) - \int\limits_{\Gamma_V} \frac{1}{2\pi}\mathcal{G}(|\mathbf{r}-\mathbf{r}'|)\Phi_{\Gamma_V}(\mathbf{r}')d\Gamma_V(\mathbf{r}') =$$

$$= J_{n\Gamma_V}(\mathbf{r}) + \int\limits_{\Gamma_V} \frac{1}{2\pi}\mathcal{G}(|\mathbf{r}-\mathbf{r}'|)2J_{n\Gamma_V}(\mathbf{r}')d\Gamma_V(\mathbf{r}')$$

$$\forall \quad \mathbf{r} \cup \mathbf{r}' \in \Gamma_V = \Gamma_2 \cup \Gamma_3\,, \qquad (4.123)$$

and:

$$\frac{1}{2}\Phi_{\Gamma_2}(\mathbf{r}) + \int\limits_{\Gamma_3} \frac{\partial G(|\mathbf{r}-\mathbf{r}'|)}{\partial n}\Phi_{\Gamma_2}(\mathbf{r}')d\Gamma_3(\mathbf{r}') =$$

$$= -\int\limits_{\Gamma_3} G(|\mathbf{r}-\mathbf{r}'|)\frac{J_{n\Gamma_2}(\mathbf{r}')}{D}d\Gamma_3(\mathbf{r}') \quad \forall \quad \mathbf{r}\cup\mathbf{r}' \in \Gamma_3\,. \tag{4.124}$$

4.14.6 Matrix form of integral equations

Equations Eq. (4.121) to the Eq. (4.124) can be presented in a matrix form:

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{0} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{0} \\ \mathbf{A}_{31} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{42} & \mathbf{A}_{43} \\ \mathbf{0} & \mathbf{A}_{52} & \mathbf{A}_{53} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_{63} \end{bmatrix} \begin{bmatrix} \mathbf{\Phi}_{\Gamma_1} \\ \mathbf{\Phi}_{\Gamma_2} \\ \mathbf{\Phi}_{\Gamma_3} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} & \mathbf{0} \\ \mathbf{B}_{21} & \mathbf{B}_{22} & \mathbf{0} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_{52} & \mathbf{B}_{53} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}_{63} \end{bmatrix} \begin{bmatrix} \mathbf{J}_{\mathbf{n}\Gamma_1} \\ \mathbf{J}_{\mathbf{n}\Gamma_2} \\ \mathbf{J}_{\mathbf{n}\Gamma_3} \end{bmatrix} + \begin{bmatrix} \mathbf{q_1} \\ \mathbf{q_2} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{4.125}$$

By including the Robin boundary conditions from Eq. (4.122) to the Eq. (4.121) we can reduce the number of unknowns to the values $\mathbf{\Phi}_{\Gamma_1}(\mathbf{r})$, $\mathbf{\Phi}_{\Gamma_2}(\mathbf{r})$, $\mathbf{\Phi}_{\Gamma_3}(\mathbf{r})$, $\mathbf{J}_{\mathbf{n}\Gamma_2}(\mathbf{r})$ and $\mathbf{J}_{\mathbf{n}\Gamma_3}(\mathbf{r})$ only.

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{0} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{42} & \mathbf{A}_{43} \\ \mathbf{0} & \mathbf{A}_{52} & \mathbf{A}_{53} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_{63} \end{bmatrix} \begin{bmatrix} \mathbf{\Phi}_{\Gamma_1} \\ \mathbf{\Phi}_{\Gamma_2} \\ \mathbf{\Phi}_{\Gamma_3} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{11}\mathbf{A}_{31} & \mathbf{B}_{12} & \mathbf{0} \\ \mathbf{B}_{21}\mathbf{A}_{31} & \mathbf{B}_{22} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_{42} & \mathbf{B}_{43} \\ \mathbf{0} & \mathbf{B}_{52} & \mathbf{B}_{53} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}_{63} \end{bmatrix} \begin{bmatrix} \mathbf{\Phi}_{\Gamma_1} \\ \mathbf{J}_{\mathbf{n}\Gamma_2} \\ \mathbf{J}_{\mathbf{n}\Gamma_3} \end{bmatrix} + \begin{bmatrix} \mathbf{q_1} \\ \mathbf{q_2} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{4.126}$$

If the first column of the right-hand matrix will be transferred to the left-hand side, then we have to rearrange the system of equations in such a way that unknown values should be on the left-hand side and the source terms on the right-hand side.

$$\begin{bmatrix} \mathbf{A}_{11}-\mathbf{B}_{11}\mathbf{A}_{31} & \mathbf{A}_{12} & -\mathbf{B}_{12} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{21}-\mathbf{B}_{21}\mathbf{A}_{31} & \mathbf{A}_{22} & -\mathbf{B}_{22} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{42} & -\mathbf{B}_{42} & \mathbf{A}_{43} & -\mathbf{B}_{43} \\ \mathbf{0} & \mathbf{A}_{52} & -\mathbf{B}_{52} & \mathbf{A}_{53} & -\mathbf{B}_{53} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{A}_{63} & -\mathbf{B}_{63} \end{bmatrix} \begin{bmatrix} \mathbf{\Phi}_{\Gamma_1} \\ \mathbf{\Phi}_{\Gamma_2} \\ \mathbf{J}_{\mathbf{n}\Gamma_2} \\ \mathbf{\Phi}_{\Gamma_3} \\ \mathbf{J}_{\mathbf{n}\Gamma_3} \end{bmatrix} = \begin{bmatrix} \mathbf{q_1} \\ \mathbf{q_2} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{4.127}$$

4.15 Results for the void gap

4.15.1 The steady state

In order to validate the steady state BEM solution, it was compared with well known Monte Carlo (MC) method. In the Fig. 4.46 there are outcomes of comparison between the MC and BEM results. In case of 5 mm gap the visibility cut-off angle is equal to 41.4^{o} and for 3 mm gap the visibility cut-off angle is equal to 31.7^{o}. Inspecting the figures we can conclude that discrepancies between BEM and MC methods still need more attention and investigation.

4.15.2 The frequency domain solution – 100MHz

For the frequency domain the same procedure was applied. This time the BEM results were compared not only with MC but also with FEM. As we can see FEM and BEM providing results which are very close each to the other. Descrepances with MC remains on the similar level as for the steady state. It could be explained by the fact that BEM and FEM are dealing with the approximation of the Boltzman equation but MC describe the physics of the problem. So far the MC solution is recognize asa reference solution.

4.15.3 Multilayered neonatal head model with the CSF layer

The multi-region strategy described in previous sections was employed. The target is to create the multilayered baby head numerical model taking into account the CSF layer. To achieve this goal we have started with the three layer spherical geometry first than with the three layer structural meshes generated on a baby's head. To achieve the structural meshes, parametric surfaces were generated with the aid of the algorithm based on spherical harmonics described in [54, 52].

To get a reference point, first the baby's head discretized by the structural mesh without a clear layer (Fig. 4.49 – left) and next with the clear layer (Fig. 4.49 – right).

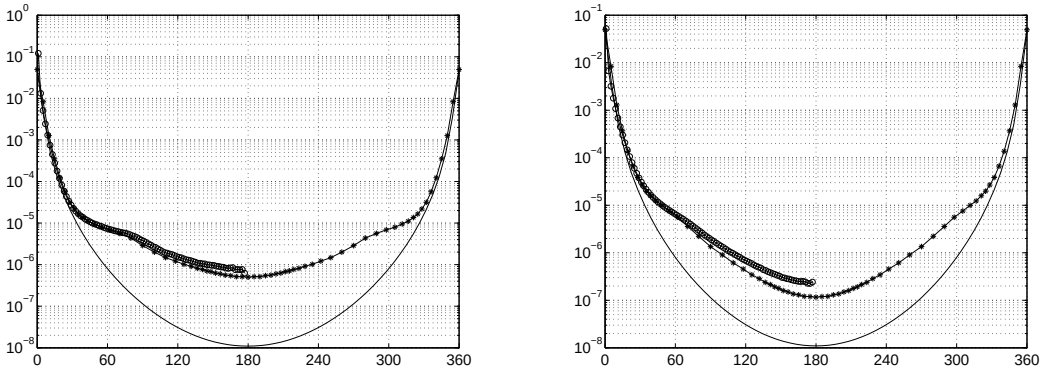


Figure 4.46: Amplitude distribution for a region with a non-scattering 5mm gap (left) and 3mm gap (right), comparison between MC (thick line) and BEM (dotted line)

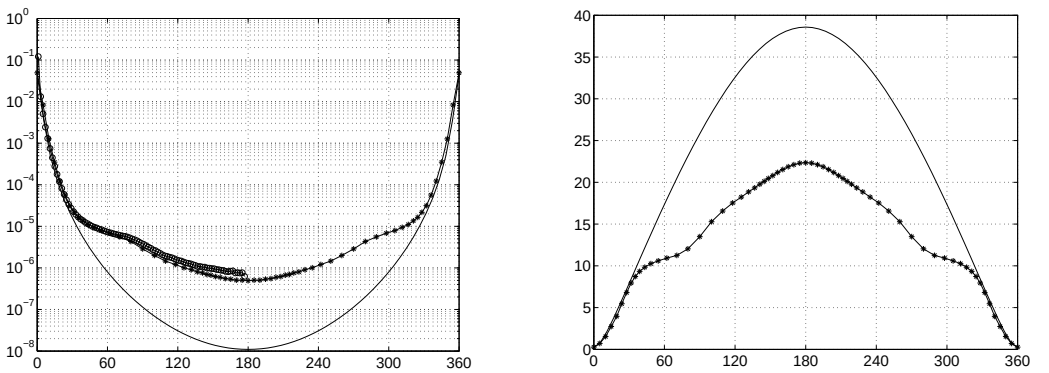


Figure 4.47: Amplitude and a phase shift distribution for a region with a non-scattering 5mm gap 100MHz, the thick line is a steady state MC, just to see a differences between the curves; the solid line is a diffusive region only

As we can see from Fig. 4.49, there is a big difference between both solutions. At the moment, the comparison is only done quantitatively as FEM or MC solutions do not exist for the baby's head so far.

4.15.4 Conclusion

The application of the boundary element method to the regions containing the non-scattering inclusions was presented in this chapter. Such an approach seems to be more natural than the finite element one, due to the fact that clear regions introduce to the discrete form of FEM a non-symmetric and

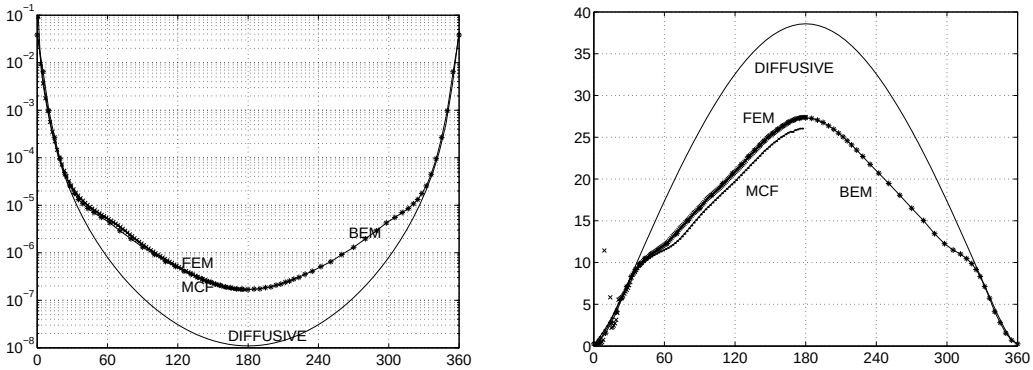


Figure 4.48: Amplitude and a phase shift distribution for a region with a non-scattering 3mm gap 100MHz, the solid line is a diffusive region only

fully populated sub-matrix, which cost of loosing the most desiring features of FEM like sparse and bounded coefficient matrix.

The clear region is not an disadvantage for the BEM and what is more, the non-local boundary conditions could be treated in the same way as the multi-region BEM but with a different function as a fundamental solution (Eq. (4.123)).

All this, as well as the more simple procedure of discretisation of the surfaces against those ones for the volume discretisation, the Authors are convinced to the conclusion that the BEM might be an efficient and flexible tool for neonatal baby's head modelling in Optical Tomography.

4.16 Domain Decomposition Method for multilayered spherical model

The image reconstruction problem in DOT is a non-linear ill-posed problem which requires feasible forward model that describe light propagation within the medium as accurately as possible. The forward problem in DOT can be modelled in an frequency domain as a diffusion equation (Helmholtz equation) with Robin boundary conditions. Additionally, in case of multilayered geometries the forward problem can be treated as a set of coupled equations what was presented in previous sections. In this section we present

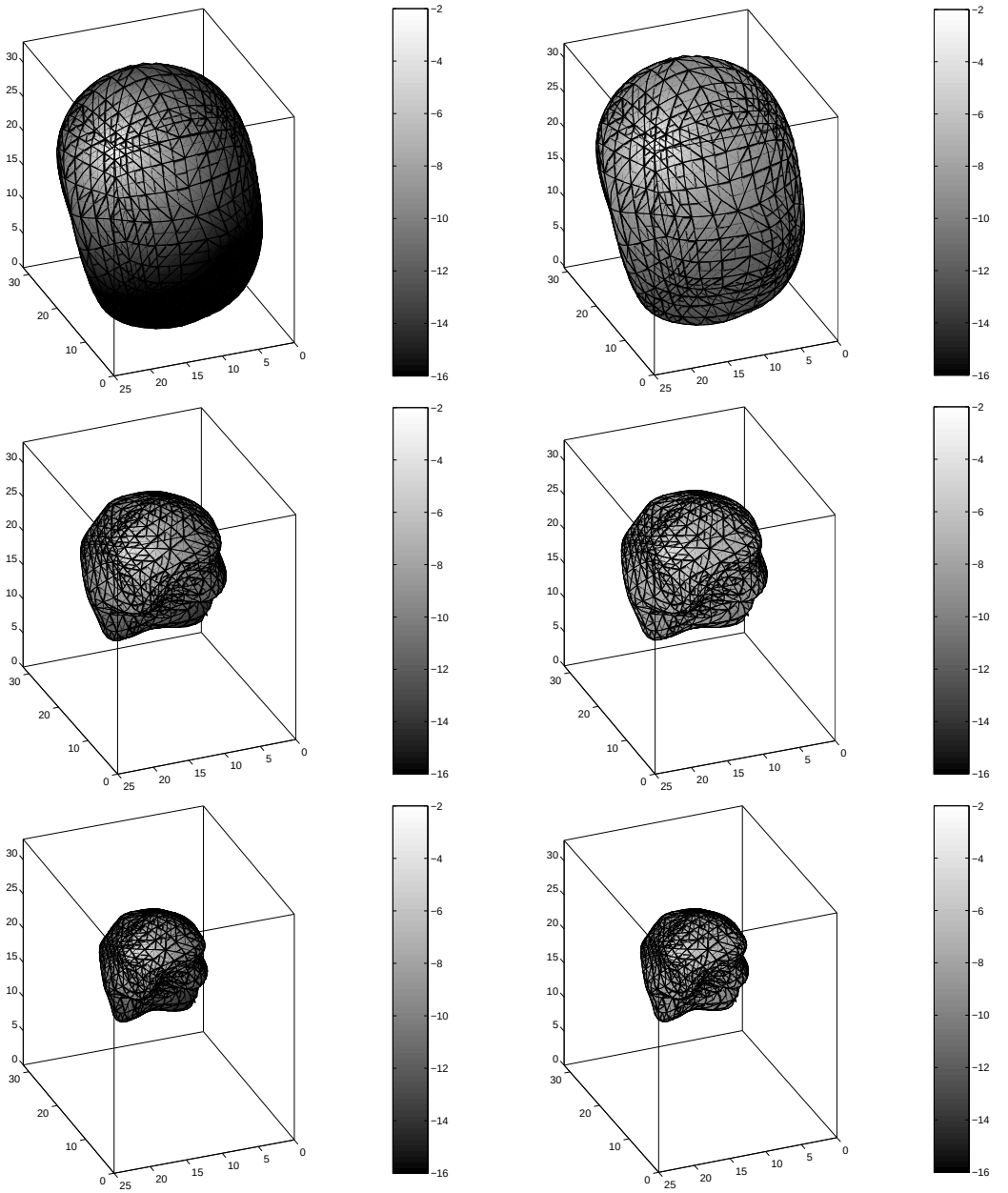


Figure 4.49: Baby's head model without a void gap (left) and with a 3mm void gap between the second and the third surface (right)

the solution for diffuse light propagation in a four-layer concentric sphere model using overlapping and non-overlapping Domain Decomposition Methods (DDM) coupled with some popular methods of solving partial differential

equations like FEM or BEM. The latter is preferred because it yields not only the value of the state function but its normal derivative as well. This fact is beneficial to DDM where these two quantities are strongly exploited.

4.16.1 Introduction

Diffuse Optical Tomography in medicine aims to recover the optical properties of biological tissue from measurement of the transmitted light made at multiple points on the surface of the body. This boundary data measurements can be used to recover the spatial distribution of internal absorbtion and scattering coefficients. It is a non-invasive modality and can generate images of parameters related to blood volume and oxygenation.

The main topic within this field is the development of an efficient and accurate method for calculating the intensity of light transmitted or reflected from the object under experimental investigation. A general model of light propagation can be described using the Radiative Transfer Equation, but a simpler model that can be derived from this equation in the case of sufficiently high scattering is the diffusion equation with Robin boundary conditions [52]. Existing methods to solve this problem are either deterministic, based on the solutions to governing equations, or stochastic based on simulations of the individual scattering and absorbtion events undertaken by each photon. The former include analytical expression based on Green functions [47], and numerical methods based on Finite Difference Method (FDM) or Finite Element Methods (FEM).

In this paper it is assumed that the object being studied is considered as a set of disjoint simply connected regions with constant optical coefficients within each region, but that may differ between regions. In this case the diffusion equation can be replaced by a set of Helmholtz equations for each domain, together with interface conditions. For this problem, analytical solution aren't easily available. Although volume based PDE solvers such as FDM or FEM can certainly be applied to this problem, there are often practical difficulties in constructing meshes for general geometries that respect the interfaces accurately. In contrast, the use of boundary integral methods (e.g. BEM) involve only representation of the surface meshes and can be much easier to implement.

The problem of Optical Tomography in a highly diffusive body Ω with boundary Γ can be modelled by the use of the diffusion equation in the frequency domain form:

$$-\nabla \cdot \kappa(\boldsymbol{r})\nabla\Phi(\boldsymbol{r};\omega) + \mu_a(\boldsymbol{r})\Phi(\boldsymbol{r};\omega) + \frac{(i\omega)}{c}\Phi(\boldsymbol{r},\omega) = q(\boldsymbol{r};\omega)\,, \qquad (4.128)$$

with Robin boundary conditions:

$$\Phi(\mathbf{m};\omega) + 2\alpha\kappa(\mathbf{m})\frac{\partial\Phi(\mathbf{m};\omega)}{\partial n} = h^{-}(\mathbf{m};\omega), \qquad \mathbf{m} \in \Gamma\,, \qquad (4.129)$$

where $\omega \in \mathbf{R}^{+}$ is the frequency modulation, Φ is the photon density c is the velocity of light, q is an internal source of light in medium, h^{-} is an incoming flux, α is a boundary term which incorporates the refractive index mismatch at the tissue-air boundary, $\mathbf{n}$ is the outward normal at the boundary Γ, κ and μ_a are the diffusion and absorption coefficients, respectively. We define, $\kappa = \frac{1}{3(\mu_a+\mu_s')}$, where μ_s' is the reduced scattering coefficient [24, 47]. We use the notation $\mathbf{r}$ for a position vector in Ω and $\mathbf{m}$ for a position vector restricted to the surface Γ.

4.16.2 The four-layer head model

In our research we have taken into consideration a four layer concentric spherical model. The surfaces modelled were the outer skin, the skull and the brain. The Generalised Minimum Residuals Method (GMRES) was used to solve the linear matrix equation $\mathbf{K}\,\mathbf{f} = \mathbf{b}$ obtained from BEM. Here, $\mathbf{f}$ is the discrete version of the unknown functions f, $\mathbf{K}$ is the system matrix, and $\mathbf{b}$ the vector of known coefficients calculated form the light sources in the problem. Additionally $\mathbf{K}$ is a block-bounded asymmetric matrix.

In order to solve equation of 20000 unknowns it takes up to 50 hours to a 64-bit Athlon processor. Taking advantage of DDM [24] as well as BEM we are able to decrease computation time to minutes. Such improvement is partly ensured by BEM which provides in each node of the mesh not only the value of the state function but its normal derivative as well.

Such decomposition is particularly efficient for multilayered geometries in biomedical applications. In the Fig. 4.50 we can see the solution of the amplitude of the state function (photon density) on the most external layer of the 3D BEM model of the neonatal head. In the Fig. 4.50 the same value is depicted but in concentric spheres model.

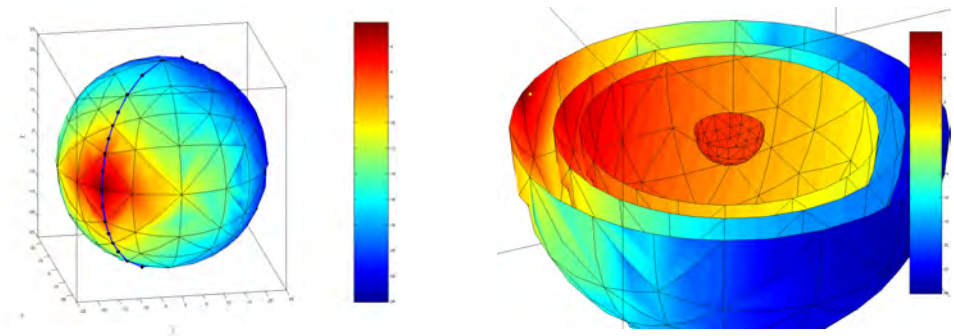


Figure 4.50: Points on the circumference of the sphere from which the values of the state function are taken for presentation. Cross-section of the model under consideration with position of the isotropic light source. Color map in both pictures shows the logarithm of the amplitude of photon density on the boundaries between regions

Generally, there are two kinds of approaches depending on whether the subdomains overlap (Schwarz methods) or are separated (Schur Complement methods [24]). The latter are called substructuring methods and are based on non-overlapping decompositions of the region into a set of subdomains. The number of equations needed to solve this smaller problem is minor, compared to the whole system. Thus the amount of memory required for allocating the equations is smaller too. In the Fig. 4.51 a scheme of Dirichlet-Neumann substructuring algorithm is presented. Arrows indicate the direction of transmission of boundary values over particular subdomains.

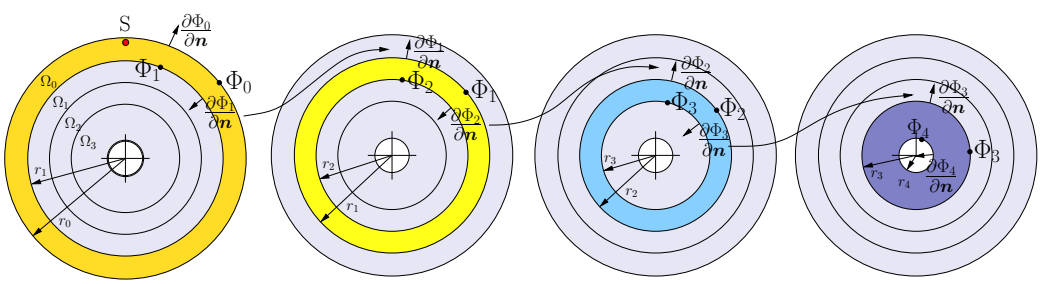


Figure 4.51: Transmission of the boundary conditions in D-N algorithm

In case of overlapping subdomains the decomposition algorithm reduces the number of sequential steps. It is an advantage form standpoint of ill-posed problems with low convergence rate problem, which may converge very slow. The main drwaback of this approach is consuming greater memory resources

as one iteration refers to two regions simultaneously. In the Fig. 4.52 a scheme of two-level overlapping algorithm is presented.

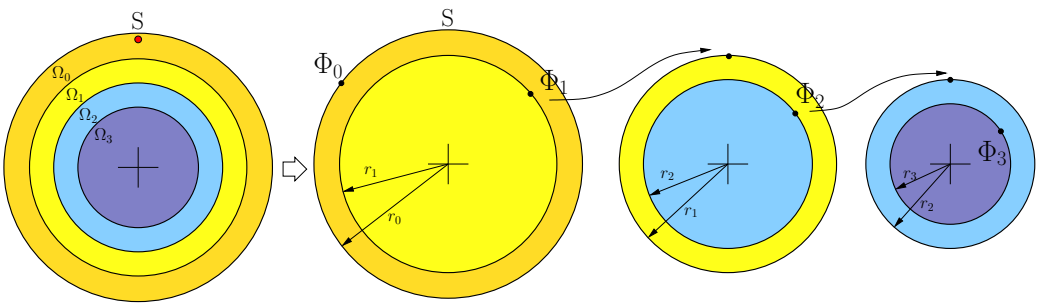


Figure 4.52: Transmission of the boundary conditions in an overlapping D-D algorithm

The four layer spherical model In our research we have taken into consideration a four layer concentric spherical model. The surfaces modelled were the outer skin, the skull and the brain. The Generalised Minimum Residuals Method (GMRES) was used to solve the linear matrix equation $\mathbf{Kf} = \mathbf{b}$ obtained from BEM.

In the Fig. 4.54 we can see the solution of the amplitude of the state function (photon density) on the most external layer of the 3D BEM concentric spherical model and its cross-section.

4.16.3 Conclusion

We have studied substructuring methods with the Dirichlet – Neumann, Neumann – Neumann and Dirichlet – Dirichlet algorithms as well as two - level overlapping method. The most promising one from point of view of this work is overlapping method with transmission Dirichlet boundary condition. However it is very sufficient when compared with any of substructuring methods. It converges very fast and the solution appears after merely five iterations (see Fig. 4.53). The similar error level is obtained after carrying out thirty iterations of nonoverlapping Dirihlet-Neumann algorithm as it was shown in Fig. 4.54.

Domain Decompositon Methods are very powrfull tool, which can be applied to almost every boundary problem. This work involves BEM as a method

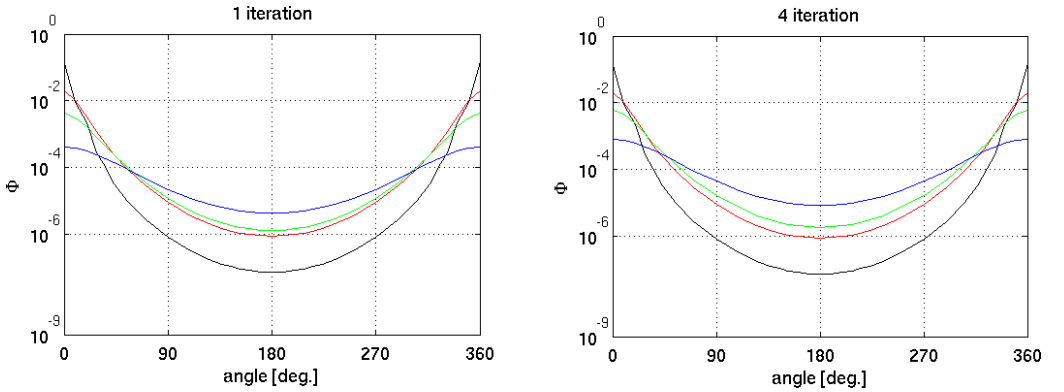


Figure 4.53: Results obtained from the Dirichlet – Dirichlet algorithm

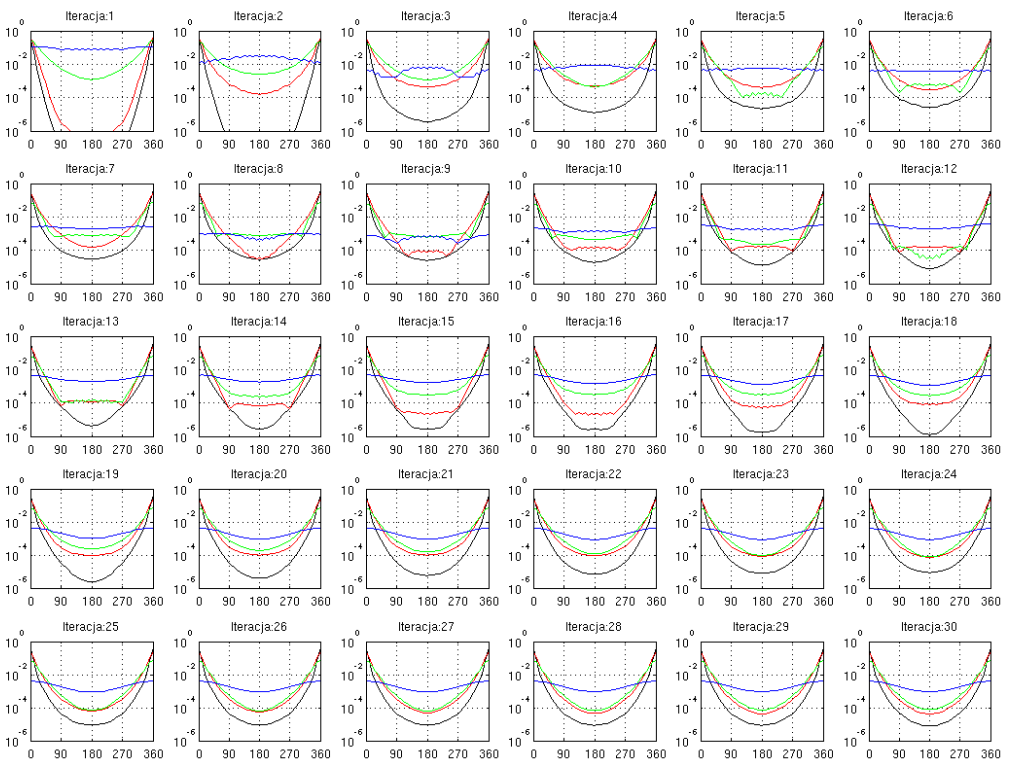


Figure 4.54: Iterations of the Dirichlet – Neumann algorithm applied to four layer spherical model; single images show the logarithm of the amplitude of photon density on boundaries between regions

of solving PDE's governing the problems discussed here, but DDM can interact with any other method e.g. FEM or FDM. So far, research have been conducted using spherical models in order to test and validate algorithms. Obtained results allow applying DDM in much more advanced geometries and in models composed of higher number of boundary elements than it was presented in this work. As usual, the only concern is to match the appriopriate method up with the problem being tackled. According to the results of simulations, ill-posed problems (e.g. discussed here DOT in multilayered environment) require very stable methods because of the insufficient number of boundary and initial conditions. It is clearly visible (see Fig. 4.54) that solution undergoes many fluctuation due to the transmission of weak conditions between decomposed regions. The situation takes turns at using another method (e.g. Dirichlet – Dirichlet) what is depicted in Fig. 4.53 – solution becomes acceptable merely after four iterations.

Bibliography

[1] M. H. Aliabadi and W.S. Hall. The regularizing transformation integration method for boundary element kernels. Comparison with series expansion and weighted Gaussian integration methods. *Engineering Analysis with Boundary Elements*, 6(2):66–70, 1989.

[2] S. R. Arridge. Optical tomography in medical imaging. *Inverse Problems*, 15(2):R41–R93, 1999.

[3] S. R. Arridge, M. Cope, and D. T. Delpy. Theoretical basis for the determination of optical pathlengths in tissue: Temporal and frequency analysis. *Physics in Medicine and Biology*, 37:1531–1560, 1992.

[4] S. R. Arridge, H. Dehghani, M. Schweiger, and E. Okada. The finite element model for the propagation of light in scaterring media: A direct method for domains with nonscattering regions. *Medical Physics*, 27(1):252–264, 2000.

[5] S. R. Arridge and J. C. Hebden. Optical imaging in medicine: II. Modelling and reconstruction. *Physics in Medicine and Biology*, 42:841–853, 1997.

[6] S. R. Arridge, M. Schweiger, M. Hiraoka, and D. T. Delpy. A finite element approach for modeling photon transport in tissue. *Medical Physics*, 20(2):299–309, 1993.

[7] G. Bal. Particle transport through scattering regions with clear layers and inclusions. *J. Comp. Phys.*, 180:659–685, 2002. http://www.columbia.edu/gb2030/pubs.html.

[8] G. Bal. Transport through diffusive and non-diffusive regions, embeded objects, and clear layers. *SIAM J. Appl. Math*, 62:1677–1697, 2002. http://www.columbia.edu/gb2030/pubs.html.

[9] G. Bal and K. Ren. Generalized diffusion model in optical tomography with clear layers. http://www.columbia.edu/gb2030/PAPERS/GDMLocal.ps.

[10] A.A. Becker. *The Boundary Element Method in Engineering. A complete course*. McGraw-Hill Book Company, 1992.

[11] G. Beer. *Programming the Boundary Element Method. An Introduction for Engineers*. John Wiley & Sons, 2001.

[12] D. A. Boas, D. H. Brooks, E. L. Miller, C. A. DiMarzio, M. Kilmer, R. J. Gaudette, and Q. Zhang. Imaging the body with diffuse optical tomography. *IEEE Sig. Proc. Magazine*, 18(6):57–75, 2001.

[13] M. Bonnet. *Boundary Integral Equation Methods for Solid and Fluids*. John Wiley & Sons, 1995.

[14] A. Buchau, C.J. Huber, W. Rieger, and W.M. Rucker. Fast BEM computations with the Adaptive Multilevel Fast Multipole Method. *IEEE Transactions on Magnetics*, 36(4):680–684, 2000.

[15] J.C. de Munck. A linear discretization of the volume conductor boundary integral equation using analitically integrated elements. *IEEE Transactions on Biomedical Engineering*, 39:986–990, 1992.

[16] J.C. de Munck, T.J.C. Faes, and R.M. Heethaar. The boundary element method in the forward and inverse problem of electrical impedance tomography. *IEEE Transactions on Biomedical Engineering*, 47(6):792–800, 2000.

[17] H. Dehghani, B. Brooksby, K. Viswanath, B.W. Pogue, and K. D. Paulsen. The effects of internal refractive index variation in near-infrared optical tomography: a finite element modelling approach. *Physics in Medicine and Biology*, 48:2713–2727, 2003.

[18] F.M.E. Duddeck. *Fourier BEM*. Springer-Verlag, 2002. Lecture Notes in Applied Mechanics, Volume 5.

[19] J.H.M. Frijns, S. L. de Snoo, and R. Schoonhoven. Improving the accuracy of the boundary element method by the use of second-order interpolation functions. *IEEE Transactions on Biomedical Engineering*, 47(10):1336–1346, 2000.

[20] W.S. Hall. *The Boundary Element Method*. Kluwer Academic Publishers, 1994.

[21] D. Hawysz and E. M. Sevick-Muraca. Developments towards diagnostic breast cancer imaging using near-infrared optical measurements and flourescent contrast agents. *Neoplasia*, 2(5):388–417, 2000.

[22] K. Hayami. *A Projection Transformation Method for Nearly Singular Surface Boundary Element Integrals*. Springer-Verlag, 1992.

[23] J. C. Hebden, S. R. Arridge, and D. T. Delpy. Optical imaging in medicine: I. Experimental techniques. *Physics in Medicine and Biology*, 42:825–840, 1997.

[24] J. Heino, S. R. Arridge, J. Sikora, and E. Somersalo. Anisotropic effects in highly scattering media. *Physical Review*, E:031908–1–8, 2003.

[25] N. Hyvönen. Analysis of Optical Tomography with Non-Scattering Regions. Msc Thesis, Helsinki University of Technology, Institute of Mathematics, 2000.

[26] M. Itagaki and C.A. Brebia. Boundary element method applied to neutron diffusion problems. In *Proc. 10th Int. Conf. BEM, Southampton Univ.* Springer-Verlag, 1988.

[27] S.A. Jankins and J.R. Bowler. Numerical evaluation of singular matrix elements in three dimensions. *IEEE Transactions on Magnetics*, 27:4438–4444, November 1991.

[28] J.H. Kane, A. Gupta, and S. Saigal. Reusable intrinsic sample point (RISP) algorithm for the efficient numerical integration of three dimensional curved boundary elements. *Int. Journ. for Numerical Meth. in Engineering*, 28:1661–1676, 1989.

[29] Tao Lu and D. O. Yevick. Boundary element analysis of dielectric waveguides. *Journal of the Optical Society of America*, 19(6):1197–1206, June 2002.

[30] J.W.H. Meijs, O.W. Weiner, M.J. Peters, and A. van Oosterom. On the numerical accuracy of the boundary element method. *IEEE Transactions on Biomedical Engineering*, 36:1038–1049, 1989.

[31] R. Model, M. Orlt, M. Walzel, and R. Hünlich. Reconstruction algorithm for near-infrared imaging in turbid media by means of time-domain data. *Journal of the Optical Society of America A: Optics Image Science and Vision*, 14(1):313–324, 1997.

[32] G. Muller, B. Chance, R. Alfano, S. Arridge, J. Beuthan, E. Gratton, M. Kaschke, B. Masters, S. Svanberg, and P. van der Zee, editors. *The forward and inverse problems in time-resolved infrared imaging*, 1993. Proc. SPIE, Medical Optical Tomography: Functional Imaging and Monitoring.

[33] Seong Jin Park and Tai Hun Kwon. Sensitivity analysis formulation for 3D conduction heat transfer with complex geometries using a boundary element method. *Int. Journ. for Numerical Meth. in Engineering*, 39:2837–2862, 1996.

[34] K. D. Paulsen and H. Jiang. Spatially-varying optical property reconstruction using a finite element diffusion equation approximation. *Medical Physics*, 22(6):691–701, 1995.

[35] H.L.G. Pina, J.L.M. Fernandes, and C.A. Brebia. Some numerical integration formulae over triangles and squares with a 1/R singularity. *Appl. Math. Modelling*, 5:209–211, 1981.

[36] J.D. Riley. Modelling of light propagation in mixed diffusing and non–diffusing domains. PhD Thesis, University College London, Department of Computer Science, 2003.

[37] J.D. Riley, S. R. Arridge, Y. Chrysanthou, H. Dehghani, E.M.C. Hillman, and M. Schweiger. Radiosity diffusion model in 3D. In Stefan Andersson-Engels and Michael F. Kaschke, editors, *Photon Migration, Optical Coherence Tomography, and Microscopy*, pages 153–164, 18-21 June 2001. Proceedings of SPIE.

[38] J.D. Riley, H. Dehghani, M. Schweiger, S. R. Arridge, J. Ripoll, and M. Nieto-Vesperinas. 3D optical tomography in the presence of void regions. OPTICS EXPRESS, 7(13), 2000.

[39] J. Ripoll. Light diffusion in turbid media with biomedical application. PhD Thesis, University of Madrid, 2000.

[40] J. Ripoll, M. Nieto-Vesperinas, and S. R. Arridge. Effect of roughness in nondiffusive regions within diffusive media. *Journal of the Optical Society of America A: Optics Image Science and Vision*, 18(4):940–947, 2001.

[41] J. Ripoll, M. Nieto-Vesperinas, S. R. Arridge, and H. Dehghani. Boundary conditions for light propagation in diffusive media with nonscattering regions. *Journal of the Optical Society of America A: Optics Image Science and Vision*, 17(9):1671–1681, 2000.

[42] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7(3):856–869, April 1986.

[43] M. Schweiger and S. R. Arridge. The finite element model for the propagation of light in scattering media: Frequency domain case. *Medical Physics*, 24(6):895–902, 1997.

[44] M. Schweiger, S. R. Arridge, M. Hiraoka, and D. T. Delpy. The finite element model for the propagation of light in scattering media: Boundary and source conditions. *Medical Physics*, 22(11):1779–1792, 1995.

[45] J. Sikora. *Boundary Element Method for Impedance and Optical Tomography.* Oficyna Wydawnicza Politechniki Warszawskiej, 2007.

[46] J. Sikora, J. Riley, S. R. Arridge, A. D. Zacharopoulos, and J. Ripoll. Analysis of light propagation in diffusive media with non–scattering regions using 3D BEM. In *Proceedings of XIIth International Symposium on Theoretical Electrical Engineering ISTET'03*, pages 511–514, 2003. Warsaw, Poland, July 6-9.

[47] J. Sikora, A. D. Zacharopoulos, A. Douiri, M. Schweiger, L. Horesh, S. R. Arridge, and J. Ripoll. Diffuse Photon Propagation in Multilaiered Geometries. *Physics in Medicine and Biology*, 51(3):497–516, 2006.

[48] Loic Vanel, P.A. Lemieux, and D.J. Durian. Diffusing wave spectroscopy for arbitrary geometries: Numerical analysis by boundary element method. *Applied Optics*, 40(24):1336–1346, 2001.

[49] L. C. Wrobel. *The Boundary Element Method; Volume 1; Applications in Thermo–Fluids and Acoustics.* John Wiley & Sons, LTD, 2002.

[50] A. Yodh and B. Chance. Spectroscopy and imaging with diffusing light. *Phys. Today*, pages 38–40, 1995.

[51] Point in triangle test. http:// www.blackpawn.com/ texts/ pointinpoly/ default.html.

[52] A. Zacharopoulos, S.R. Arridge, O. Dorn, V. Kolehmainen, and J. Sikora. 3D Shape Reconstruction in Optical Tomography Using Spherical Harmonics and BEM. *Journal of Electromagnetic Waves and Applications*, (13):1827–1836, 2006.

[53] A. Zacharopoulos, S.R. Arridge, O. Dorn, V. Kolehmainen, and J. Sikora. Three-dimensional reconstruction of shape and picewise constant region values for optical tomography using spherical harmonic parametrization and a boundary element method BEM. *Inverse Problems*, pages 1–24, 2006.

[54] A.D. Zacharopoulos and S. R. Arridge. Surface mesh creation for 3D objects with sphere topology by homeomorphism on a unit sphere. In *Proceedings of IVth International Workshop: Computational Problems of Electrical Engineering*, pages 211–215, 2002. Zakopane, Poland, September 2-5.

[55] O.C. Zienkiewicz, D.W. Kelly, and P. Bettess. The coupling of the finite elements method and boundary solution procedures. *Int. Journ. for Numerical Meth. in Engineering*, pages 355–375, 1977.

[56] O.C. Zienkiewicz and R.L. Taylor. *The Finite Element Method*. McGraw-Hill, 1993. 4th edn., New York.

Chapter 5

Infinite Boundary Elements

J. Sikora, W. Wójcik, M. Pańczyk, A. Kotyra

5.1 Introduction

There are numerous examples where the region of interests extents to infinity like in geotechnical or wave problems. Sometimes surrounding medium is important and unbounded e.g. electromagnetic field around an electrical machine. In other cases distant boundary conditions may not be clearly defined as for example in the area of medicine when we can not place detectors on some surfaces. Technical term of infinite is determined by the severity of the physical phenomena decay. It can be closer to mathematical infinite continuum concept in geotechnics but can also be measured in millimeters like in light propagation through some highly vascularized body tissue in Optical Tomography.

Normally the analyzed area is extended outside the region of interest to create a distant boundary where conditions and its exact form should not have a great impact on the results. Wrong boundary conditions or improper placement of such artificial boundary can introduce an unknown error if the truncation occurs too near. On the other hand excessive mesh increases

number of boundary elements and decreases the computational efficiency especially annoying while calculating inverse problem solution.

A more effective method is to incorporate infinite elements into the conventional BEM analysis. Implementation of such elements can reduce the mesh and avoid the problem of setting incorrect boundary conditions by creating an open boundary model.

5.2 Infinite elements classification

There are two main lines of infinite elements development [10, 5, 21, 4, 15]:

1. **mapped infinite elements** where the element is transformed from finite to infinite domain,

2. **decay functions infinite elements** which uses special decay functions in conjunction with ordinary boundary element interpolation functions.

Both types offers similar accuracy. Of course final mesh will consists of ordinary and infinite boundary elements.

In case of two dimensional infinite boundary elements, used to describe three-dimensional objects, discussed infinite elements will be narrowed to these based on 8 node second order quadrilateral isoparametric standard element. It is reasonable as the transformation of the element to infinity requires higher order interpolation functions and quadrilateral elements are more convenient to describe the transformation in one or more directions.

5.3 Mapped infinite boundary elements

5.3.1 One-dimensional infinite boundary elements

One-dimensional infinite boundary elements are used for two-dimensional objects. For *mapped infinite elements* special basis interpolation functions M are introduced [22, 19, 16] (Fig. 5.1). The infinite basis interpolation functions M_i^m are applied to the geometry and are used to generate the Jacobian matrix, its inverse, and its determinant. The standard basis interpolation functions N_i^{std} [21, 5, 4] are applied to the field variables. Basis interpolation

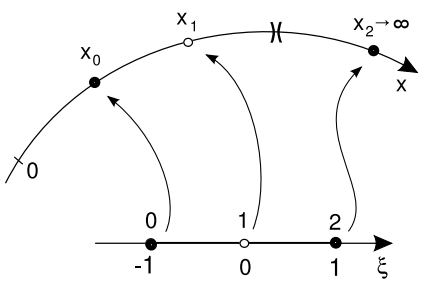


Figure 5.1: Mapped infinite boundary element domain transformation

functions M_i^1 and M_i^2 should sum to unity:

$$M_0(\xi) + M_1(\xi) = 1. \tag{5.1}$$

For one-dimensional element like presented in Fig. 5.1, it's geometry is interpolated as:

$$x = M_0 x_0 + M_1 x_1, \quad \text{where} \quad M_0 = -\frac{2\xi}{1-\xi}, \quad M_1 = \frac{1+\xi}{1-\xi}. \tag{5.2}$$

This yields $x = x_0$ at $\xi = -1$, $x = x_1$ at $\xi = 0$ and

$$x_2 = \lim_{\xi \to 1} \frac{-2\xi x_0 + (1+\xi)x_1}{1-\xi} = \infty .$$

The infinite basis functions M_0 and M_1 distribution over the boundary element are presented in Fig. 5.2. Boundary Integral Equation in discretized

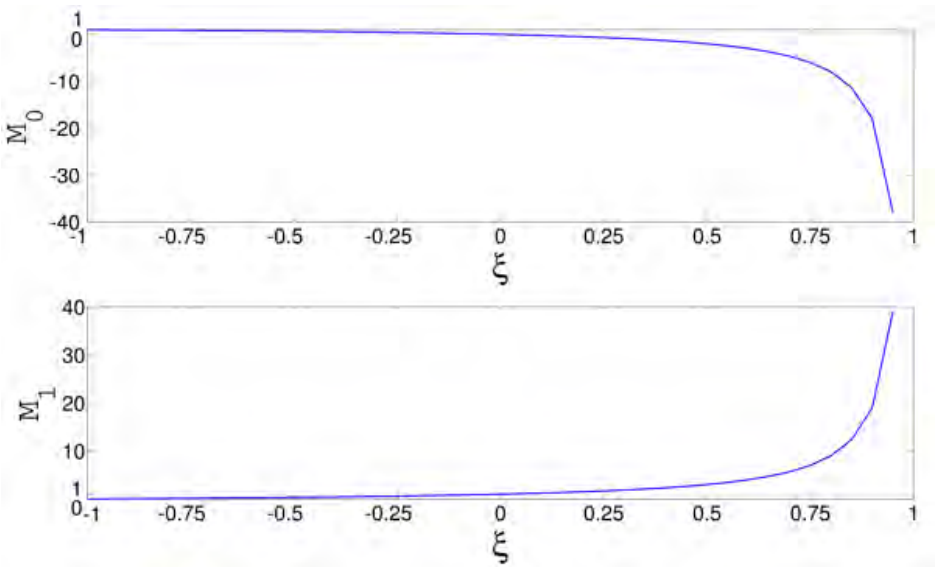


Figure 5.2: Infinite basis interpolation functions distribution

form including both finite and infinite regions can be written in as follows:

$$
\begin{aligned}
c(\mathbf{r})\Phi_i(\mathbf{r}) \quad &+ \quad \sum_{i=0}^{n-1}\sum_{k=0}^{2}\int_{-1}^{+1}\Phi(\mathbf{r}')N_k(\xi)\frac{\partial G(|\mathbf{r}-\mathbf{r}'|)}{\partial n}J^N(\xi)d\xi \quad + \\
&+ \quad \sum_{j=0}^{m-1}\sum_{l=0}^{1}\int_{-1}^{\infty}\Phi(\mathbf{r}')M_l(\xi)\frac{\partial G(|\mathbf{r}-\mathbf{r}'|)}{\partial n}J^M(\xi)d\xi \quad = \qquad (5.3) \\
&= \quad \sum_{i=0}^{n-1}\sum_{k=0}^{2}\int_{-1}^{+1}\frac{\partial\Phi(\mathbf{r}')}{\partial n}G(|\mathbf{r}-\mathbf{r}'|)N_k(\xi)J^N(\xi)d\xi \quad + \\
&+ \quad \sum_{j=0}^{m-1}\sum_{l=0}^{1}\int_{-1}^{\infty}\frac{\partial\Phi(\mathbf{r}')}{\partial n}G(|\mathbf{r}-\mathbf{r}'|)M_l(\xi)J^M(\xi)d\xi.
\end{aligned}
$$

It is to notice that in the above equation (5.3) the third node which tends to infinity will not take part in the calculations.

5.3.2 Two-dimensional infinite boundary elements

Two-dimensional infinite boundary elements are used for three-dimensional objects analysis. In that case so called serendipity basis interpolation functions M_i^m are used for geometry transformation [10, 22, 19, 16]. It mapps

eight node quadrilateral isoparametric boundary element into corresponding five node infinite boundary element – Fig. 5.3.

There are no exact analogy for mapping functions like in standard interpolation functions i.e. all the shape functions should sum to unity and all the derivatives to zero or in other words to having unit value of the field variable at all nodes. Before summing up, mapping functions have to be multiplied by a constant a, where a is a distance from Zienkiewicz type 'pole' radial point to the first node of such one-dimensional element like it is presented by Bettess ([10] in chapter 4). That is for one infinite positive ξ direction M_0 , M_6 and M_7 have to be multiplied by 1 and M_1 and M_5 by 2. By choosing eight node quadrilateral boundary elements, serendipity mapping functions are used. Despite their names procedure for deriving these basis interpolation functions is quite logical [23, 10]. Eight node quadrilateral boundary element and it's transformation into five node mapped infinite element is presented in Fig. 5.3. It is to notice that infinite boundary element based on 8-node quadrilateral second order boundary element will consists only from 5 nodes numbered $0, 1, 5, 6$ and 7 like it is presented in Fig. 5.3. Relevant infinite basis interpolation functions for remaining nodes are as follows:

$$
\begin{aligned}
M_0 &= \frac{-1-\xi+\xi\eta+\eta^2}{1-\xi}, & M_1 &= \frac{1+\xi}{1-\xi}\cdot\frac{1}{2}\left(1-\eta\right), \\
M_5 &= \frac{1+\xi}{1-\xi}\cdot\frac{1}{2}\left(1+\eta\right), & M_6 &= \frac{-1-\xi-\xi\eta+\eta^2}{1-\xi}, \\
& & M_7 &= \frac{2}{1-\xi}\cdot\left(1-\eta^2\right).
\end{aligned}
\tag{5.4}
$$

Serendipity infinite mapping functions distribution is presented in Fig. 5.5. The first derivatives of the mapping functions 5.4 with respect to ξ and η are given by:

$$
\begin{aligned}
\frac{\partial M_7}{\partial \xi} &= \frac{2}{(1-\xi)^2}\left(1-\eta^2\right), & \frac{\partial M_7}{\partial \eta} &= \frac{-4\eta}{1-\xi}, \\
\frac{\partial M_1}{\partial \xi} &= \frac{1-\eta}{(1-\xi)^2}, & \frac{\partial M_1}{\partial \eta} &= -\frac{1}{2}\frac{1+\xi}{1-\xi}, \\
\frac{\partial M_5}{\partial \xi} &= \frac{1+\eta}{(1-\xi)^2}, & \frac{\partial M_5}{\partial \eta} &= \frac{1}{2}\frac{1+\xi}{1-\xi}, \\
\frac{\partial M_0}{\partial \xi} &= \frac{-2+\eta+\eta^2}{(1-\xi)^2}, & \frac{\partial M_0}{\partial \eta} &= \frac{\xi-2\eta}{1-\xi}, \\
\frac{\partial M_6}{\partial \xi} &= \frac{-2-\eta-\eta^2}{(1-\xi)^2}, & \frac{\partial M_6}{\partial \eta} &= \frac{2\eta-\xi}{1-\xi}.
\end{aligned}
\tag{5.5}
$$

Sometimes it may be necessary to have a two dimensional element which extends to infinity in two directions. Serendipity mapping functions can be developed for both ξ and η directions.

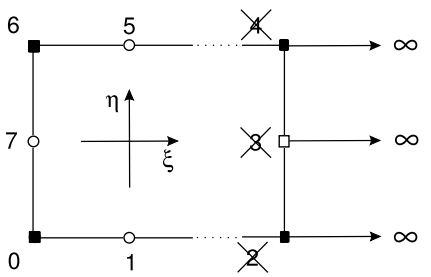


Figure 5.3: Serendipity type mapped infinite boundary element based on standard 8 node quadrilateral boundary element

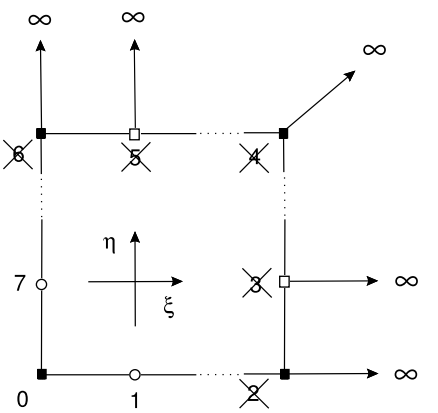


Figure 5.4: Serendipity type mapped infinite boundary corner-element based on standard 8 node quadrilateral boundary element

Transformation of 8 node quadrilateral boundary element into relevant 3 node mapped infinite element which extends into infinity in both (positive) ξ and η directions is presented in Fig. 5.4. Nodes $2-6$ do not figure in the calculations, and relevant infinite basis interpolation functions for remaining

nodes $0, 1, 7$ are as follows:

$$\begin{aligned} M_0 &= \frac{-4\,(1+\xi+\eta)}{(1-\eta)\,(1-\xi)}, \\ M_1 &= \frac{1+\xi}{1-\xi}\,\frac{2}{1-\eta} = \frac{2\,(1+\xi)}{(1-\xi)\,(1-\eta)}, \\ M_7 &= \frac{2}{1-\xi}\,\frac{1+\eta}{1-\eta} = \frac{2\,(1+\eta)}{(1-\xi)\,(1-\eta)}. \end{aligned} \tag{5.6}$$

Infinite basis interpolation functions for 5 node element are presented in Fig. 5.5 and for 3 node element in Fig. 5.6. For debugging purposes, in case of ordinary basis interpolation functions, it is to check if all basis interpolation functions sum to unity and all the derivatives to zero. The simple test is to check if each function has unit value on their own node and zero on the others. For mapping functions as it is presented in figures 5.5 and 5.6 nodes remaining in the calculations fulfils that condition but there is no exact analogy for all nodes. Further tests using Zienkiewicz type of mapped infinite elements [22] are devised by Bettess [10].

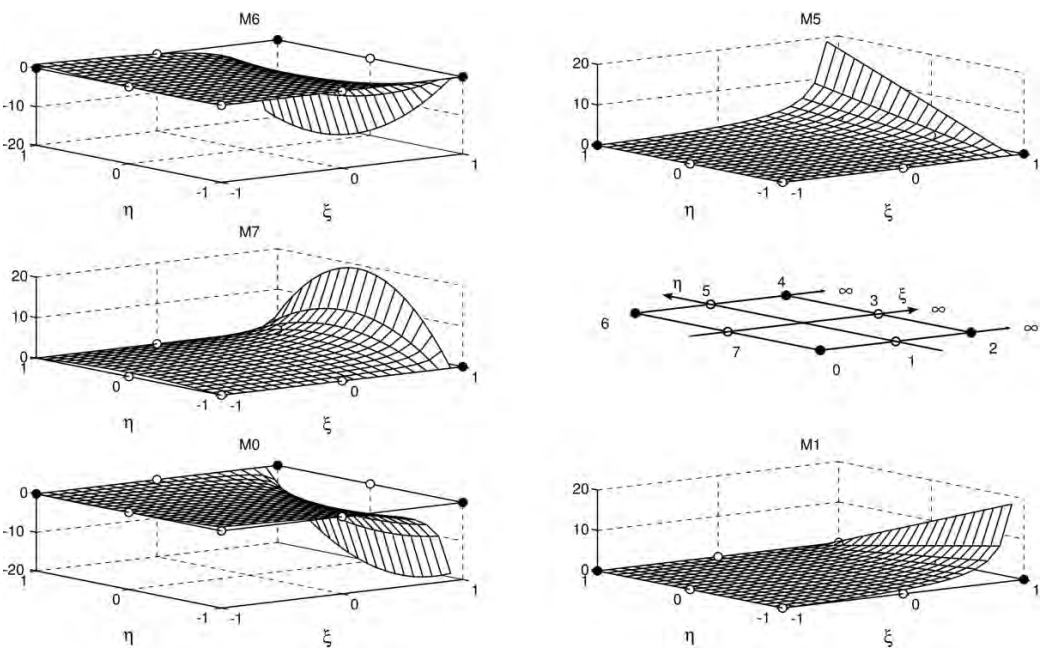


Figure 5.5: Serendipity infinite mapping functions distribution for one infinite direction along positive ξ axis i.e. basis interpolation functions M_0, M_1, M_5, M_6, M_7, (Eq. (5.4)), for 5 node mapped infinite boundary element

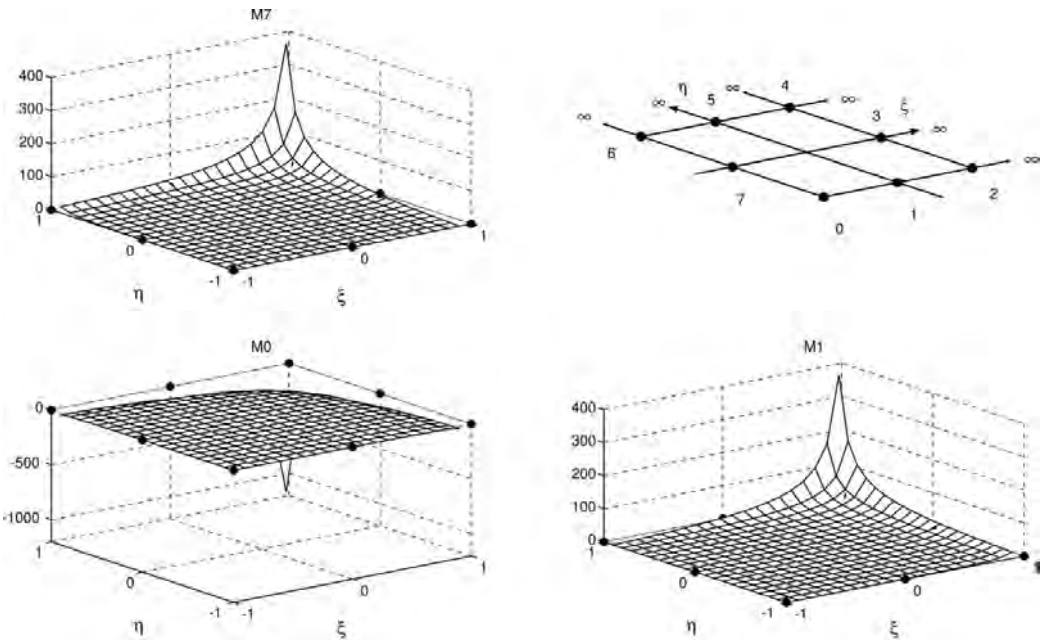


Figure 5.6: Serendipity infinite mapping functions distribution for two infinite direction along positive ξ and η axis i.e. basis interpolation functions M_0, M_1, M_7, (Eq. (5.6)), for 3 node mapped infinite boundary element

To study boundary elements which are two-dimensional structures placed in the 3D space, first we need to define the way in which we can pass from the xyz global Cartesian system to the ξ, η, ζ system defined over the element, where ξ, η are oblique coordinates and ζ is in the direction of the normal. The transformation for a given function Φ is related through the following:

$$\begin{bmatrix} \frac{\partial \Phi}{\partial \xi} \\ \frac{\partial \Phi}{\partial \eta} \\ \frac{\partial \Phi}{\partial \zeta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} \begin{bmatrix} \frac{\partial \Phi}{\partial x} \\ \frac{\partial \Phi}{\partial y} \\ \frac{\partial \Phi}{\partial z} \end{bmatrix}, \tag{5.7}$$

where the square matrix is the Jacobian matrix (or Jacobi matrix).

Transformation of this type allows us to describe differentials of surface in the Cartesian system in terms of the curvilinear coordinates. A differential of area will be given by:

$$d\Gamma = |\mathbf{n}|\, d\xi d\eta = \left| \frac{\partial \mathbf{r}}{\partial \xi} \times \frac{\partial \mathbf{r}}{\partial \eta} \right| d\xi d\eta = \sqrt{{n_x}^2 + {n_y}^2 + {n_z}^2}\, d\xi d\eta, \tag{5.8}$$

where:

$$n_x = \frac{\partial y}{\partial \xi}\frac{\partial z}{\partial \eta} - \frac{\partial y}{\partial \eta}\frac{\partial z}{\partial \xi}, \quad n_y = \frac{\partial z}{\partial \xi}\frac{\partial x}{\partial \eta} - \frac{\partial z}{\partial \eta}\frac{\partial x}{\partial \xi}, \quad n_z = \frac{\partial x}{\partial \xi}\frac{\partial y}{\partial \eta} - \frac{\partial x}{\partial \eta}\frac{\partial y}{\partial \xi}. \tag{5.9}$$

This mapping introduces the Jacobian[1] J proportional to the magnitude of the area of the mapped boundary element.

The first derivatives of the mapped interpolation functions with respect to the ξ and η for 3 node mapped infinite element are given by:

$$\begin{aligned}
\frac{\partial M_0}{\partial \xi} &= \frac{-4\,(2+\eta)}{(1-\xi)^2\,(1-\eta)}\,, & \frac{\partial M_0}{\partial \eta} &= \frac{-4\,(2+\xi)}{(1-\xi)\,(1-\eta)^2}, \\
\frac{\partial M_1}{\partial \xi} &= \frac{4}{(1-\xi)^2\,(1-\eta)}\,, & \frac{\partial M_1}{\partial \eta} &= \frac{2\,(\xi+1)}{(1-\xi)\,(1-\eta)^2}\,, \\
\frac{\partial M_7}{\partial \xi} &= \frac{2\,(\eta+1)}{(1-\xi)^2\,(1-\eta)}\,, & \frac{\partial M_7}{\partial \eta} &= \frac{4}{(1-\xi)\,(1-\eta)^2}\,.
\end{aligned} \tag{5.10}$$

The boundary integral equation containing both finite (surface covered by ordinary 8 node boundary elements) and infinite boundary elements (surface covered by infinite 5 node mapped boundary elements) after discretization will take the form:

$$\begin{aligned}
& c(\mathbf{r})\Phi_i(\mathbf{r}) + \sum_{i=0}^{n-1}\sum_{k=0}^{7}\int_{-1}^{+1}\int_{-1}^{+1}\Phi(\mathbf{r}')N_k(\xi,\eta)\frac{\partial G(|\mathbf{r}-\mathbf{r}'|)}{\partial n}J^N(\xi,\eta)d\xi d\eta + \\
+ & \sum_{j=0}^{m-1}\sum_{l=0}^{4}\int_{-1}^{\infty}\int_{-1}^{+1}\Phi(\mathbf{r}')M_l(\xi,\eta)\frac{\partial G(|\mathbf{r}-\mathbf{r}'|)}{\partial n}J^M(\xi,\eta)d\xi d\eta \quad = \\
= & \sum_{i=0}^{n-1}\sum_{k=0}^{7}\int_{-1}^{+1}\int_{-1}^{+1}\frac{\partial\Phi(\mathbf{r}')}{\partial n}G(|\mathbf{r}-\mathbf{r}'|)N_k(\xi,\eta)J^N(\xi,\eta)d\xi d\eta \quad + \\
+ & \sum_{j=0}^{m-1}\sum_{l=0}^{4}\int_{-1}^{\infty}\int_{-1}^{+1}\frac{\partial\Phi(\mathbf{r}')}{\partial n}G(|\mathbf{r}-\mathbf{r}'|)M_l(\xi,\eta)J^M(\xi,\eta)d\xi d\eta \quad .
\end{aligned} \tag{5.11}$$

For singularity treatment as one of the most effective, regularization method [20] was used. The advantage of using that method is that the calculation schema remains unchanged for infinite elements, except nodes tending to infinity which does not take part in the calculations.

[1]Jacobian in this context means determinant of the Jacobi matrix

5.4 Decay basis functions

5.4.1 One-dimensional infinite boundary elements

The idea of the decay functions infinite elements is to construct new infinite basis interpolation functions M_i by multiplying standard basis interpolation functions N_i, known from ordinary boundary elements, by a decay functions D_i [22, 7, 8, 10, 5, 4, 15]. Decay functions D_i have to ensure that the element behaviour at infinity corresponds to the physics of the problem. Relations between infinite basis interpolation functions M_i, standard basis interpolation functions N_i and decay functions D_i are as follows:

$$M_i(\xi) = N_i(\xi) D_i(\xi), \tag{5.12}$$

where index $i = 0, 1, \ldots, 8$ represents element node number. Both standard and infinite basis interpolation functions should have unit value of all nodes, so decay functions should also be unity at their own nodes:

$$D_i(\xi_i) = 1. \tag{5.13}$$

Reciprocal decay functions

Reciprocal decay functions in local coordinates takes the following form [22, 7, 10, 15]:

$$D_i(\xi) = \left(\frac{\xi_i - \xi_o}{\xi - \xi_o} \right)^n, \tag{5.14}$$

where ξ_o is some origin point, which rule is to avoid a singularity in the infinite element.

Point ξ_o must be located outside the infinite element in the opposite direction to that which extends to infinity. For the element which extends to infinity in positive ξ direction $\xi_o < -1$. Moreover exponent n have to be greater than the highest power of ξ encountered in M which ensures that $\xi \to \infty$, $M_i \to 0$. In case the element is extended to infinity in negative ξ direction decay function remains unchanged:

$$D_i(\xi) = \left(\frac{\xi_o - \xi_i}{\xi_o - \xi} \right)^n, \tag{5.15}$$

but ξ_o must be > 1.

Exponential decay functions

Exponential decay functions expressed by e^{-r} form decays to zero faster than reciprocal. For the decay only in one positive direction of ξ axis more precise form for the decay function is:

$$D_i(\xi) = e^{(\xi_i - \xi)/L}, \tag{5.16}$$

where L is a length which determines the severity of the decay, and ξ_i ensures holding the condition (5.13).

In case of decay in the negative ξ direction equation (5.16) becomes:

$$D_i(\xi_i) = e^{(\xi - \xi_i)/L}. \tag{5.17}$$

5.5 Numerical integration

5.5.1 Reciprocal decay functions – Gauss-Legendre

For reciprocal decay it is convenient to map Gauss-Legendre integration rule from the range of $< -1, +1 >$ to $< -1, \infty >$ [1, 12, 18, 10]. Following Davies and Rabinowitz [12] :

$$\int_a^b f(x)dx = (b-a)\int_0^\infty f\left(\frac{a+bt}{1+t}\right)\frac{dt}{(1+t^2)}. \tag{5.18}$$

For $a = -1$ and $b = +1$ equation (5.18) becomes:

$$\int_{-1}^{+1} f(x)dx = 2\int_0^\infty f\left(\frac{t-1}{t+1}\right)\frac{dt}{(1+t^2)}, \tag{5.19}$$

where $x = (t-1)/(t+1)$ and $t = 1 + 2x/(1-x)$.

Defining new variable $\xi = t - 1$ and $t = 1 + \xi$ for desired range of numerical integration will be achieved:

$$\int_{-1}^{\infty} f(\xi)d\xi = 2\int_{-1}^{+1} f\left(\frac{2x}{1-x}\right)\frac{dx}{(1-x^2)}. \tag{5.20}$$

It also to set a new variable of integration abscissa and weights:

$$\xi = 2x/(1-x), \tag{5.21}$$

and:

$$W_{new} = W_{old}\, 2/(1-x)^2, \tag{5.22}$$

where x and W_{old} are tabulated Gaussa-Legendre abscissa and weights.

5.5.2 Exponential decay functions – Gauss-Laguerre

For exponential decay functions infinite elements it is to transform the standard Gauss-Laguerre formula (5.23):

$$\int_0^{\infty} f(x)e^{-x}dx, \tag{5.23}$$

to desired for infinite elements integration limits $< -1, \infty >$ and decay functions (5.16). For exponential decay function infinite elements, basis interpolation function and its derivatives typical term is:

$$p(\xi)e^{-\xi/L}. \tag{5.24}$$

The element matrix is the product of these terms and element has been extended to infinite. Hence, the typical formula to be integrated is:

$$\int_{-1}^{\infty} q(\xi)e^{-2\xi/L}d\xi. \tag{5.25}$$

Possible mapping to transfer the (5.25) integration range can be defined as:

$$\begin{aligned} t &= \frac{2}{L}(\xi+1) \quad , \qquad \xi = \frac{L}{2}t - 1, \\ \frac{dt}{d\xi} &= \frac{2}{L} \quad , \qquad \frac{d\xi}{dt} = \frac{L}{2}. \end{aligned} \tag{5.26}$$

The integration formula related to standard Gauss-Laguerre becomes:

$$\int_{-1}^{\infty} q(\xi)e^{-2\xi/L}d\xi = \int_{0}^{\infty} q(\xi)(L/2)e^{2/L}e^{-t}dt, \tag{5.27}$$

where new abscissa and weights corresponding to tabulated standard are:

$$\xi = \frac{L}{2}t - 1,$$

and:

$$W_{new} = W_{old}\frac{L}{2}e^{2/L}. \tag{5.28}$$

For the decay in the negative direction equation corresponding to (5.27) is:

$$\int_{-\infty}^{+1} q(\xi)e^{2\xi/L}d\xi = \int_{0}^{\infty} q(\xi)(L/2)e^{2/L}e^{-t}dt, \tag{5.29}$$

where $\xi = 1 - \frac{L}{2}\,t$.

5.6 Modified boundary integral equation

For infinite boundary elements based on second order isoparametric boundary element, standard basis interpolation function $N_k(\xi)$ [14, 6, 11, 17, 20, 2] are used to transform global to local coordinates:

$$\begin{aligned} N_0(\xi) &= -\frac{\xi}{2}(1-\xi) = 0.5\xi(\xi-1), \\ N_1(\xi) &= (1+\xi)(1-\xi) = 1-\xi^2, \\ N_2(\xi) &= +\frac{\xi}{2}(1+\xi) = 0.5\xi(\xi+1). \end{aligned} \tag{5.30}$$

In conjunction to reciprocal decay functions (5.14) infinite basis interpolation functions are received:

$$\begin{aligned} M_0^p(\xi) &= 0.5\xi(\xi-1)\left(\frac{-1-\xi_o}{\xi-\xi_o}\right)^3, \\ M_1^p(\xi) &= \left(1-\xi^2\right)\left(\frac{-\xi_o}{\xi-\xi_o}\right)^3, \\ M_2^p(\xi) &= 0.5\xi(\xi+1)\left(\frac{1-\xi_o}{\xi-\xi_o}\right)^3. \end{aligned} \tag{5.31}$$

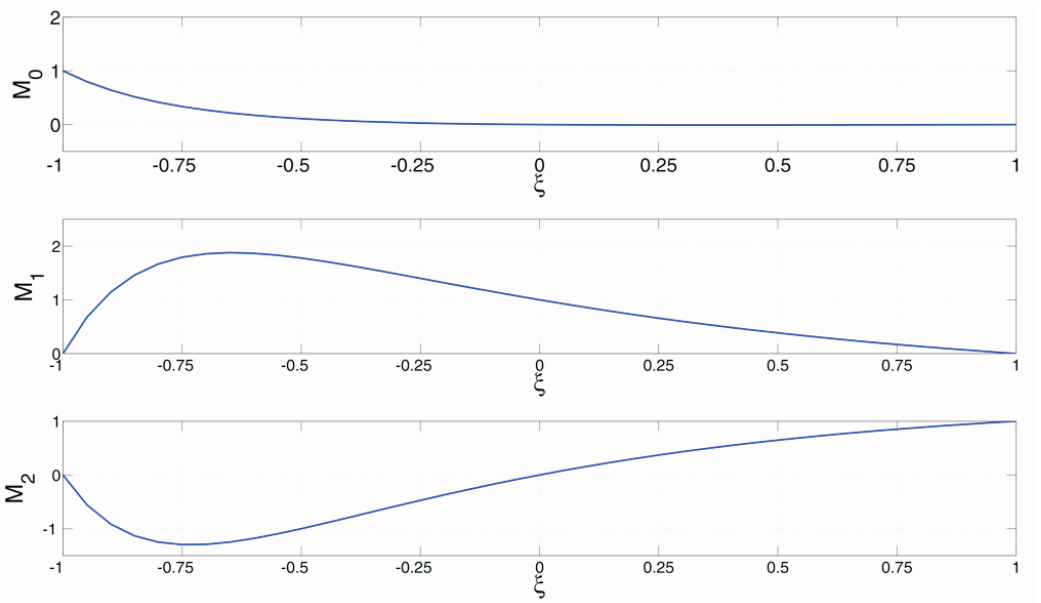


Figure 5.7: Distribution of basis interpolation functions M_0, M_1 i M_2, using reciprocal decay functions

The infinite basis interpolation functions, using reciprocal decay functions (5.31), distribution over the boundary element are presented in Fig. 5.7. The differentials of the basis interpolation functions (5.31) are as follows:

$$\begin{aligned}
\frac{\partial M_0^p(\xi)}{\partial \xi} &= \frac{(\xi_o+1)^3\,(2\xi\xi_o-\xi_o+\xi^2-2\xi)}{2(\xi-\xi_o)^4},\\
\frac{\partial M_1^p(\xi)}{\partial \xi} &= -\frac{{\xi_o}^3\,(2\xi\xi_o+\xi^2-3)}{(\xi-\xi_o)^4},\\
\frac{\partial M_2^p(\xi)}{\partial \xi} &= \frac{(\xi_o-1)^3\,(2\xi\xi_o+\xi_o+\xi^2+2\xi)}{2(\xi-\xi_o)^4}.
\end{aligned} \tag{5.32}$$

In conjunction of exponential decay functions (5.16), exponential infinite basis interpolation functions are:

$$\begin{aligned}
M_0^e(\xi) &= 0.5\xi(\xi-1)e^{(-1-\xi)/L},\\
M_1^e(\xi) &= \left(1-\xi^2\right)e^{\xi/L},\\
M_2^e(\xi) &= 0.5\xi(\xi+1)e^{(1-\xi)/L}.
\end{aligned} \tag{5.33}$$

The infinite basis interpolation functions, using exponential decay functions (5.33), distribution over the boundary element are presented in Fig. 5.8. The

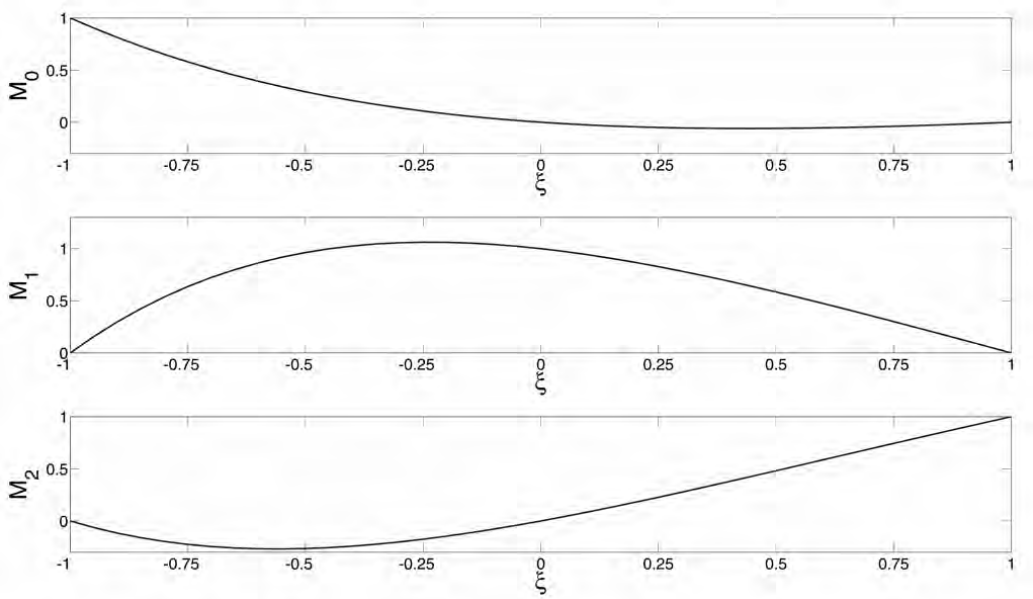


Figure 5.8: Distribution for basis interpolation functions M_0, M_1 i M_2, using exponential decay functions

differentials of basis interpolation functions (5.16) are as follows:

$$\begin{aligned}
\frac{\partial M_0^e(\xi)}{\partial \xi} &= -\frac{-2xL + L + x^2 - x}{2L} e^{(-x-1)/L}, \\
\frac{\partial M_1^e(\xi)}{\partial \xi} &= -\frac{2xL + x^2 - 1}{L} e^{x/L}, \\
\frac{\partial M_2^e(\xi)}{\partial \xi} &= -\frac{-2xL - L + x^2 + x}{2L} e^{(1-x)/L}.
\end{aligned} \tag{5.34}$$

These functions are used either for geometry transformation:

$$\begin{aligned}
x(\xi) &= \sum_{k=0}^{2} M_k(\xi) x_k = M_0(\xi)x_0 + M_1(\xi)x_1 + M_2(\xi)x_2, \\
y(\xi) &= \sum_{k=0}^{2} M_k(\xi) y_k = M_0(\xi)y_0 + M_1(\xi)y_1 + M_2(\xi)y_2,
\end{aligned} \tag{5.35}$$

or physical quantities of the object:

$$\Phi(\xi) = \sum_{k=0}^{2} M_k(\xi)\Phi_k = M_0(\xi)\Phi_0 + M_1(\xi)\Phi_1 + M_2(\xi)\Phi_2, \tag{5.36}$$

and:

$$\frac{\partial \Phi(\xi)}{\partial n} = \sum_{k=0}^{2} M_k(\xi) \frac{\partial \Phi_k}{\partial n} = M_0(\xi)\frac{\partial \Phi_0}{\partial n} + M_1(\xi)\frac{\partial \Phi_1}{\partial n} + M_2(\xi)\frac{\partial \Phi_2}{\partial n}. \quad (5.37)$$

Modified boundary integral equation for two-dimensional case:

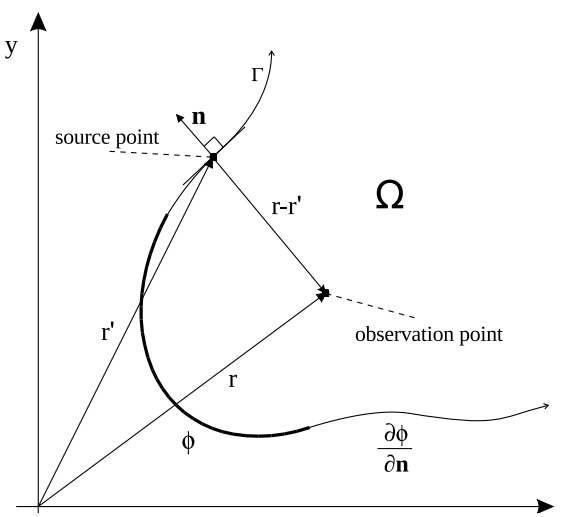


Figure 5.9: Solution region in local coordinate system for open boundary case

$$\begin{aligned} c(\mathbf{r})\Phi_i(\mathbf{r}) \quad &+ \quad \sum_{i=0}^{n-1}\sum_{k=0}^{2}\int_{-1}^{+1} \Phi(\mathbf{r}')\frac{\partial G(|\mathbf{r}-\mathbf{r}'|)}{\partial n} J^N(\xi)d\xi + \\ &+ \quad \sum_{j=0}^{m-1}\sum_{l=0}^{2}\int_{-1}^{\infty} \Phi(\mathbf{r}')\frac{\partial G(|\mathbf{r}-\mathbf{r}'|)}{\partial n} J^M(\xi)d\xi = \\ &= \quad \sum_{i=0}^{n-1}\sum_{k=0}^{2}\int_{-1}^{+1} \frac{\partial \Phi(\mathbf{r}')}{\partial n} G(|\mathbf{r}-\mathbf{r}'|) J^N(\xi)d\xi + \\ &+ \quad \sum_{j=0}^{m-1}\sum_{l=0}^{2}\int_{-1}^{\infty} \frac{\partial \Phi(\mathbf{r}')}{\partial n} G(|\mathbf{r}-\mathbf{r}'|) J^M(\xi)d\xi. \end{aligned} \quad (5.38)$$

where: n – number of standard boundary elements, m – number of infinite

boundary elements, J^N and J^M represent Jacobians [2] of the transformation (5.7) from the global to local coordinate system introduced by basis interpolation functions for finite boundary elements and infinite boundary elements, respectively. The object is covered by standard boundary elements and the surrounding by infinite elements.

5.7 Two-dimensional infinite boundary elements

The idea of constructing basis interpolation functions for two-dimensional infinite boundary elements is identical like for one-dimensional. Two additional cases will be considered. First when standard element is transferred to infinite only in one direction ξ or η and the second when element is extended to infinity in both directions ξ and η simultaneously (corner element) – Fig. 5.10

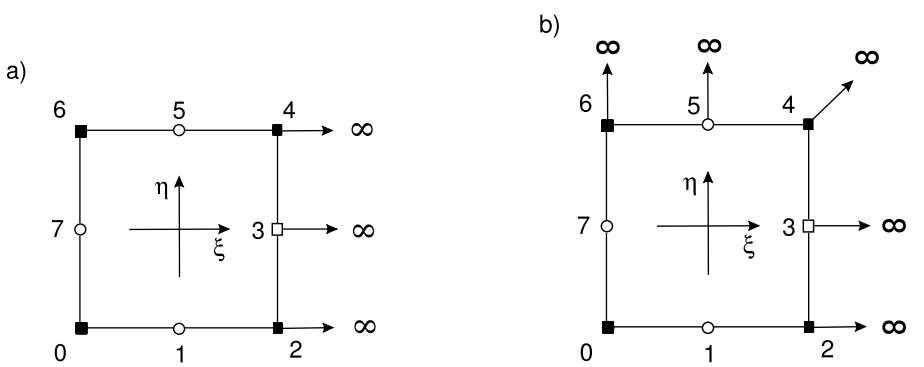


Figure 5.10: Decay functions infinite boundary elements: a) extended to infinity only along ξ axes and b) along both ξ and η axes

[2]Jacobian – determinant of Jacobi matrix.

5.7.1 Decay functions

Reciprocal decay functions

Reciprocal decay functions in local coordinate system [22, 7, 10, 15] for the decay in positive ξ axis direction is of the form:

$$D_i(\xi) = \left(\frac{\xi_i - \xi_o}{\xi - \xi_o}\right)^n, \tag{5.39}$$

for the decay in both ξ and η directions:

$$D_i(\xi, \eta) = \left(\frac{\xi_i - \xi_o}{\xi - \xi_o}\right)^n \left(\frac{\eta_i - \eta_o}{\eta - \eta_o}\right)^m, \tag{5.40}$$

where (ξ_o, η_o) is an origin point.

This point must be outside the infinite element, in the opposite side to that which extends to infinity. For the decay in positive ξ direction $\xi_o < -1$ and for η axis $\eta_o < -1$. For negative ξ and η directions, $\xi_o > 1$ and $\eta_o > 1$. It is necessary to avoid a singularity within the infinite element.

Moreover n and m must be greater than the highest power of ξ and η respectively, encountered in M. This ensures that for $\xi \to \infty$ $\lim_{\xi\to\infty} M_i = 0$ and $\eta \to \infty$ $\lim_{\eta\to\infty} M_i = 0$. Setting $n = 3$ i $m = 3$ from expression (5.40) for standard basis interpolation function (5.41):

$$\begin{aligned}
N_0(\xi,\eta) &= -(1-\xi)(1-\eta)(1+\xi+\eta)/4, \\
N_1(\xi,\eta) &= (1-\xi^2)(1-\eta)/2, \\
N_2(\xi,\eta) &= -(1+\xi)(1-\eta)(1-\xi+\eta)/4, \\
N_3(\xi,\eta) &= (1+\xi)(1-\eta^2)/2, \\
N_4(\xi,\eta) &= -(1+\xi)(1+\eta)(1-\xi-\eta)/4, \\
N_5(\xi,\eta) &= (1-\xi^2)(1+\eta)/2, \\
N_6(\xi,\eta) &= -(1-\xi)(1+\eta)(1+\xi-\eta)/4, \\
N_7(\xi,\eta) &= (1-\xi)(1-\eta^2)/2,
\end{aligned} \tag{5.41}$$

for infinite elements which are extended in one positive ξ direction, the basis

interpolation functions becomes:

$$
\begin{aligned}
M_0(\xi,\eta) &= -(1-\xi)(1-\eta)(1+\xi+\eta)/4 \left(\frac{1-\xi_o}{\xi-\xi_o}\right)^3, \\
M_1(\xi,\eta) &= (1-\xi^2)(1-\eta)/2 \left(\frac{-\xi_o}{\xi-\xi_o}\right)^3, \\
M_2(\xi,\eta) &= -(1+\xi)(1-\eta)(1-\xi+\eta)/4 \left(\frac{1-\xi_o}{\xi-\xi_o}\right)^3, \\
M_3(\xi,\eta) &= (1+\xi)(1-\eta^2)/2 \left(\frac{1-\xi_o}{\xi-\xi_o}\right)^3, \\
M_4(\xi,\eta) &= -(1+\xi)(1+\eta)(1-\xi-\eta)/4 \left(\frac{1-\xi_o}{\xi-\xi_o}\right)^3, \\
M_5(\xi,\eta) &= (1-\xi^2)(1+\eta)/2 \left(\frac{-\xi_o}{\xi-\xi_o}\right)^3, \\
M_6(\xi,\eta) &= -(1-\xi)(1+\eta)(1+\xi-\eta)/4 \left(\frac{-1-\xi_o}{\xi-\xi_o}\right)^3, \\
M_7(\xi,\eta) &= (1-\xi)(1-\eta^2)/2 \left(\frac{-1-\xi_o}{\xi-\xi_o}\right)^3.
\end{aligned}
\tag{5.42}
$$

The basis interpolation function (5.42) distribution over the boundary element are presented in Fig. 5.11. First derivatives of the basis interpolation

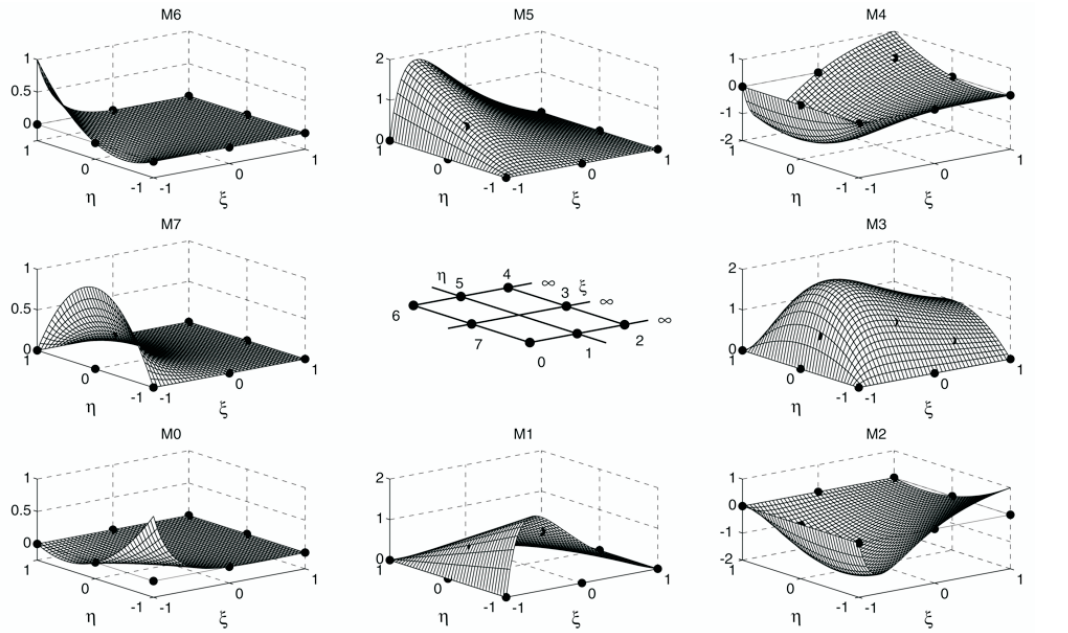


Figure 5.11: Distribution of basis interpolation functions (5.42)

functions (5.42) with respect to ξ are as follows:

$$
\begin{aligned}
\frac{\partial M_0(\xi,\eta)}{\partial \xi} &= -\frac{(\xi_o+1)^3(\eta-1)(2\xi\eta+\xi_o\eta-3\eta+\xi^2+2\xi_o\xi-3)}{4(\xi-\xi_o)^4},\\
\frac{\partial M_1(\xi,\eta)}{\partial \xi} &= \frac{{\xi_o}^3(\xi^2+2\xi_o\xi-3)(\eta-1)}{2(\xi-\xi_o)^4},\\
\frac{\partial M_2(\xi,\eta)}{\partial \xi} &= -\frac{(\xi_o-1)^3(\eta-1)(-2\xi\eta-\xi_o\eta-3\eta+\xi^2+2\xi_o\xi-3)}{4(\xi-\xi_o)^4},\\
\frac{\partial M_3(\xi,\eta)}{\partial \xi} &= -\frac{(\xi_o-1)^3(2\xi+\xi_o+3)(\eta^2-1)}{2(\xi-\xi_o)^4},\\
\frac{\partial M_4(\xi,\eta)}{\partial \xi} &= \frac{(\xi_o-1)^3(\eta+1)(2\xi\eta+\xi_o\eta+3\eta+\xi^2+2\xi_o\xi-3)}{4(\xi-\xi_o)^4},\\
\frac{\partial M_5(\xi,\eta)}{\partial \xi} &= -\frac{{\xi_o}^3(\xi^2+2\xi_o\xi-3)(\eta+1)}{2(\xi-\xi_o)^4},
\end{aligned}
\tag{5.43}
$$

$$\begin{aligned}
\frac{\partial M_6(\xi,\eta)}{\partial \xi} &= \frac{(\xi_o+1)^3(\eta+1)(-2\xi\eta-\xi_o\eta+3\eta+\xi^2+2\xi_o\xi-3)}{4(\xi-\xi_o)^4},\\
\frac{\partial M_7(\xi,\eta)}{\partial \xi} &= \frac{(\xi_o+1)^3(2\xi+\xi_o-3)(\eta^2-1)}{2(\xi-\xi_o)^4},
\end{aligned}$$

and with respect to η:

$$\begin{aligned}
\frac{\partial M_0(\xi,\eta)}{\partial \eta} &= \frac{(\xi_o+1)^3(\xi-1)(2\eta+\xi)}{4(\xi-\xi_o)^3},\\
\frac{\partial M_1(\xi,\eta)}{\partial \eta} &= -\frac{{\xi_o}^3(\xi^2-1)}{2(\xi-\xi_o)^3},\\
\frac{\partial M_2(\xi,\eta)}{\partial \eta} &= \frac{(\xi_o-1)^3(\xi+1)(\xi-2\eta)}{4(\xi-\xi_o)^3},\\
\frac{\partial M_3(\xi,\eta)}{\partial \eta} &= \frac{(\xi_o-1)^3(\xi+1)\eta}{(\xi-\xi_o)^3},\\
\frac{\partial M_4(\xi,\eta)}{\partial \eta} &= -\frac{(\xi_o-1)^3(\xi+1)(2\eta+\xi)}{4(\xi-\xi_o)^3},\\
\frac{\partial M_5(\xi,\eta)}{\partial \eta} &= \frac{{\xi_o}^3(\xi^2-1)}{2(\xi-\xi_o)^3},\\
\frac{\partial M_6(\xi,\eta)}{\partial \eta} &= -\frac{(\xi_o+1)^3(\xi-1)(\xi-2\eta)}{4(\xi-\xi_o)^3},\\
\frac{\partial M_7(\xi,\eta)}{\partial \eta} &= \frac{(\xi_o+1)^3(1-\xi)\eta}{(\xi-\xi_o)^3}.
\end{aligned} \tag{5.44}$$

For the decay in both positive ξ and η directions the basis interpolation function (5.45) are of the form:

$$\begin{aligned}
M_0(\xi,\eta) &= -(1-\xi)(1-\eta)(1+\xi+\eta)/4\left(\frac{1-\xi_o}{\xi-\xi_o}\right)^3\left(\frac{1-\eta_o}{\eta-\eta_o}\right)^3,\\
M_1(\xi,\eta) &= (1-\xi^2)(1-\eta)/2\left(\frac{-\xi_o}{\xi-\xi_o}\right)^3\left(\frac{-1-\eta_o}{\eta-\eta_o}\right)^3,
\end{aligned}$$

$$
\begin{aligned}
M_2(\xi,\eta) &= -(1+\xi)(1-\eta)(1-\xi+\eta)/4 \left(\frac{1-\xi_o}{\xi-\xi_o}\right)^3 \left(\frac{-1-\eta_o}{\eta-\eta_o}\right)^3, \\
M_3(\xi,\eta) &= (1+\xi)(1-\eta^2)/2 \left(\frac{1-\xi_o}{\xi-\xi_o}\right)^3 \left(\frac{-\eta_o}{\eta-\eta_o}\right)^3, \\
M_4(\xi,\eta) &= -(1+\xi)(1+\eta)(1-\xi-\eta)/4 \left(\frac{1-\xi_o}{\xi-\xi_o}\right)^3 \left(\frac{1-\eta_o}{\eta-\eta_o}\right)^3, \\
M_5(\xi,\eta) &= (1-\xi^2)(1+\eta)/2 \left(\frac{-\xi_o}{\xi-\xi_o}\right)^3 \left(\frac{1-\eta_o}{\eta-\eta_o}\right)^3, \\
M_6(\xi,\eta) &= -(1-\xi)(1+\eta)(1+\xi-\eta)/4 \left(\frac{-1-\xi_o}{\xi-\xi_o}\right)^3 \left(\frac{1-\eta_o}{\eta-\eta_o}\right)^3, \\
M_7(\xi,\eta) &= (1-\xi)(1-\eta^2)/2 \left(\frac{-1-\xi_o}{\xi-\xi_o}\right)^3 \left(\frac{-\eta_o}{\eta-\eta_o}\right)^3.
\end{aligned}
\tag{5.45}
$$

The basis interpolation function (5.45) distribution over the boundary element are presented in Fig. 5.12. First derivatives of the basis interpolation

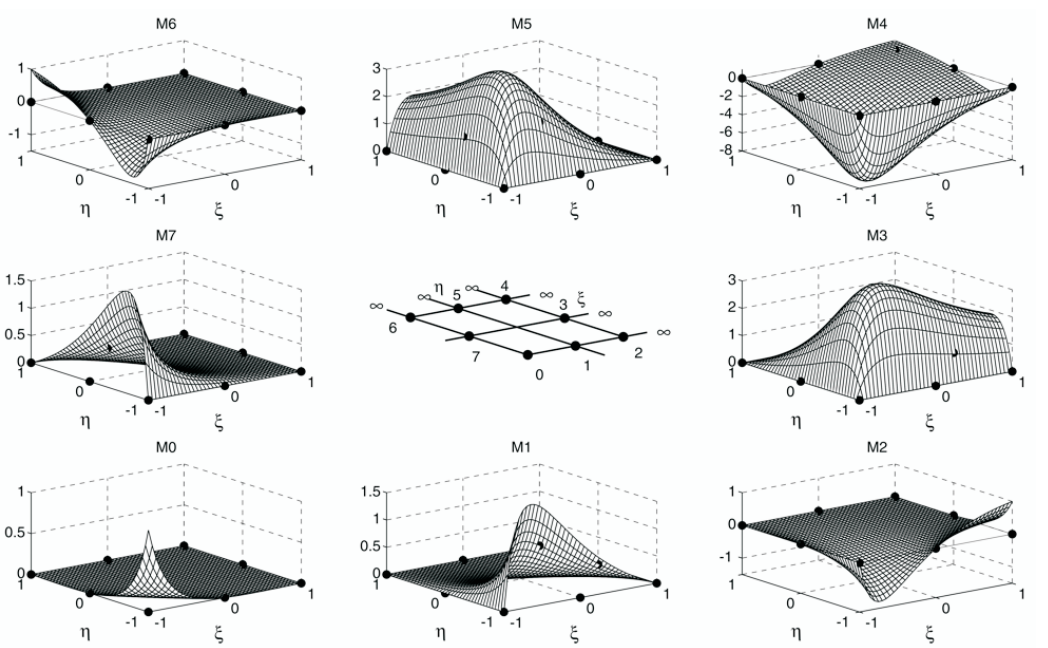


Figure 5.12: Distribution of basis interpolation functions (5.45)

functions (5.45) with respect to ξ are as follows:

$$\begin{aligned}
\frac{\partial M_0(\xi,\eta)}{\partial \xi} &= \frac{(\eta-1)(\eta_0+1)^3\left[\xi^2+2\xi\xi_0-3+\eta(2\xi+\xi_0-3)\right](\xi_0+1)^3}{4(\eta-\eta_0)^3(\xi-\xi_0)^4},\\
\frac{\partial M_1(\xi,\eta)}{\partial \xi} &= -\frac{(\eta-1)(\eta_0+1)^3\left(-3+\xi^2+2\xi\xi_0\right){\xi_0}^3}{2(\eta-\eta_0)^3(\xi-\xi_0)^4},\\
\frac{\partial M_2(\xi,\eta)}{\partial \xi} &= \frac{(\eta-1)(\eta_0+1)^3\left[\xi^2+2\xi\xi_0-3-\eta(2\xi+\xi_0+3)\right](\xi_0-1)^3}{4(\eta-\eta_0)^3(\xi-\xi_0)^4},\\
\frac{\partial M_3(\xi,\eta)}{\partial \xi} &= \frac{\left(-1+\eta^2\right){\eta_0}^3(3+2\xi+\xi_0)(\xi_0-1)^3}{2(\eta-\eta_0)^3(\xi-\xi_0)^4},
\end{aligned} \tag{5.46}$$

$$\begin{aligned}
\frac{\partial M_4(\xi,\eta)}{\partial \xi} &= \frac{(\eta+1)(\eta_0-1)^3\left[-\xi^2-2\xi\xi_0+3-\eta(2\xi+\xi_0+3)\right](\xi_0-1)^3}{4(\eta-\eta_0)^3(\xi-\xi_0)^4},\\
\frac{\partial M_5(\xi,\eta)}{\partial \xi} &= -\frac{(\eta+1)(\eta_0-1)^3\left(3+\xi^2+2\xi\xi_0\right){\xi_0}^3}{2(\eta-\eta_0)^3(\xi-\xi_0)^4},\\
\frac{\partial M_6(\xi,\eta)}{\partial \xi} &= \frac{(\eta+1)(\eta_0-1)^3\left[-\xi^2-2\xi\xi_0+3+\eta(2\xi+\xi_0-3)\right](\xi_0+1)^3}{4(\eta-\eta_0)^3(\xi-\xi_0)^4},\\
\frac{\partial M_7(\xi,\eta)}{\partial \xi} &= -\frac{\left(-1+\eta^2\right){\eta_0}^3(-3+2\xi+\xi_0)(\xi_0+1)^3}{2(\eta-\eta_0)^3(\xi-\xi_0)^4},
\end{aligned}$$

and with respect to η:

$$\begin{aligned}
\frac{\partial M_0(\xi,\eta)}{\partial \eta} &= \frac{(\eta_0+1)^3(\xi-1)\left[\eta^2-3+(-3+\eta_0)\xi+2\eta(\eta_0+\xi)\right](\xi_0+1)^3}{4(\eta-\eta_0)^4(\xi-\xi_0)^3},\\
\frac{\partial M_1(\xi,\eta)}{\partial \eta} &= -\frac{(-3+2\eta+\eta_0)(\eta_0+1)^3\left(\xi^2-1\right){\xi_0}^3}{2(\eta-\eta_0)^4(\xi-\xi_0)^3},\\
\frac{\partial M_2(\xi,\eta)}{\partial \eta} &= -\frac{(\eta_0+1)^3(\xi+1)\left[\eta^2-3+2\eta(\eta_0-\xi)-(\eta_0-3)\xi\right](\xi_0-1)^3}{4(\eta-\eta_0)^4(\xi-\xi_0)^3},\\
\frac{\partial M_3(\xi,\eta)}{\partial \eta} &= \frac{\left(-3+\eta^2+2\eta\eta_0\right){\eta_0}^3(\xi+1)(\xi_0-1)^3}{2(\eta-\eta_0)^4(\xi-\xi_0)^3},
\end{aligned} \tag{5.47}$$

$$\frac{\partial M_4(\xi,\eta)}{\partial \eta} = -\frac{(\eta_0-1)^3(\xi+1)\left[\eta^2-3+(3+\eta_0)\xi+2\eta(\eta_0+\xi)\right](\xi_0-1)^3}{4(\eta-\eta_0)^4(\xi-\xi_0)^3},$$

$$\frac{\partial M_5(\xi,\eta)}{\partial \eta} = -\frac{(3+2\eta+\eta_0)(\eta_0-1)^3\left(\xi^2+1\right){\xi_0}^3}{2(\eta-\eta_0)^4(\xi-\xi_0)^3},$$

$$\frac{\partial M_6(\xi,\eta)}{\partial \eta} = \frac{(\eta_0-1)^3(\xi-1)\left[\eta^2-3+2\eta(\eta_0-\xi)-(\eta_0+3)\xi\right](\xi_0+1)^3}{4(\eta-\eta_0)^4(\xi-\xi_0)^3},$$

$$\frac{\partial M_7(\xi,\eta)}{\partial \eta} = -\frac{(-3+\eta^2+2\eta\eta_0)\,{\eta_0}^3(\xi-1)(\xi_0+1)^3}{2(\eta-\eta_0)^4(\xi-\xi_0)^3}.$$

Exponential decay functions

Exponential decay functions for infinite element extended only in one positive ξ axis direction are as follows:

$$D_i(\xi,\eta) = e^{(\xi_i-\xi)/L}. \tag{5.48}$$

Multiplying decay functions (5.48) with standard basis interpolation functions (5.41) exponential infinite basis interpolation functions (5.49) are received.

$$\begin{aligned}
M_0(\xi,\eta) &= (\xi-1)(1-\eta)(\eta+\xi+1)/4\cdot e^{(1-\xi)/L},\\
M_1(\xi,\eta) &= \left(1-\xi^2\right)(1-\eta)/2\cdot e^{-\xi/L},\\
M_2(\xi,\eta) &= (-\xi-1)(1-\eta)(\eta-\xi+1)/4\cdot e^{(-\xi-1)/L},\\
M_3(\xi,\eta) &= (\xi+1)\left(1-\eta^2\right)/2\cdot e^{(-\xi-1)/L},\\
M_4(\xi,\eta) &= (-\xi-1)(-\eta-\xi+1)(\eta+1)/4\cdot e^{(-\xi-1)/L},\\
M_5(\xi,\eta) &= \left(1-\xi^2\right)(\eta+1)/2\cdot e^{-\xi/L},\\
M_6(\xi,\eta) &= (\xi-1)(-\eta+\xi+1)(\eta+1)/4\cdot e^{(1-\xi)/L},\\
M_7(\xi,\eta) &= (1-\xi)\left(1-\eta^2\right)/2\cdot e^{(1-\xi)/L}.
\end{aligned} \tag{5.49}$$

Basis interpolation function (5.48) distribution over the boundary element are presented in Fig. 5.13. First derivatives of basis interpolation functions

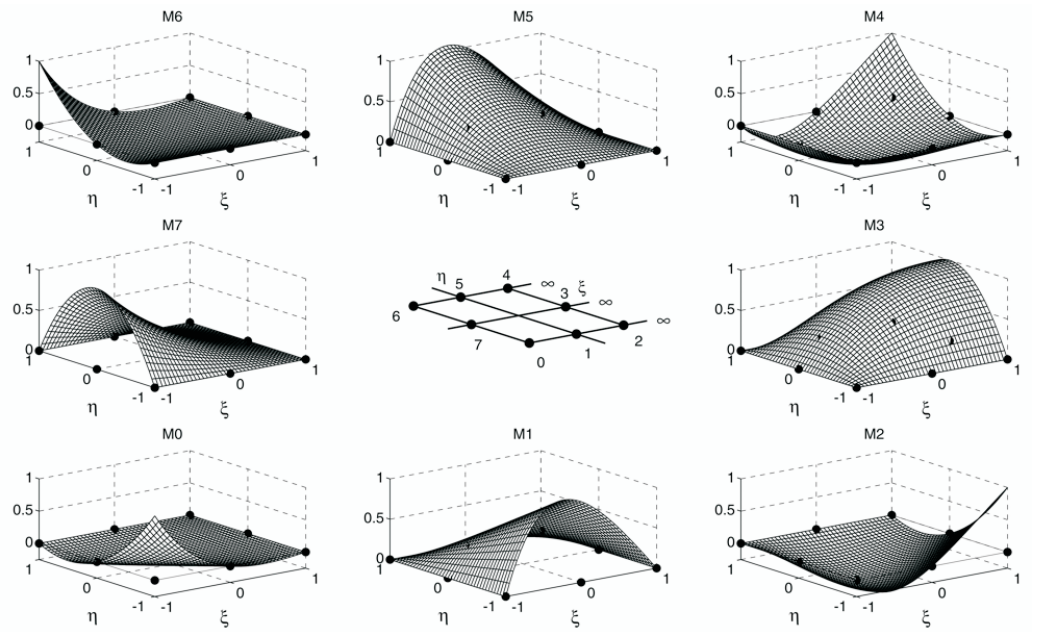


Figure 5.13: Distribution of basis interpolation functions (5.48)

(5.48) with respect to ξ are as follows:

$$\begin{aligned}
\frac{\partial M_0(\xi,\eta)}{\partial \xi} &= \frac{(\xi-1)(\eta-1)(\xi+\eta+1)\,e^{\frac{1-\xi}{L}}}{4L} - \frac{(\eta-1)(\xi+\eta+1)\,e^{\frac{1-\xi}{L}}}{4} \\
&- \frac{(\xi-1)(\eta-1)\,e^{\frac{1-\xi}{L}}}{4}, \\
\frac{\partial M_1(\xi,\eta)}{\partial \xi} &= -\frac{(\xi^2-1)(\eta-1)\,e^{-\frac{\xi}{L}}}{2L} + \xi(\eta-1)\,e^{-\frac{\xi}{L}}, \\
\frac{\partial M_2(\xi,\eta)}{\partial \xi} &= \frac{(\xi+1)(\eta-1)(\xi-\eta-1)\,e^{\frac{-\xi-1}{L}}}{4L} - \frac{(\eta-1)(\xi-\eta-1)\,e^{\frac{-\xi-1}{L}}}{4} \\
&- \frac{(\xi+1)(\eta-1)\,e^{\frac{-\xi-1}{L}}}{4},
\end{aligned}$$

$$
\begin{aligned}
\frac{\partial M_3(\xi,\eta)}{\partial \xi} &= \frac{(\xi+1)\,(\eta^2-1)\,e^{\frac{-\xi-1}{L}}}{2L} - \frac{(\eta^2-1)\,e^{\frac{-\xi-1}{L}}}{2},\\
\frac{\partial M_4(\xi,\eta)}{\partial \xi} &= \frac{(\xi+1)\,(\eta+1)\,(-\xi-\eta+1)\,e^{\frac{-\xi-1}{L}}}{4L} + \frac{(1+\eta)\,(\xi+\eta-1)\,e^{\frac{-\xi-1}{L}}}{4}\\
&+ \frac{(\xi+1)\,(\eta+1)\,e^{\frac{-\xi-1}{L}}}{4},\\
\frac{\partial M_5(\xi,\eta)}{\partial \xi} &= \frac{(\xi^2-1)\,(\eta+1)\,e^{-\frac{\xi}{L}}}{2L} - \xi\,(\eta+1)\,e^{-\frac{\xi}{L}},\\
\frac{\partial M_6(\xi,\eta)}{\partial \xi} &= -\frac{(\xi-1)\,(\eta+1)\,(\xi-\eta+1)\,e^{\frac{1-\xi}{L}}}{4L} + \frac{(\eta+1)\,(\xi-\eta+1)\,e^{\frac{1-\xi}{L}}}{4}\\
&+ \frac{(\xi-1)\,(\eta+1)\,e^{\frac{1-\xi}{L}}}{4},\\
\frac{\partial M_7(\xi,\eta)}{\partial \xi} &= -\frac{(\xi-1)\,(\eta^2-1)\,e^{\frac{1-\xi}{L}}}{2L} + \frac{(\eta^2-1)\,e^{\frac{1-\xi}{L}}}{2},
\end{aligned}
\tag{5.50}
$$

and with respect to η:

$$
\begin{aligned}
\frac{\partial M_0(\xi,\eta)}{\partial \eta} &= -\frac{(\xi-1)\,(\eta-1)\,e^{\frac{1-\xi}{L}}}{4} - \frac{(\xi-1)\,(\xi+\eta+1)\,e^{\frac{1-\xi}{L}}}{4},\\
\frac{\partial M_1(\xi,\eta)}{\partial \eta} &= \frac{(\xi^2-1)\,e^{-\frac{\xi}{L}}}{2},\\
\frac{\partial M_2(\xi,\eta)}{\partial \eta} &= \frac{(\xi+1)\,(\eta-1)\,e^{\frac{-\xi-1}{L}}}{4} + \frac{(\xi+1)\,(-\xi+\eta+1)\,e^{\frac{-\xi-1}{L}}}{4},\\
\frac{\partial M_3(\xi,\eta)}{\partial \eta} &= -(\xi+1)\,\eta\,e^{\frac{-\xi-1}{L}},\\
\frac{\partial M_4(\xi,\eta)}{\partial \eta} &= \frac{(\xi+1)\,(\eta+1)\,e^{\frac{-\xi-1}{L}}}{4} + \frac{(\xi+1)\,(\xi+\eta-1)\,e^{\frac{-\xi-1}{L}}}{4},
\end{aligned}
\tag{5.51}
$$

$$\begin{aligned}
\frac{\partial M_5(\xi,\eta)}{\partial \eta} &= -\frac{(\xi^2-1)\,e^{-\frac{\xi}{L}}}{2},\\
\frac{\partial M_6(\xi,\eta)}{\partial \eta} &= \frac{(\xi-1)\,(\xi-\eta+1)\,e^{\frac{1-\xi}{L}}}{4} - \frac{(\xi-1)\,(\eta+1)\,e^{\frac{1-\xi}{L}}}{4},\\
\frac{\partial M_7(\xi,\eta)}{\partial \eta} &= (\xi-1)\,\eta\,e^{\frac{1-\xi}{L}}.
\end{aligned}$$

For the decay in both ξ and η axis positive direction exponential decay function is described by expression:

$$D_i(\xi,\eta) = e^{-(\xi+\eta-\xi_i-\eta_i)/L}. \tag{5.52}$$

For the decay in both ξ and η axis negative direction exponential decay function is described by expression:

$$D_i(\xi,\eta) = e^{(\xi+\eta-\xi_i-\eta_i)/L}. \tag{5.53}$$

Using decay functions (5.52) in conjunction with (5.41) infinite basis interpolation functions (5.54) are derived.

$$\begin{aligned}
M_0(\xi,\eta) &= -(1-\xi)(1-\eta)(1+\xi+\eta)/4\cdot e^{(-2-\xi-\eta)/L},\\
M_1(\xi,\eta) &= (1-\xi^2)(1-\eta)/2\cdot e^{(-1-\xi-\eta)/L},\\
M_2(\xi,\eta) &= -(1+\xi)(1-\eta)(1-\xi+\eta)/4\cdot e^{(-\xi-\eta)/L},\\
M_3(\xi,\eta) &= (1+\xi)(1-\eta^2)/2\cdot e^{(1-\xi-\eta)/L}\\
M_4(\xi,\eta) &= -(1+\xi)(1+\eta)(1-\xi-\eta)/4\cdot e^{(2-\xi-\eta)/L},\\
M_5(\xi,\eta) &= (1-\xi^2)(1+\eta)/2\cdot e^{(1-\xi-\eta)/L},\\
M_6(\xi,\eta) &= -(1-\xi)(1+\eta)(1+\xi-\eta)/4\cdot e^{(-\xi-\eta)/L},\\
M_7(\xi,\eta) &= (1-\xi)(1-\eta^2)/2\cdot e^{(-1-\xi-\eta)/L}.
\end{aligned} \tag{5.54}$$

Basis interpolation function (5.52) distribution over the boundary element are presented in Fig. 5.14. First derivatives of basis interpolation functions

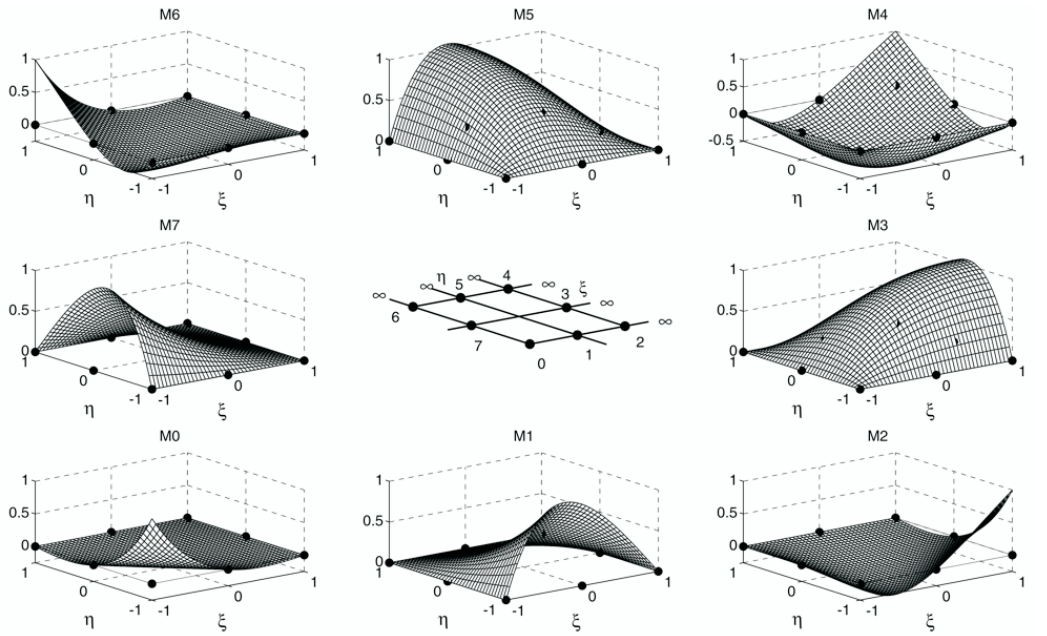


Figure 5.14: Distribution of basis interpolation functions (5.52), $L = -2$

(5.48) with respect to ξ are as follows:

$$
\begin{aligned}
\frac{\partial M_0(\xi,\eta)}{\partial \xi} &= \frac{-\left[\left(\eta^2+(2\xi-1)\,\eta-2\xi\right)L+(1-\xi)\,\eta^2+\left(\xi-\xi^2\right)\eta+\xi^2-1\right]e^{(-\eta-\xi-2)/L}}{4L},\\
\frac{\partial M_1(\xi,\eta)}{\partial \xi} &= \frac{\left[(2\xi\eta-2\xi)\,L+\left(1-\xi^2\right)\eta+\xi^2-1\right]e^{(-\eta-\xi+1)/L}}{2L},\\
\frac{\partial M_2(\xi,\eta)}{\partial \xi} &= \frac{\left[\left(\eta^2+(-2\xi-1)\,\eta+2\xi\right)L+(-\xi-1)\,\eta^2+\left(\xi^2+\xi\right)\eta-\xi^2+1\right]e^{(-\eta-\xi)/L}}{4L},\\
\frac{\partial M_3(\xi,\eta)}{\partial \xi} &= \frac{-\left[\left(\eta^2-1\right)L+(-\xi-1)\,\eta^2+\xi+1\right]e^{(-\eta-\xi+1)/L}}{2L},\\
\frac{\partial M_4(\xi,\eta)}{\partial \xi} &= \frac{\left[\left(\eta^2+(2\xi+1)\,\eta+2\xi\right)L+(-\xi-1)\,\eta^2+\left(-\xi^2-\xi\right)\eta-\xi^2+1\right]e^{(-\eta-\xi+2)/L}}{4L},\\
\frac{\partial M_5(\xi,\eta)}{\partial \xi} &= \frac{-\left[(2\xi\eta+2\xi)\,L+\left(1-\xi^2\right)\eta-\xi^2+1\right]e^{(-\eta-\xi+1)/L}}{2L},\\
\frac{\partial M_6(\xi,\eta)}{\partial \xi} &= \frac{-\left[\left(\eta^2+(1-2\xi)\,\eta-2\xi\right)L+(1-\xi)\,\eta^2+\left(\xi^2-\xi\right)\eta+\xi^2-1\right]e^{(-\eta-\xi)/L}}{4L},\\
\frac{\partial M_7(\xi,\eta)}{\partial \xi} &= \frac{\left[\left(\eta^2-1\right)L+(1-\xi)\,\eta^2+\xi-1\right]e^{(-\eta-\xi-1)/L}}{2L},
\end{aligned}
\tag{5.55}
$$

and with respect to η:

$$\begin{aligned}
\frac{\partial M_0(\xi,\eta)}{\partial \xi} &= \frac{-\left[\left((2\xi-2)\,\eta+\xi^2-\xi\right)L+(1-\xi)\,\eta^2+\left(\xi-\xi^2\right)\eta+\xi^2-1\right]e^{(-\eta-\xi-2)/L}}{4L},\\
\frac{\partial M_1(\xi,\eta)}{\partial \xi} &= \frac{\left[\left(\xi^2-1\right)L+\left(1-\xi^2\right)\eta+\xi^2-1\right]e^{(-\eta-\xi-1)/L}}{2L},\\
\frac{\partial M_2(\xi,\eta)}{\partial \xi} &= \frac{\left[\left((2\xi+2)\,\eta-\xi^2-\xi\right)L+(-\xi-1)\,\eta^2+\left(\xi^2+\xi\right)\eta-\xi^2+1\right]e^{(-\eta-\xi)/L}}{4L},\\
\frac{\partial M_3(\xi,\eta)}{\partial \xi} &= \frac{-\left[(2\xi+2)\,\eta L+(-\xi-1)\,\eta^2+\xi+1\right]e^{(-\eta-\xi+1)/L}}{2L},\\
\frac{\partial M_4(\xi,\eta)}{\partial \xi} &= \frac{\left[\left((2\xi+2)\,\eta+\xi^2+\xi\right)L+(-\xi-1)\,\eta^2+\left(-\xi^2-\xi\right)\eta-\xi^2+1\right]e^{(-\eta-\xi+2)/L}}{4L},\\
\frac{\partial M_5(\xi,\eta)}{\partial \xi} &= \frac{-\left[\left(\xi^2-1\right)L+\left(1-\xi^2\right)\eta-\xi^2+1\right]e^{(-\eta-\xi+1)/L}}{2L},\\
\frac{\partial M_6(\xi,\eta)}{\partial \xi} &= \frac{-\left[\left((2\xi-2)\,\eta-\xi^2+\xi\right)L+(1-\xi)\,\eta^2+\left(\xi^2-\xi\right)\eta+\xi^2-1\right]e^{(-\eta-\xi)/L}}{4L},\\
\frac{\partial M_7(\xi,\eta)}{\partial \xi} &= \frac{\left[(2\xi-2)\,\eta L+(1-\xi)\,\eta^2+\xi-1\right]e^{(-\eta-\xi-1)/L}}{2L}.
\end{aligned} \tag{5.56}$$

5.8 Numerical examples

5.8.1 2D calculations

To follow the infinite boundary elements incorporation into BEM a simple 2D example like on left part (see Fig. 5.15) will be presented. It consists of 2 semi infinite edges perpendicular to each other - horizontal $0X$ with potential $0V$ and vertical $0Y$ with potential $10V$. The mesh is build from 46 standard elements and 2 infinite elements. To avoid singularity on the potential leap on the edges contact point (0,0) 2 small elements in that corner have potential decreasing smoothly from $10V$ to $5V$ and then to $0V$. Achieved results were compared with the solution based on standard elements only and a mesh extended to double distance from the zone of interests (hatched area) right part in Fig. 5.15. The example has known analytical solution (5.57) important for the accuracy comparison.

Boundary and potential smoothing near the edge's contact point required by our numerical solution differs from the theoretical solution. That causes an extra error in the (0,0) point neighbourhood. Values of $(\partial\phi(r)/\partial n)$ calculated in nodes of the boundary mesh for the region of interests are presented in Fig. 5.16 and the accuracy comparison in Fig. 5.17.

$$\begin{aligned}\phi(x,y) &= \frac{V_0}{\pi}\, 2\,\mathrm{arctg}\frac{y}{x},\\ \frac{\partial\phi(x,y)}{\partial x} &= \frac{V_0}{\pi}\, 2\frac{y}{x^2+y^2},\\ \frac{\partial\phi(x,y)}{\partial y} &= \frac{V_0}{\pi}\, 2\frac{x}{x^2+y^2}.\end{aligned} \tag{5.57}$$

Normal derivative $(\partial\phi(r)/\partial n)$ is expressed by (5.58):

$$\frac{\partial\phi(x,y)}{\partial \mathbf{n}} = \frac{V_0}{\pi}\, 2\left(\frac{x}{\sqrt{x^2+y^2}}\mathbf{1_y} - \frac{y}{\sqrt{x^2+y^2}}\mathbf{1_x}\right). \tag{5.58}$$

Numerical solutions with incorporated infinite elements were compared with analytical and with numerical where standard boundary elements only were used (ballooning [9, 15]). The idea of decreasing analyzed area by using infinite elements is presented in Fig. 5.15. Solution area is marked as hatched part. Figure 5.16 presents potential normal derivatives in nodes located on the axes and Fig. 5.17 achieved accuracy. Because of symmetry along $y = x$, figures 5.16 and 5.17 represents solution on $0 - x$ axes.

Results for reciprocal and exponential decay infinite elements are almost the same and very close to these with used mapped infinite elements and therefore they are difficult to distinguish in figures 5.16 and 5.17. The role of the model surroundings is only to receive the correct solution in the area of interest. Therefore, points close to value 1 in figures 5.16 and 5.17 are omitted in further discussion. Both types of infinite elements offer similar results and accuracy. Calculation time was twice time shorter for models with infinite elements comparing to model consisting only of standard boundary elements (forward problem solution takes 20 sec. instead of 44 sec.).

5.8.2 3D calculations

Four simple theoretical models of human breast were investigated. For all models one placement of the light source was presented - located near the

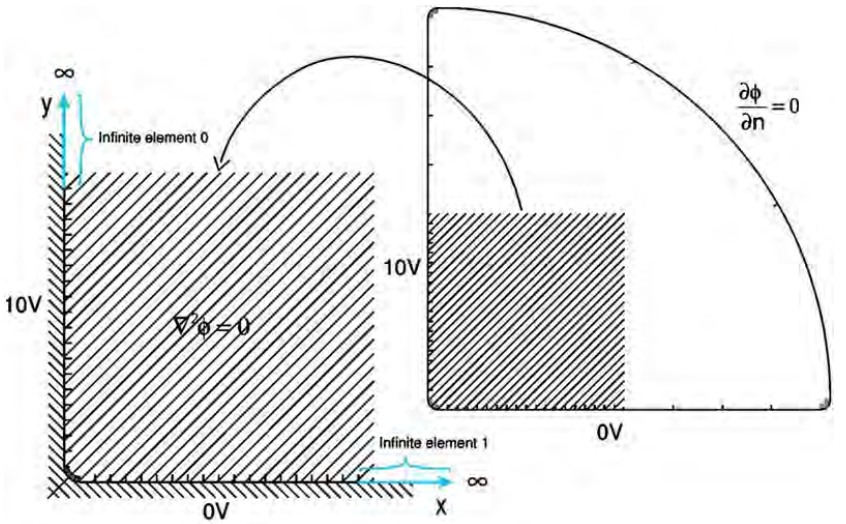


Figure 5.15: The idea of decreasing analyzed area by using infinite elements; the discretization and shapes of the areas correspond to the two-dimensional computational model

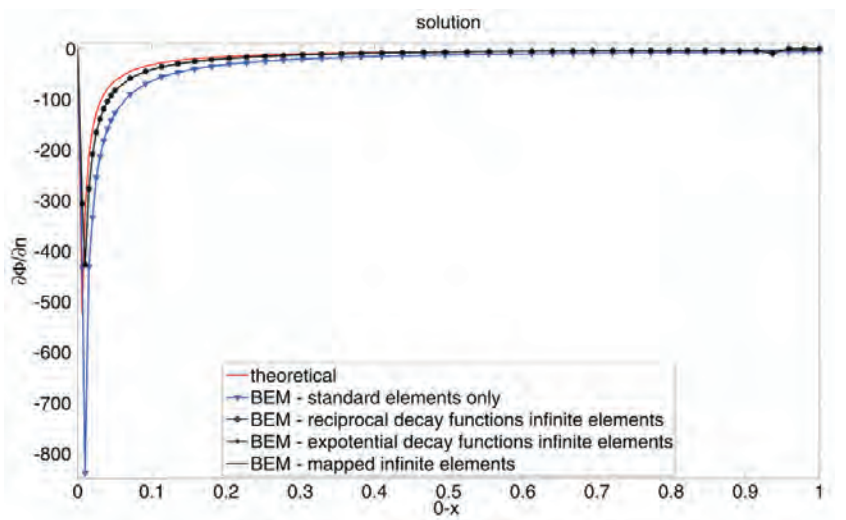


Figure 5.16: Solutions comparison: theoretical (red line), with standard elements only (blue line) and with infinite elements incorporated (black line: with o – for reciprocal decay functions, with x – for exponential decay functions and full line – for mapped infinite elements)

bottom of the hemisphere model. The first model presented in Fig.5.19 corresponds to the pure hemisphere. The second model was extended by adding

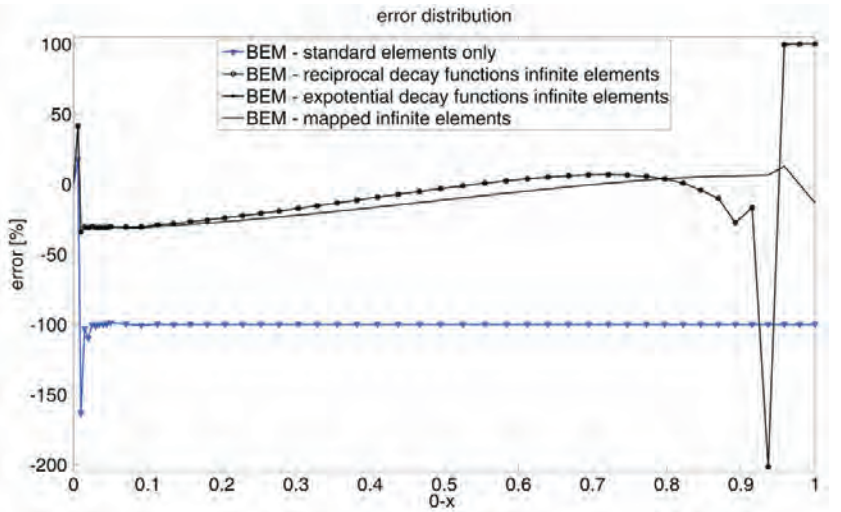


Figure 5.17: Solutions accuracy comparison: with standard elements only (blue line) and with infinite element incorporated (black line: with o – for reciprocal decay functions, with x – for exponential decay functions and full line – for mapped infinite elements)

a cylinder in the bottom (see Fig. 5.20). The intention of this was to avoid possible errors at the bottom of the hemisphere. Another model develops that idea by adding the cylinder with identical height but bigger diameter (see Fig. 5.21). The aim of these models is to eliminate the errors near the basis circumference. All models were constructed from 1536-second order 8-node quadrilateral boundary elements and 4610 nodes. A half of the elements covers the hemisphere.

Governing equation for the problem is diffusion approximation of the transport equation [20] (Helmholtz - assuming that scattering and absorption are homogeneous):

$$\nabla^2\Phi\left(\mathbf{r},\omega\right) - k^2\Phi\left(\mathbf{r},\omega\right) = -\frac{q_0\left(\mathbf{r},\omega\right)}{D}, \qquad \forall \mathbf{r} \in \Omega/\Gamma, \tag{5.59}$$

where Φ stands for photon density, $k = \sqrt{\frac{\mu_a}{D} - j\frac{\omega}{cD}}$ is the complex wave number, $D = \left[3\left(\mu_a + \mu_S'\right)\right]^{-1}\left[\mathrm{mm}^{-1}\right]$ is the diffusion coefficient, μ_S' is the reduced scattering coefficient, μ_a is the absorbing coefficient, c is the speed of light in the medium q_0 is a source of light (number of photons per volume unit emitted by concentrated light source located at position $\mathbf{r}$ with modulation frequency ω).

Generally in diffusive optical tomography the distributions of μ_a and μ_s' are investigated.

There are Robin boundary conditions on surfaces [3, 20]:

$$\Phi\left(\mathbf{r},\omega\right)+2\alpha D\frac{\partial\Phi\left(\mathbf{r},\omega\right)}{\partial n}=0,\qquad\forall\mathbf{r}\in\Gamma. \tag{5.60}$$

with different coefficients for breast tissue and for skeletal muscles on the basis [3] imposed. In the analyzed example the following breast tissue properties were taken [3]: $\mu_a = 0.025[\mathrm{mm}^{-1}], \mu_s' = 2[\mathrm{mm}^{-1}], \alpha = 1, f = 100\mathrm{kHz}$. Last open boundary model consists of 768 standard boundary elements and 64 infinite mapped elements based on 8-node second order quadrilateral boundary elements [10, 5, 4]. The number of nodes is reduced to 2433 nodes in that case (see Fig. 5.22). Serendipity infinite mapping functions [10] were

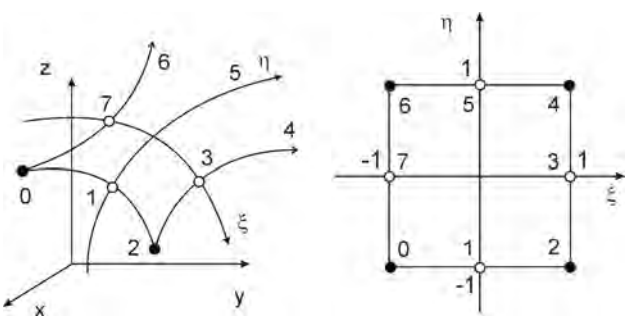


Figure 5.18: Five node mapped infinite boundary element based on 8 node quadrilateral isoparametric boundary element

used for element transformation like those presented in Fig. 5.18. The relevant boundary integral equation for surfaces covered by standard and infinite elements can be written as:

$$C(\mathbf{r})\Phi(\mathbf{r})+\int\limits_{\Gamma}\frac{\partial G(|\mathbf{r}-\mathbf{r}'|,\omega)}{\partial n}\Phi(\mathbf{r}')d\Gamma+\int\limits_{\Gamma_\infty}\frac{\partial G(|\mathbf{r}-\mathbf{r}'|,\omega)}{\partial n}\Phi(\mathbf{r}')d\Gamma_\infty=$$

$$=\int\limits_{\Gamma}G(|\mathbf{r}-\mathbf{r}'|,\omega)\frac{\partial\Phi(\mathbf{r}')}{\partial n}d\Gamma+\int\limits_{\Gamma_\infty}G(|\mathbf{r}-\mathbf{r}'|,\omega)\frac{\partial\Phi(\mathbf{r}')}{\partial n}d\Gamma_\infty-\sum_{s=0}^{n_{src}-1}Q_sG(|\mathbf{r}_s-\mathbf{r}|,\omega)$$

where Q_s is the magnitude of the concentrated source ($q_0 = Q_s\delta(\mathbf{r}_s)$) and n_{src} is a number of these sources, Φ stands for the photon density and G is the fundamental solution for the diffusion equation [3, 20].
In 3D space the fundamental solution for the diffusion equation is:

$$G\left(|\mathbf{r}-\mathbf{r}'|,\omega\right)=\frac{1}{4\pi\left|\mathbf{r}-\mathbf{r}'\right|}e^{-k|\mathbf{r}-\mathbf{r}'|}. \tag{5.61}$$

The normal derivative of the Green function G can be written as:

$$\mathbf{n} \cdot \nabla G = \mathbf{n} \cdot \frac{\mathbf{r} - \mathbf{r}'}{|\mathbf{r} - \mathbf{r}'|} \left(\frac{-1}{4\pi \left|\mathbf{r} - \mathbf{r}'\right|^2} - \frac{k}{4\pi \left|\mathbf{r} - \mathbf{r}'\right|} \right) e^{-k|\mathbf{r}-\mathbf{r}'|}. \tag{5.62}$$

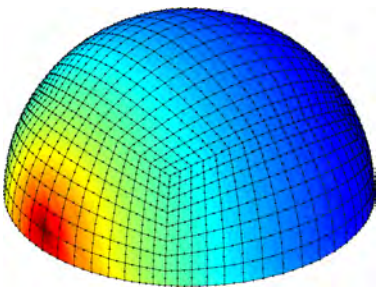


Figure 5.19: Base model of the breast - hemisphere

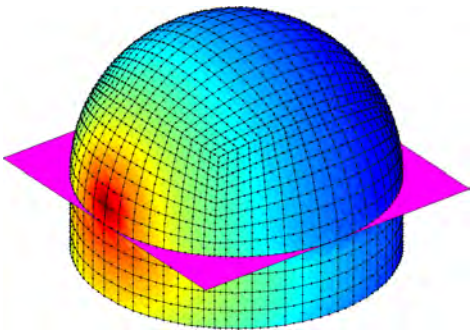


Figure 5.20: Extended model with additional part of chest, hemisphere with cylinder on the bottom

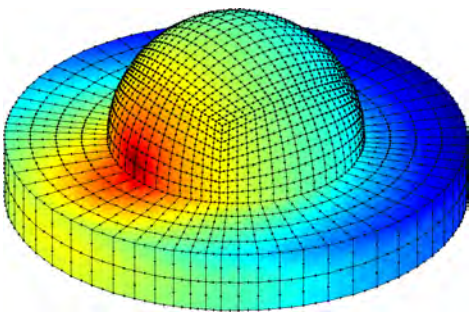


Figure 5.21: Extended model with additional part of chest, hemisphere with wider cylinder on the bottom

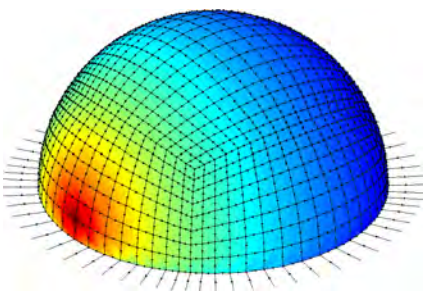


Figure 5.22: Open boundary hemisphere breast model with infinite boundary elements on the bottom

5.8.3 Results

Values of the module and phase of $\partial\Phi/\partial n$ at hemisphere circumference cross-section for $y = 0$ are presented in figures 5.23 and 5.24, respectively. To estimate the solution differences, models with extended bottom part – figures 5.20 , 5.21 and 5.22 were compared to basic hemisphere – figure 5.19.

Generally three extended models offers similar results, distinctly different from those achieved for the simple hemisphere in Fig. 5.19. It should be noticed that there is a logarithmic scale in the graph $\partial\Phi/\partial n$ module in Fig. 5.23. The medium value of approximation differences for module is about 30% and for phase oscillates about 3%.

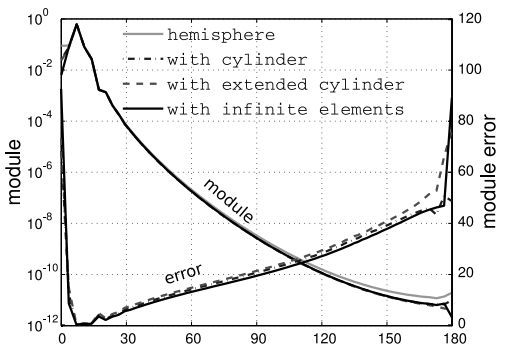


Figure 5.23: Module of $\partial\Phi/\partial n(\Psi)$ for all models and solution differences related to hemisphere model

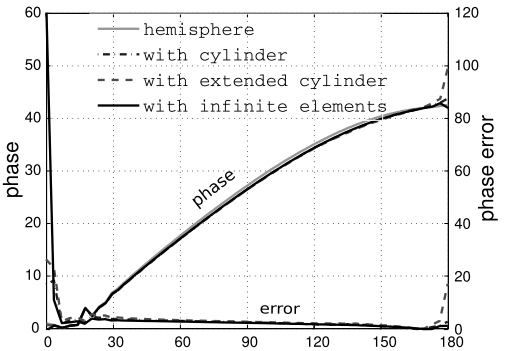


Figure 5.24: Phase of $\partial\Phi/\partial n(\Psi)$ for all models and solution differences compared to hemisphere model

5.9 Conclusion

Generally, the three extended breast models (those with additional cylindrical part on the hemisphere basis or with infinite elements incorporated) offer similar accuracy – figures 5.23 and 5.24. The worst results were achieved then using the pure hemisphere model. The medium differences between the hemisphere and extended models are about 30% and for the module and about 3% for the phase. Thirty-percent differences are significant enough so that the definition area of the problem has to be extended. In this case it includes not only the breast tissue represented by the hemisphere but also a part of chest with muscles and bones relevant to additional cylindrical part or ring consisting of infinite elements. It should be noticed that the graph in Fig. 5.23 of the module $\partial\Phi/\partial n$ has a logarythmic scale.

The advantage of using infinite elements consists in reducing the calculation time and keeping the accuracy similar to the accuracy provided by the extended models.

Reducing the number of mesh elements almost to 50% is fundamental for inverse problem solution when the forward problem has to be calculated many times. All extended models were built from the same number of standard 8-node second order quadrilateral boundary elements. Mesh density on the additional surface related to cylindrical part of the model was lower than that on the hemisphere surface. This is a typical practical solution, as the additional part represents the region outside the zone of interest, and is included only to improve the accuracy.

Except for the models mentioned above which consist of 1536 elements and 4610 nodes, calculations were performed also for models covered by 384 elements and 1154 nodes and also 6144 elements and 18434 nodes. The results calculated from the simplest model built from 384 elements exhibit little oscillations rather than smooth character. The model with the highest mesh density, with 6144 elements, required too much memory and took too long calculation time without a significant improvement of accuracy. The calculations took 18 seconds for 384 node models and 4 minutes and 47 seconds in case of 4160 node models. The model with infinite elements built in total 832 elements that corresponds to the standard model built from 1536 elements and 4610 nodes required 1 minute and 24 seconds for calculations. Self-made generator was used to build presented meshes containing only quadrilateral elements and to create open boundary models with infinite boundary elements.

Implementation of infinite boundary elements into boundary element method improves computational efficiency compared to the mesh truncation. The process of incorporating infinite elements into BEM calculation scheme is quite logical. Presented application in Optical or Electrical Impedance mammography can be used as a screening examination in breast cancer detection but infinite elements can be used also for other purposes.

Bibliography

[1] M. Abramowitz, I.A. Stegun: Handbook of mathematical functions with formulas, graphs and mathematical tables, John Wiley, New York, 1973.

[2] M.H. Aliabadi: The Bouundary Element Method Volume 2, John Willey & Sons, 2005.

[3] S.R. Arridge: Optical tomography in medical imaging, Inverse Problems, Vol. 15, No. 2, 1999, pp. R41-R93.

[4] G. Beer, J.O. Watson, G. Swoboda: Three-dimensional analysis of tunnels using infinite boundary elements, Computers and Geotechnics, Vol. 3, 1987, pp. 37-58.

[5] G. Beer, J.O. Watson: Infinite Boundary Elements, International Journal for Numerical Methods in Engineering, Vol. 28, 1989, pp. 1233-1247.

[6] G. Beer: Programming the Boundary Element Method. An Introduction for Engineers, John Wiley & Sons Ltd., Chichester, 2001.

[7] P. Bettess: Infinite Elements, International Journal for Numerical Methods in Engineering, Vol.11, 1977, pp. 53-64.

[8] P. Bettess: More on Infinite Elements, International Journal for Numerical Methods in Engineering, Vol. 15, 1980, pp. 1613-1626.

[9] P. Bettess: Finite Element Modelling of Exterior Electromagnetic Problems, IEEE Transaction on Magnetics, Vol. 24, No. 1, 1988, pp. 238-243.

[10] P. Bettess: Infinite Elements, Penshaw Press, 1992.

[11] C.A. Brebbia: The Boundary Element Method for engineers, Pentech Press, London, 1978.

[12] P.J. Davis, P. Rabinowitz: Methods of Numerical Integration, Academic Press, New York, London, 1984.

[13] T.G. Davies, S. Bu: Infinite Boundary Elements for the Analysis of Halfspace Problems, Computers and Geotechnics, Vol. 19, No. 2, 1996, pp. 137-151.

[14] J.H.M. Frijns,S.L. De Snoo, R. Schoonhoven: Improving the Accuracy of the Boundary Element Method by Use of Second-Order Interpolation Functions, IEEE Transactions on Biomedical Engineering, Vol. 47, No. 10, 2000, pp. 1336-1346.

[15] S. Gratkowski: Infinite elements for axisymmetric electrical field problems with open boundaries, COMPEL – The International Journal for Computation and Mathematics in Electrical and Electronic Engineering, Vol. 17, No. 5/6, 1998, pp. 781-788.

[16] W. Moser, C. Duenser, G. Beer: Mapped infinite elements for three-dimensional multi-region boundary element analysis, International Journal for Numerical Methods in Engineering, Vol. 61, 2004, pp. 317-328.

[17] C. Pozrikidis: A Practical Guide to Boundary Element Methods with the Software Library BEMLIB, Chapman & Hall/CRC, 2000.

[18] P. Rabinowitz, G. Weiss: Tables of abscissas and weights for numerical evaluation of integrals of the form: $\int_0^\infty x^n e^{-x} f(x)dx$, Math. Tables Other Aids to Comput., Vol. 13, 1959, pp. 285-294.

[19] M. Ross: Modeling Methods for Silent Boundaries in Infinite Media, ASEN 5519-006: Fluid-Structure Interaction Aerospace Engineering Sciences – University of Colorado at Boulder, 2004, http://www.colorado.edu/engineering/CAS/courses.d/FSI.d/FSI.projects.d /FSI.projects.Ross.silent.pdf

[20] J. Sikora: Boundary Element Method for Impedance and Optical Tomography, Oficyna Wydawnicza Politechniki Warszawskiej, 2007.

[21] J.O. Watson: Advanced implementation of the boundary element method for two- and three-dimensional elastostatics, Developments in Boundary Element Methods - 1 (Editors P.K. Banerjee and R. Butterfield), Elsevier Applied Science Publishers, Vol. 61, , 1979, pp. 31-63.

[22] O.C. Zienkiewicz, C. Emson, P. Bettess: A novel boundary infinite element, International Journal for Numerical Methods in Engineering, Vol. 16, 1983, pp. 393-404.

[23] O.C. Zienkiewicz: The Finite Element Method, MCGraw-Hill, 1993.

Chapter 6

BEMLAB-open source, objective Boundary Element Method library

J. Sikora, P. Wieleba, W. Wójcik

Nomenclature

Abbreviations

BEM – Boundary Element Method

FEM – Finite Element Method

DOT – Diffuse Optical Tomography

EIT – Electrical Impedance Tomography

CT – Computer Tomography

NMRI – Nuclear Magnetic Resonance Imaging

NIR – Near Infra Red

PDE – Partial Differential Equation

BIE – Boundary Integral Equation

RTE – Radiative Transfer Equation

Symbols

Γ – Boundary

Ω – Domain

Γ_j – Boundary element j

φ – Potential [V]; Photon density $\left[\frac{1}{\mathrm{m}^3}\right]$

$\vec{n}$ – Normal vector outward the domain Ω

k – Wave number

G – Fundamental solution, Green function

$\frac{\partial}{\partial n}$ – Normal derivative

ω – Frequency of source light intensity modulation

6.1 Introduction

Tomography imaging techniques require proper numerical models. Data gathered from the hardware are used to create an image of the examined object internal structure. The inverse problem using the adequate numerical model is solved and its results compose the picture of the object internal. The inverse problem allows to find object internal model parameters m using data d gathered form the boundary of the model using the scan hardware. The relationship between d and m can be written as:

$$d = \Upsilon(m)\,, \tag{6.1}$$

where Υ is a non-linear operator which represents the numerical model of the physical problem.

The base unit of the inverse problem is the forward problem which allows to calculate d based on known m. It is very important to calculate forward

problems as fast as possible to process further image reconstruction effectively. It is particularly important in Diffuse Optical Tomography (DOT) and Electrical Impedance Tomography (EIT) where calculations are very time consuming [11].

The most common tomography techniques in medicine are the X-ray based Computer Tomography and the Nuclear Magnetic Resonance Imaging (NMRI). Computer Tomography uses X-rays which are the ionizating radiation. Long tissues exposure on X-ray based radiation is very dangerous therefore CT cannot be used frequently. Whereas Nuclear Magnetic Resonance Imaging uses magnetic field from 0.5 to 2 Tesla. Larger values of magnetic field are prohibited in medicine because it is not neutral for living organisms. Both mentioned techniques are volumetric, therefore precise image is obtained and exact object interior structure is presented.

In contrast to DOT commonly used X-ray based Computer Tomography and NMRI use fast algorithms. CT uses back-projection while NMRI uses Fourier Transform based algorithms. Availability of fast algorithms to reconstruct the image made it possible to popularize these methods in medicine testing.

However CT and NMRI are not ideal testing methods. X-ray based CT can only be used rarely because of its dangerous influence on tissues. NMRI is also not neutral for living organisms. Moreover both methods require devices of big dimensions. NMRI requires extensive cooling, therefore special installations have to be applied. Diffuse Optical Tomography does not have drawbacks of CT and NMRI. DOT uses near infrared (NIR) light which is safe for tissues, which can be exposed to it permanently. The size of optical scanners is relatively small – they are portable. However DOT cannot be used for precise volumetric imaging. It also requires much more efficient processing units. Nowadays used algorithms are very slow therefore they make it impossible to introduce DOT to every day medical testing.

The main areas of DOT application in medicine are:

- neonatal head testing of brain haemorrhage,
- breast testing for detecting tumours.

Usage of DOT for testing infants brain haemorrhage is especially important. It is required to test the brain with haemorrhage permanently, so doctors

could know if it increases or decreases and if applied treatment is appropriate. It is also beneficial to use DOT for detecting tumours in breasts. Nowadays popularly used mammography uses X-ray based ionizating radiation and as testing should be done regularly it is not neutral. Moreover while the mammography test is taken, breasts are deformed whereas while using DOT scanners they are not. Despite of fact that the image is not as precise as that reconstructed using CT or NMRI it is desired to introduce DOT imaging in the mentioned areas. However the main drawback of DOT image reconstruction, which is the long time of image reconstruction has to be solved.

Further sections describe universal, open source and objective software implementing Boundary Element Method (BEM) for solving partial differential equations, which can be used in tomography applications.

6.2 Radiative Transport Equation in Diffuse Optical Tomography

Firstly the numerical model for Diffuse Optical Tomography have to be introduced. Near infrared light used in DOT is an electromagnetic wave. Therefore the light transport phenomenon can be described using Radiative Transport Equation (RTE). Depending on the type of scanner or its work mode the source of near infrared light can be described as defined in:

- time domain – the signal is in the form of ultra fast impulses,
- frequency domain – the light intensity modulation.

RTE defined in the time domain has the following form:

$$\left(\check{s}\cdot\nabla+\mu_a(r)+\mu_s(r)+\frac{1}{c}\frac{\partial}{\partial t}\right)\varphi(r,\check{s},t)=\mu_s(r)\int_s^{n-1}\Theta(\check{s},\check{s}')\varphi(r,\check{s},t)d\check{s}'+q(r,\check{s},t), \tag{6.2}$$

and in the frequency domain [13]:

$$\left(\check{s}\cdot\nabla+\mu_a(r)+\mu_s(r)+i\frac{\omega}{c}\right)\varphi(r,\check{s},\omega)=\mu_s(r)\int_{s^{n-1}}\Theta(\check{s},\check{s}')\varphi(r,\check{s},\omega)d\check{s}'+q(r,\check{s},\omega)\,, \tag{6.3}$$

where: $\check{s}$ – directional vector; $\Theta(\check{s}, \check{s}')$ – dispersing phase function describing probability that photon with the beginning direction $\check{s}'$ will have direction $\check{s}$ after the dispersing event occurs; q – source inside the examined domain Ω, φ – photon density; μ_a – absorption coefficient; μ_s – disperse coefficient; r – geometrical coordinates of the examined point; c – the speed of light in the medium; t – time; ω – frequency.

The above RTE equation is a precise numerical model for light transport phenomenon including NIR. However the numerical model based on RTE is difficult to solve, because of a long time required to obtain results using nowadays hardware. Monte Carlo is one of the methods which can be used to solve RTE.

However, the Diffusion Optical Tomography operates on testing objects consisting of tissues, which characterize the following relation:

$$\mu_s' \gg \mu_a \,. \tag{6.4}$$

Tissues absorption coefficient is much smaller then the dispersing coefficient. Thanks to this fact RTE can be reduced to the diffusion equation without loss of results quality. Then the diffusion equation in the time domain can be formed as [1, p. 1535] [9, p. 1780] [10, p. 896]:

$$\left(\nabla \cdot D\nabla - \mu_a - \frac{\partial}{\partial t}\right)\varphi(r,t) = q_0(r,t)\,, \tag{6.5}$$

and in the frequency domain [11, p. 139] [10, p. 896]:

$$\left(\nabla \cdot D\nabla - \mu_a - \frac{i\omega}{c}\right)\varphi(r,\omega) = q_0(r,\omega)\,, \tag{6.6}$$

where D – the diffusion coefficient:

$$D = \frac{1}{3(\mu_a + \mu_s')}\,. \tag{6.7}$$

Further discussion will concentrate on a frequency domain because then the medical testing is shorter. The diffusion equation in the frequency domain (6.6) can be presented as the Helmholtz equation including the source q_0:

$$\nabla^2\varphi(r,\omega) - k^2\varphi(r,\omega) = -\frac{q_0(r,\omega)}{D}\,, \quad \text{where} \quad k^2 = \frac{\mu_a}{D} - i\frac{\omega}{cD}\,, \tag{6.8}$$

where: $k \in \mathbb{C}$ – is the complex wavenumber.

The collimated source NIR light is supplied to the baby head or the surface of the breast through the fibre-optic. In the numerical model it is modelled as the point source located under the surface at the distance of $\frac{1}{\mu_s'}$.

Forward problem definition requires also setting boundary conditions (BC). The third kind BC, also known as Robin BC, are used in the DOT model and they represent the following relationship:

$$\frac{\partial \varphi}{\partial n} = m_{RBC}\,\varphi + n_{RBC}\,, \tag{6.9}$$

where m_{RBC}, n_{RBC} – parameters.

The above Helmholtz Partial Differential Equation (PDE) can be solved using any applicable method and also Boundary Element Method with title BEMLAB software.

6.3 Governing equation in Electrical Impedance Tomography

Another considered type of tomography imaging technique is Electrical Impedance Tomography (EIT). EIT uses electrical properties of examined materials like electrical conductivity σ. The examined object is stimulated using voltage or current source and the layout of potential on the surface is collected using sensors. These data are used by the reconstruction algorithm which gives a layout of objects internal. EIT numerical model involves the Laplace equation [12, p. 112]:

$$\operatorname{div}(\operatorname{grad}\varphi) = 0\,. \tag{6.10}$$

The Laplace PDE can also be solved using Boundary Element Method and the title BEMLAB software.

6.4 Boundary Element Method

Diffusive Optical Tomography and Electrical Impedance Tomography problems are popularly solved using Finite Element Method (FEM) [14]. FEM is

the domain method which means that the whole object domain Ω has to be discretized. One of the FEM alternatives is the Boundary Element Method. BEM requires only the surface Γ of the examined object to be discretized, therefore its dimension is smaller by one than in domain methods. The good quality boundary mesh creation task is much simpler than creating a domain mesh. BEM is the method characterized by the square computational complexity $O(N^2)$ [8]. Moreover calculation of φ in any point inside the examined domain Ω is done without remeshing the domain. The number of equations in BEM is usually much less than in FEM. BEM has advantages comparing to FEM, but also there are some drawbacks. When the problem is characterized with the unsuitable geometry, which means that the number of boundary elements Γ_j is close to the number of domain elements Ω_j in FEM, then BEM calculations are slower than in FEM. This is strengthened by the fact that the left hand side matrix $\mathbf{a}$ in the set of linear equations (6.20) is dense in BEM in contrast to the sparse one in FEM.

Laplace equation (6.10) or Helmholtz equation (6.8) can be solved using BEM when they are defined as Boundary Integral Equation (BIE). BIE has the following analytical form:

$$c_i\varphi + \int_\Gamma \varphi \frac{\partial G}{\partial n}\, d\Gamma = \int_\Gamma \frac{\partial \varphi}{\partial n} G\, d\Gamma + \int_\Omega fG\, d\Omega\,, \tag{6.11}$$

where: Ω – the problem domain, Γ – boundary of domain Ω, φ – field function potential or photon density in DOT, G – Green function, so-called fundamental solution, $n = |\vec{n}|$ – length of the normal vector directed outwards to the boundary element, $\frac{\partial}{\partial n}$ – normal derivative, f – domain (source) function, c_i – coefficient removing or restricting the singularity from the primitive function of PDE solution.

The Green function G mentioned above varies for each type of PDE and dimension of the space in which the problem is defined. Selected Green functions for Laplace/Poisson and Helmholtz equations were gathered in table 6.1. It is worth to see that when the value of R decreases to zero ($R \searrow 0$), the singularity occurs and special integration procedure has to be taken.

If the problem is to be calculated using BEM, only the boundary Γ of the examined domain Ω has to be discretized.

In figure 6.1 there is presented an example 2D object, which boundary Γ was discretized with linear boundary elements. The boundary Γ around the domain Ω was discretized with 12 boundary elements Γ_j, where $j \in \langle 1, 12\rangle$.

Table 6.1: Green functions for Laplace / Poisson and Helmholtz PDE

Space dimension	Green functions $G(R)$ for:	
	Laplace / Poisson PDE	Helmholtz PDE
1D	$G(R) = -\frac{1}{2}R$ (6.12)	$G(R) = -\frac{1}{2k}\sinh(kR)$ (6.13)
2D	$G(R) = \frac{1}{2\pi}\ln\frac{1}{R}$ (6.14)	$G(R) = \frac{1}{2\pi}K_0(kR)$ (6.15)
3D	$G(R) = \frac{1}{4\pi R}$ (6.16)	$G(R) = \frac{1}{4\pi R}e^{-kR}$ (6.17)

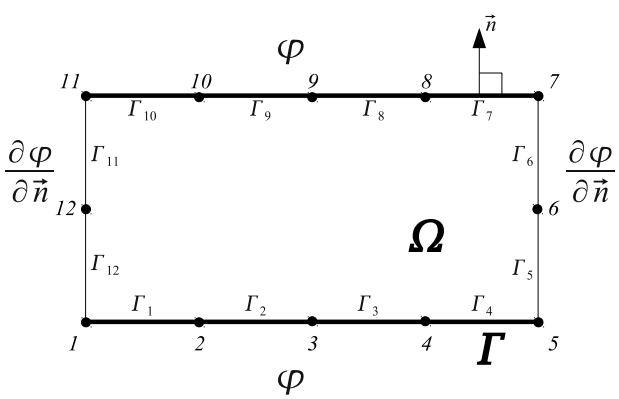


Figure 6.1: Example 2D boundary mesh with the marked boundary conditions

There are also 12 boundary nodes, where $i \in \langle 1, 12\rangle$. Boundary conditions were also marked in the model. Dirichlet Boundary Conditions (DBC) were applied on the top and bottom border (known potential φ), whereas Neumann Boundary Conditions were applied on the left and right border of the model (known potential normal derivative $\frac{\partial\varphi}{\partial n}$. As can be noticed the domain was not discretized. While creating the boundary mesh it is required that the normal vector to the boundary element $\vec{n}$ is directed outward the examined domain Ω.

When the domain boundary Γ is discretized the Boundary Integral Equation (6.11) has to be written in the discretized form. The boundary was discretized into J boundary elements and it has I nodes. Each element has K nodes which interpolate potential φ and L nodes which interpolate its normal derivative $\frac{\partial \varphi}{\partial n}$. When the source is defined in the domain Ω, it can be discretized into D domain elements. The discretized BIE for one domain problem has the following form:

$$\begin{aligned} c_i(\mathbf{r}_i)u_i(\mathbf{r}_i) + \sum_{j=1}^{J}\sum_{k=1}^{K} u_k^{(j)} \int\limits_{\Gamma_j(\boldsymbol{\xi})} N_k^{(u)}(\boldsymbol{\xi}) \frac{\partial G\big(\mathbf{r}(\boldsymbol{\xi}),\mathbf{r}_i\big)}{\partial n} \left|J_{\Gamma_j}^{(g)}(\xi)\right| d\Gamma_j(\boldsymbol{\xi}) = \\ = \sum_{j=1}^{J}\sum_{l=1}^{L} q_l^{(j)} \int\limits_{\Gamma_j(\boldsymbol{\xi})} N_l^{(q)}(\boldsymbol{\xi}) G\big(\mathbf{r}(\boldsymbol{\xi}),\mathbf{r}_i\big) \left|J_{\Gamma_j}^{(g)}(\boldsymbol{\xi})\right| d\Gamma_j(\boldsymbol{\xi}) + \\ + \sum_{d}^{D} \int\limits_{\Omega_d(\boldsymbol{\zeta})} f\big(\mathbf{r}(\boldsymbol{\zeta})\big) G\big(\mathbf{r}(\boldsymbol{\zeta}),\mathbf{r}_i\big) \left|J_{\Omega_d}^{(g\Omega)}(\boldsymbol{\zeta})\right| d\Omega_d(\boldsymbol{\zeta}) \,, \end{aligned} \tag{6.18}$$

where: $N_k^{(u)}$ – base interpolation functions for potential φ; $N_l^{(q)}$ – base interpolation functions for potential derivative $\frac{\partial \varphi}{\partial n}$; $\left|J_{\Gamma_j}^{(g)}(\xi)\right|$ – Jacobian of transformation of geometrical boundary element from the global coordinate system to the local coordinate system; $\left|J_{\Omega_d}^{(g\Omega)}(\xi)\right|$ – Jacobian of transformation of geometrical domain element from the global coordinate system to the local coordinate system; $\boldsymbol{\xi}$ – boundary point local coordinates of transformed boundary element; $\boldsymbol{\zeta}$ – domain point local coordinates of transformed domain element; $u_k^{(j)}$ – potential φ value in the node k of the element j; $q_l^{(j)}$ – potential normal derivative $\frac{\partial \varphi}{\partial n}$ in the node l of the element j.

Normally potential φ and its normal derivative $\frac{\partial \varphi}{\partial n}$ are interpolated by boundary elements of the same type, which results the following relation: $K = L$ – number of boundary element nodes is the same for both interpolation functions φ and $\frac{\partial \varphi}{\partial n}$.

Matrices are good containers and matrix calculations are clear, therefore BIE (6.18) can be presented in the matrix form:

$$\mathbf{A}\,\boldsymbol{\varphi} = \mathbf{B}\,\frac{\partial}{\partial n}\boldsymbol{\varphi} + \mathbf{F}\,. \tag{6.19}$$

When the Laplace PDE is calculated the vertical vector $\mathbf{F}$ responsible for domain source potential is equal zero: $\mathbf{F} = \mathbf{0}$. Introducing boundary conditions is the next step. BCs are applied to the equation (6.19) which results in the following set of linear equations:

$$\mathbf{a}\,\mathbf{x} = \mathbf{b}\,, \tag{6.20}$$

where: $\mathbf{x}$ – is the unknown vector composed from unknown boundary potentials $u = \varphi$ and/or potential normal derivatives $q = \frac{\partial \varphi}{\partial n}$: $\mathbf{x} = \begin{bmatrix} \boldsymbol{\varphi} \\ \frac{\partial \boldsymbol{\varphi}}{\partial n} \end{bmatrix}$. It is worth to notice that the left hand side matrix $\mathbf{a}$ is dense.

The set of equations (6.20) can be calculated using solvers based on LU decomposition [6], faster GMRES [7, 18] or any other available. As the result of BEM calculations all potential $u = \varphi$ and its normal derivative $q = \frac{\partial \varphi}{\partial n}$ are known in the boundary nodes. Internal node values do not have to be calculated but if needed their values are calculated using potential and its normal derivative values from all boundary nodes.

6.4.1 Multi domain problems

The above discretized form of BIE (6.18) is applicable for one domain problem. But BEM can also solve multi domain problems. Then BIE should be modified so regions are included.

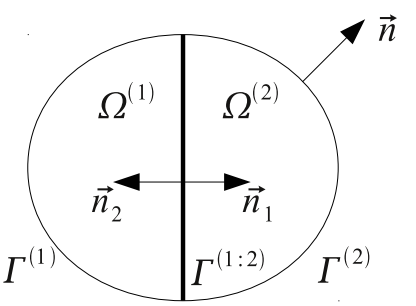


Figure 6.2: Two domain ($\Omega^{(1)}$ and $\Omega^{(2)}$) problem with a marked interface $\Gamma^{(1:2)}$ between them

Two domain problem is presented in figure 6.2. There is a boundary $\Gamma^{(1:2)}$ named an interface between domains (regions) $\Omega^{(1)}$ and $\Omega^{(2)}$. There are also marked external boundaries for particular domains: $\Gamma^{(1)}$ for $\Omega^{(1)}$ and $\Gamma^{(2)}$ for $\Omega^{(1)}$. This particular notation of boundaries and consequently BIE

was created specially for the BEMLAB library. Normal vectors $\vec{n}$ to the boundaries were also marked. When the domain $\Omega^{(1)}$ is considered the vector $\vec{n}_1$ is taken, when the domain $\Omega^{(2)}$ is considered the vector $\vec{n}_2$ is taken.

The Boundary Integral Equation for the multi domain problem made of $\mathcal{R}$ domains created for BEMLAB numerical package has the following form:

$$\sum_{r=1}^{\mathcal{R}}\sum_{b=1}^{\mathcal{B}_r}\big(\,c_i(\mathbf{r}_i)u_i(\mathbf{r}_i)\,\big)+$$

$$+\sum_{r=1}^{\mathcal{R}}\sum_{b=1}^{\mathcal{B}_r}\left(\sum_{j=1}^{J_{b_r}}\sum_{k=1}^{K}u_k^{(j)}\int\limits_{\Gamma_j(\boldsymbol{\xi})}N_k^{(u)}(\boldsymbol{\xi})\frac{\partial G\big(\mathbf{r}(\boldsymbol{\xi}),\mathbf{r}_i\big)}{\partial n}\left|J_{\Gamma_j}^{(g)}(\xi)\right|d\Gamma_j(\boldsymbol{\xi})\right)=$$

$$=\sum_{r=1}^{\mathcal{R}}\sum_{b=1}^{\mathcal{B}_r}\left(\sum_{j=1}^{J_{b_r}}\sum_{l=1}^{L}q_l^{(j)}\int\limits_{\Gamma_j(\boldsymbol{\xi})}N_l^{(q)}(\boldsymbol{\xi})G\big(\mathbf{r}(\boldsymbol{\xi}),\mathbf{r}_i\big)\left|J_{\Gamma_j}^{(g)}(\boldsymbol{\xi})\right|d\Gamma_j(\boldsymbol{\xi})+\right.$$

$$\left.+\sum_{d=1}^{D_r}\int\limits_{\Omega_d^{(r)}(\boldsymbol{\zeta})}f_r\big(\mathbf{r}(\boldsymbol{\zeta})\big)G_r\big(\mathbf{r}(\boldsymbol{\zeta}),\mathbf{r}_{i_r}\big)\left|J_{\Omega_d}^{(g\Omega)}(\boldsymbol{\zeta})\right|d\Omega_d^{(r)}(\boldsymbol{\zeta})\right)\,, \tag{6.21}$$

where: $\mathcal{B}_r$ – is the number of boundaries neighbouring the current region (domain) r; b_r – is the current boundary between the current region r and the neighbour (or external region marked as Ω', where $\Omega' \nsubseteq \Omega$).

The matrix BIE equation (6.19) and the set of equations (6.20) has the same form for multi domain problems. However before applying boundary conditions, interface conditions have to be additionally applied to the equation (6.21). There are two interface continuity conditions on the interface:

Potential continuity – In the interface node the following relationship occurs:

$$\varphi_1|_{\Gamma^{(1:2)}} = \varphi_2|_{\Gamma^{(1:2)}}\,. \tag{6.22}$$

Potential normal derivative continuity – In the interface node the following relationship occurs:

$$m_1\left.\frac{\partial\varphi_1}{\partial n}\right|_{\Gamma^{(1:2)}} = -m_2\left.\frac{\partial\varphi_2}{\partial n}\right|_{\Gamma^{(1:2)}}\,, \tag{6.23}$$

where: m_1, m_2 – are material parameters of particular domains. The $-$ (minus) sign exists because normal vectors has the same direction but opposite turns as is marked in Fig. 6.2.

After solving the set of equation (6.20) for multi domain problems, all potential $u = \varphi$ and its normal derivative $q = \frac{\partial \varphi}{\partial n}$ values in nodes of external boundaries $\Gamma^{(r)}$ and interfaces $\Gamma^{(r_1:r_2)}$ are known.

This section includes some basic information about Boundary Element Method, which were required to model and implement BEMLAB library. There were presented notation and modelling procedure which make possible to create the universal BEM software applicable to diverse problems.

6.5 BEMLAB software

The name Boundary Element Method was proposed by C. A. Brebbia in 1970s [2]. The number of applications increased since then and further areas are being investigated. At the same time development of FEM was much more rapid and nowadays its role is indisputable. When the computers become more common and high level programming languages arise, lots of applications implementing FEM were created. There is a broad choice of open source as well as commercial FEM software. Everyone can choose the one which is the best for particular applications. This is also one of the reasons why the FEM is much more popular method for solving PDEs than BEM.

The BEM software availability is much smaller than the broad choice of FEM one [5, 15]. And unfortunately, only little BEM codes were created for all the time since the BEM arose. One of the reasons may be the fact that BEM mathematical description is more complicated. Another one probable cause is the existence of problematic singularities which have to be taken into consideration and solved. Because of lack of BEM software applicable to tomography applications the BEMLAB [17] project was initiated.

6.5.1 Technology

Licensing

It was decided that the BEMLAB software will be developed using the open source licence. Some of the reasons why the code is publicly available are: the intention of creating the community around the project, Boundary Element Method popularization among engineers and scientists, its further development and acceleration.

BEMLAB binary packages and the source code are distributed under GNU LGPL (Lesser General Public License) license terms. The project provides universal library and the reference application, which is the easiest way for solving problems using BEM. There are also auxiliary programs provided to facilitate engineer's tasks.

Technology main goals

The licensing and technology were chosen in the way so the BEMLAB project could be named as a "good open source project", which comply criteria described in e.g. [3]. The following aims were set against the project:

- calculation correctness,
- usage easiness,
- further development easiness, by choosing well known technologies.

The numerical software is a special type of software which target group is relatively small comparing to the system or application software. Therefore the chosen technology, tools, modelling and development procedures should be already known to the potential users, so they can be easily engaged to use the created software. It is particularly important in case of BEM software, because the method by itself is not widely known.

Objectivity

BEMLAB software has an objective architecture because it simplifies the process of modelling, development and further hosting. This also decreases the entrance level for new developers so they can faster and easier get to know the project architecture. Moreover there is a Unified Modelling Language (UML) [4] available, which allows creation of standardized diagrams made of unified symbols. UML allows creation software requirements and architecture during the whole process of modelling and development. UML diagrams are unambiguous therefore all projects participants has a clear view of how the code will be implemented or is already implemented. Furthermore "pictures" like diagrams are generally easier assimilated than the code by itself. The objectivity allows modelling and creating the code which is better adapted to the reality, than the functional one.

Programming language

Nowadays, many programming languages support object-oriented programming. C++ was chosen as the main programming language used in BEMLAB software. Other considered were Java and C#, but both require virtual machines and their programmes are slower than those written in C++. It is important that those three programming languages are popular and are taught in all computer engineering studies of the undergraduate and graduate courses. Nowadays computer companies use them to create software. Some available BEM codes [15, **?**] are written in Fortran which was a popular language of numerical software. Nowadays it is not popular, commercial software does not use it and finally it is taught in a scarce scope if not at all. Moreover functional C and Objective C++ are popular in the open source community.

Compiler

C++ compilers are available on almost any platform and operating system. Almost all C/C++ software in Linux/Unix like operating systems use GNU compilers. BEMLAB reference implementation also uses GNU C++ Compiler which is the base compiler among open source software. There are multithreaded algorithms specially designed and implemented for BEMLAB,

which fasten BEM calculations on multicore processors and on multiprocessor computers. BEMLAB library uses threads introduced in C++0x specification. Therefore it is recommended to use GNU Compiler version 4.4.0 or newer, because C++0x *threads* are available from that version only. However up to now it is possible to use previous versions of GNU compilers (tested all major versions since GNU Compiler 3.3), but with the restriction that multithreaded algorithms are turned off. This is deprecated but makes possible to use BEMLAB on older operating systems where newer versions of GNU Compiler are not yet available. However the code without multithreading will be completely removed in the future.

Code manager

The most important case for the end user is possibility of using the software. The user wants to run the software in the known and the easiest way possible. Similarly the programmer wants to compile and build the software efficiently. Code managers come with help to fulfil these requirements. Code managers allow to automate such tasks like environment configuration, code compilation, binaries building, installation or source package creation. All tasks are very important but the environment configuration is worth noting, because thanks to it the developer does not have to bother how the end user environment and the operating system are configured. Differences between platforms and systems distributions are transparent and the code manager manages with them automatically. Moreover it provides the possibility to provide user's special configuration options in the unified way e.g. install the application in a non standard location. Two code managers were taken into account:

- the suite of tools: `automake`, `autoconf` and `libtool`, also known as `autotools`
- `cmake`

Finally the first one was used despite of its disadvantages like e.g. difficulty to programme. However it is much more common, users are used to it and simply is fair enough. The second one (`cmake`) is not so popular and maybe it will not become. The code manager is required to develop the code effectively.

Version control system

Almost none programming project can obey without Version Control System (VCS). And not only programming one e.g. this chapter creation (writing) was carried out with the help of version control system. VCS is designed to store all versions of particular files. It allows to compare changes, create branches, merge codes versions. It has the full history of changes made in the code or document. It allows a group of people to develop the code effectively. Without the VCS it would be impossible. When the BEMLAB project was started out two version control systems engines were considered:

- CVS – popular and widely used,
- SVN – newer, somewhat less popular than CVS, but with substantial advantages and much modern.

Nowadays GIT is obtaining consecutive applications because of its modern architecture. However it was not mature enough that time and is not yet as popular as both mentioned above together – support on various operating systems is still restricted. Finally the SVN was chosen, which has been used for the whole processed development. Publicly the repository is available under the address:

```
svn://svn.bemlab.org/bemlab
```

Until now SVN is sufficient but the repository migration to GIT cannot be excluded in the future.

Website

Many open source projects have websites and every "good open source project" must have one. The website is easily available and is the most popular place for distributing software and documentation. The community is gathering around the projects website and available services. This is a very important part of the project. BEMLAB project has its own domain:

```
http://bemlab.org/
```

There are many web applications available but only actively developed and with a good support (bug fixes deployment) were considered to be used with a BEMLAB project. Mainly Content Management Systems (CMS) such as Joomla or Drupal were considered, and Wiki applications such as MoinMoin and MediaWiki. Finally MediaWiki [20] was chosen – it is broadly used (among others by Wikipedia) and bug fixes are systematically made available. Moreover a lot of people know MediaWiki interpreter and it supports LaTeX equations.

The website and the software also require the logo so they can be easily recognized. Figure 6.3 presents the logo specially designed for the BEMLAB project – it presents a discretized surface of sphere, the boundary.

6.5.2 Data Input/Output format

While creating specification for the project it was decided that the calculated problems will be defined fully using one file format. It was also decided that the problem can be split among many files. Exchanging data with external files was a priority.

File format

Taking above into account the text file format compatible with Matlab M-files was chosen. The example matrix definition named `matrix` consisting of 2 rows and 3 columns with a marked comment is as follows:

```
matrix=[1 3 7; 4 6 8]; % comment (2,3)
```

The choosen file format can easily be converted into Scilab sci-format, by changing the comment string.

```
shell% sed -e 's/%/\/\//g' file_matlab.m >
file_scilab.sci
```

The converted file can then be read into scilab:

```
scilab-> exec('file_scilab.sci')
```

Furthermore a text file format is more user friendly. It simplifies tasks related with mesh creation. Text files can easily be edited and manipulated using standard tools such as text editors (e.g. `vi`) or line editors (e.g. `sed`, `awk`). It is also easer to write programmes which process data files in script languages like `perl`, `python`, `ruby` or using the shell interpreter.

Data format

The BEMLAB data format is based on the file format presented above. The base unit is the matrix. Mesh files are composed of many matrices with the strictly defined names. The huge advantage of the file format is its universality. Any number of domains, elements, sources, etc. can be defined for the calculated problem. The matrix definitions can be specified in files in any order. A basic example is presented in figure 6.4.

Figure 6.3: BEMLAB logo

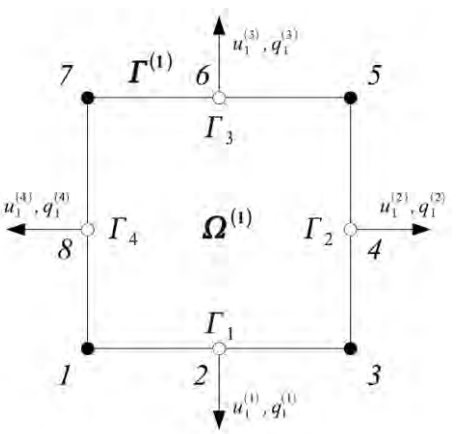


Figure 6.4: 2D boundary mesh

This is a two dimensional problem discretized with $J = 4$ boundary elements. Potential u_k^j and its normal derivative q_l^j were marked in the $I = 4$ interpolation nodes. One node, constant elements are used to interpolate potential and its normal derivative $K = L = 1$ – see equation (6.18). Geometry is discretized with 2 node linear boundary elements.

The mesh from figure 6.4 can be defined in BEMLAB data format in the following form:

```
nodes=[0,0; 5,0; 10,0; 10,5; 10,10; 5,10; 0,10; 0,5]; %

elementsGeom=[1,3; 3,5; 5,7; 7,1]; % 4
elementsU=[2; 4; 6; 8]; % 4
elementsQ=elementsU;

elementTypeGeom=['LineLinear'];
elementTypeU=['LineConst'];
elementTypeQ=elementTypeU;

dirichletElements=[ 1, 3 ]; % 2

dirichletBC=[0; 10]; % 2

neumannElements=[ 2, 4 ]; % 2

neumannBC=[0; 0]; % 2

angleCoefficients=[ 2, 4, 6, 8 ]; % 4
angleCoefficientValues=[ 0.5, 0.5, 0.5, 0.5 ]; % 4

methodType_1=['poisson'];
```

where:

- `nodes` – is the matrix which includes all nodes coordinates defined in the problem,
- `elementsGeom` – is the matrix which includes geometrical boundary elements definitions – contains indexes to nodes defined in `nodes` matrix,
- `elementsU` – is the matrix which includes definitions of elements interpolating potential $u = \varphi$ – contains indexes to nodes defined in `nodes` matrix,
- `elementsQ` – is the matrix which includes definitions of elements interpolating potential normal derivative $q = \frac{\partial \varphi}{\partial n}$ – contains indexes to nodes defined in `nodes` matrix, and usually is equal to `elementsU` matrix,

- `elementTypeGeom` – is the matrix which defines geometrical element type – here first order linear element is defined: `['LineLinear']`,
- `elementTypeU` – is the matrix which defines type of element interpolating potential $u = \varphi$ – here first order linear element is defined: `['LineConst']`,
- `elementTypeQ` – is the matrix which defines type of element interpolating potential normal derivative $q = \frac{\partial \varphi}{\partial n}$ – usually is equal to `elementTypeU` matrix,
- `dirichletElements` – is the matrix which defines elements which has Dirichlet Boundary Condition defined – here two elements 1 and 3,
- `dirichletBC` – is the matrix which defines Dirichlet Boundary Condition values (potential u) in consecutive elements defined in matrix named `dirichletElements` – here element 1 has the potential $u_1^{(1)} = 0$ defined in node 2 and element 3 has the potential $u_1^{(3)} = 10$ in node 6,
- `neumannElements` – is the matrix which defines elements which has Neumann Boundary Condition defined – here two elements 2 and 4,
- `neumannBC` – is the matrix which defines Neumann Boundary Condition values (potential normal derivative q) in consecutive elements defined in matrix named `neumannElements` – here elements 2 has the value $q_1^{(2)} = 0$ defined in node 4 and element 4 has the value $q_1^{(4)} = 0$ in node 8,
- `angleCoefficients` – is the matrix which contains indices of nodes which interpolate potential u and its normal derivative q and will have c_i coefficient value manually defined – see equation (6.18) – this matrix is auxiliary for this model,
- `angleCoefficientValues` – is the matrix which contains values of c_i coefficient for nodes defined in matrix `angleCoefficients` – here all c_i values are equal 0.5, where $i = 1, 2, 3, 4$. $c_i = 0.5$ is the default value therefore `angleCoefficients` and `angleCoefficientValues` matrices don't have to be defined in this case,
- `methodType_1` – is the matrix which contains the name of an integral kernel which is used to set up BEM matrices: $\mathbf{A}, \mathbf{B}, \mathbf{F}$.

The above problem defined in the presented file `example_2d.m` can be solved using BEMLAB software by issuing the following command:

```
% obem_solve -i example_2d.m -m 1234 -o
solution_output.m
```

The results are written to the output file `solution_output.m`.

The BEMLAB package also provides programs which can be used for post-processing tasks like drawing a plot.

6.5.3 BEMLAB architecture

This section presents only selected information about the BEMLAB software architecture.

Before creating the BEMLAB library architecture the mathematical description of the numerical method has been done (section 6.4). Among others use case models has been created. One of the diagrams which show the whole process of any problem modelling is the activity diagram presented in figure 6.5.

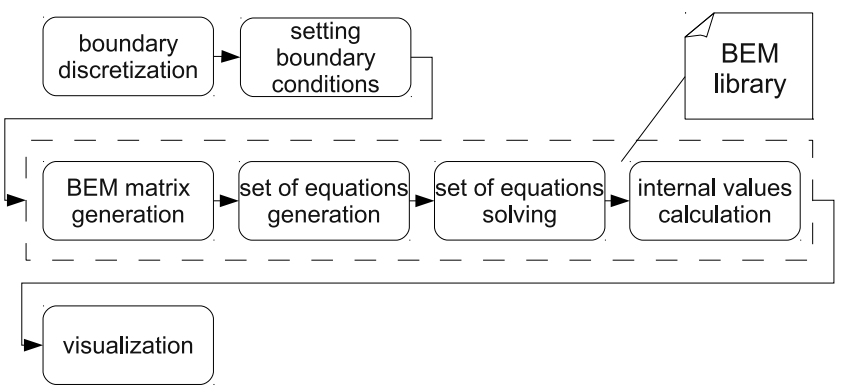


Figure 6.5: Activity diagram containing main activities taken while solving problems defined by Partial Differential Equations, which include preprocessing (first row), chosen numerical method calculations (second row), and finally postprocessing (third row)

The diagram contains main activities taken during solving problems defined by Partial Differential Equations. The whole process is divided into three stages from the numerical software point of view. The following stages include basic BEM activities presented in the one domain model case for the simplicity, but extendible in the multi domain one:

1. Preprocessing – Includes mainly:
   - *boundary discretization* Γ of the problem domain Ω,
   - *setting boundary conditions* on the external boundary of the problem.
2. Numerical method calculations – Includes activities involving Boundary Element Method tasks, which mainly include:
   - *BEM matrix generation* $\mathbf{A}, \mathbf{B}, \mathbf{F}$ – equation (6.19),
   - *set of equations generation* – the left hand side $\mathbf{a}$ and the right hand side $\mathbf{b}$ matrices are generated using among others generated previously $\mathbf{A}, \mathbf{B}, \mathbf{F}$ matrices and boundary conditions – equation (6.20),
   - *set of equations solving* – this activity is the most time consuming and includes running the solver – as a result all unknown boundary values of potential u and its normal derivative q are known,
   - *internal values calculations* – this activity is optional and run only when needed, it uses the BEM engine and needs the data required for previous tasks and calculated boundary values of potential u and its normal derivative q.
3. Postprocessing – Includes tasks involving usage of results obtained with numerical method in the previous stage like:
   - *visualization.*

The basic activities of the BEM core included in the BEMLAB library are marked with a dashed line.

Another diagram which was created during modelling the BEMLAB software is the component diagram presented in figure 6.6.

The software was divided into: *lib* – the library, *application* – reference application and *tests* – testing module.

The main logic is included in the library *lib*. As the BEMLAB software implements BEM comprehensively providing the multi physics package, the *lib* was divided into several components:

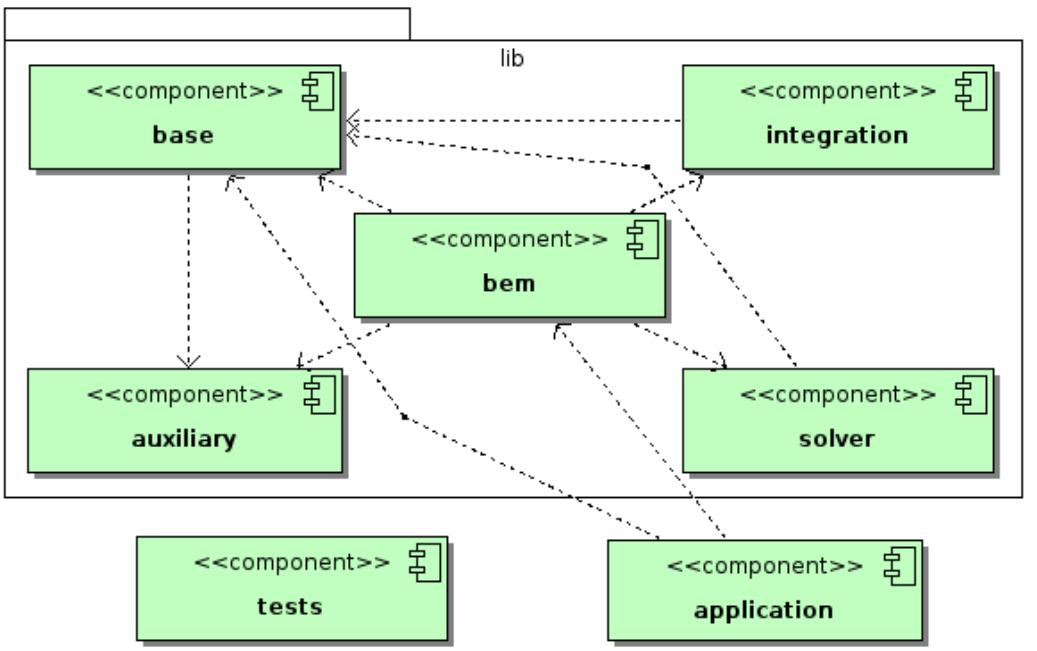


Figure 6.6: BEMLAB component diagram

- *base* – Includes containers used accross the library, methods implementing required algorithms like matrix calculations or iterators, implements methods responsible for the data file format compatible with Matlab M-files,
- *bem* – Includes algorithms wich implement boundary element method. There are main algorithms implementing activities from the diagram 6.5, the equation (6.21), Greens functions, integration kernels calculations, boundary and interface conditions application, internal point calculations, etc.,
- *integration* – Includes required integration algorithms e.g. Gauss Quadrature,
- *solver* – Includes algorithms used for solving sets of linear equations (6.20): $\mathbf{a}\,\mathbf{x} = \mathbf{b}$,
- *auxiliary* – Includes auxiliary algorithms not available in Standard Template Library (STL) distributed with C++ but required by the BEMLAB software.

The application component consists of several classes which use the library

to provide its functionality for the end-user. Using application is the easiest way to proceede calculations by the end-user.

A very important is the test component. The development cannot be efficient without broad range of tests. It is especially important in the numerical software where a small change may have a big impact on the correctness of calculations. Tests allow to detect mistakes and bugs on the very early stage. There are several type of tests. Some of them test particular methods and classes, where the others the library and the application as a whole – acceptance testing. The whole bunch of tests is run before every commit to the version control system.

6.5.4 CSparskit2

The longest task which is proceeded during BEM calculations is solving the set of linear equations as presented in diagram 6.5 and equation (6.20). Moreover the left hand side matrix **a** is dense in opposite to FEM where it is sparse and symmetric. Therefore fast algorithms known from FEM cannot be used. However calculations can be fastened using GMRES algorithms. The Generalized Minimal Residual Method (GMRES) was proposed by Yousef Saad [7] in 1980s.

One of the GMRES implementations is the SPARSKIT2 package [19] by Yousef Saad. The source code is written in Fortran. It was essential to write a wrapper in C++ to make it use the BEMLAB containers implementing matrix format compatible with Matlab M-files. The C++ wraper was named *CSparskit2* and is available at [18]. CSparskit2 uses BEMLAB *base* component (rysunek 6.6) and depends on SPARSKIT2 package.

6.6 The Diffuse Optical Tomography problem described by means of the baby head model

Diagnosing and controling head haemorhages in newborn infants especially premature babies, and woman breast tumour detection are the main areas of scientists intersest for Diffuse Optical Tomography application (DOT). DOT

uses near infrared light which wavelength λ is usually from 760 to 830mm.

A three dimensional baby head model is presented in this section. Figure 6.7 presents the model of baby head.

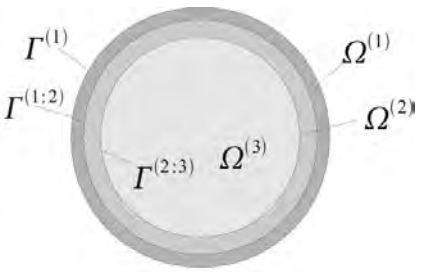


Figure 6.7: Schematic three layers model of baby head

The head is divided into three domains: $\Omega^{(1)}$ – scalp, $\Omega^{(2)}$ – skull and $\Omega^{(3)}$ – brain. Each domain $\Omega^{(r)}$, where $r \in \langle 1, 2, 3 \rangle$ differs in tissue optical parameters. Boundary Element Method requires only boundaries to be discretized, therefore only meshes for $\Gamma^{(1)}$, $\Gamma^{(2)}$ and $\Gamma^{(3)}$ boundaries have to be generated. The boundaries were discretized using six node quadratic triangle.

The NIR source characterized by the frequency modulation of source light intensity $\omega = 100$MHz is used in this example. The NIR light source is modelled as the collimated point source placed $\frac{1}{\mu_s'}$ under the model surface. This is the most accurate mapping of light source [9] as the light dispersion starts only when the light ray passes the $\frac{1}{\mu_s'}$ length. The point source is located inside $\Omega^{(1)}$ domain in the presented model.

All calculations were done using BEMLAB software. The visualization was done using BEMLAB programmes. The following figures present direct results of the modelled forward problem. Figure 6.8a) and b) presents the layout of photon density on the head surface $\Gamma^{(1)}$, c) and d) on the skull surface $\Gamma^{(1:2)}$ and e) and f) on the brain surface for the amplitude and the phase shift respectively.

All figures consist of two subfigures were the first one presents amplitude of photon density and the second one its phase – logarithmic scale was used. The yellow marker shows the placement of the source point. It is impossible to calculate such model analytically, because of a non regular geometry. However it can be stated that the obtained results are correct based on the correct results obtained for the example geometries where analytical solu-

tion is known and calculated using the same process. Moreover the obtained photon density changes on the boundaries in the expected way – amplitude decreases when the distance from the source increases and the phase value increases when the distance from the source increases.

6.7 Summary

The BEMLAB software is designed to solve Diffuse Optical Tomography and Electrical Impedance Tomography problems. Among others it can also be used for solving electromagnetic problems. BEMLAB is protected by the open source license, which means that can be freely distributed (binaries as well as the source code). It has an objective architecture which eases modelling and development. It uses multi-threaded BEM algorithms which accelerate calculations on multicore or multiprocessor computers.

The BEMLAB software was designed to be an universal BEM package which implements various types of boundary elements, Partial Differential Equations. It can be used to calculate multi domain problems of any geometry and with any number of domains. It provides an easy to use data format compatible with Matlab M-files. There are also some auxiliary programmes for preprocessing and postprocessing provided.

The BEMLAB project aspires to be the platform for Boundary Element Method improvement. The projects created infrastructure allows to run the dispersed development. Now, BEMALB is a ready to use software for solving problems described by PDEs including tomography problems as it was presented.

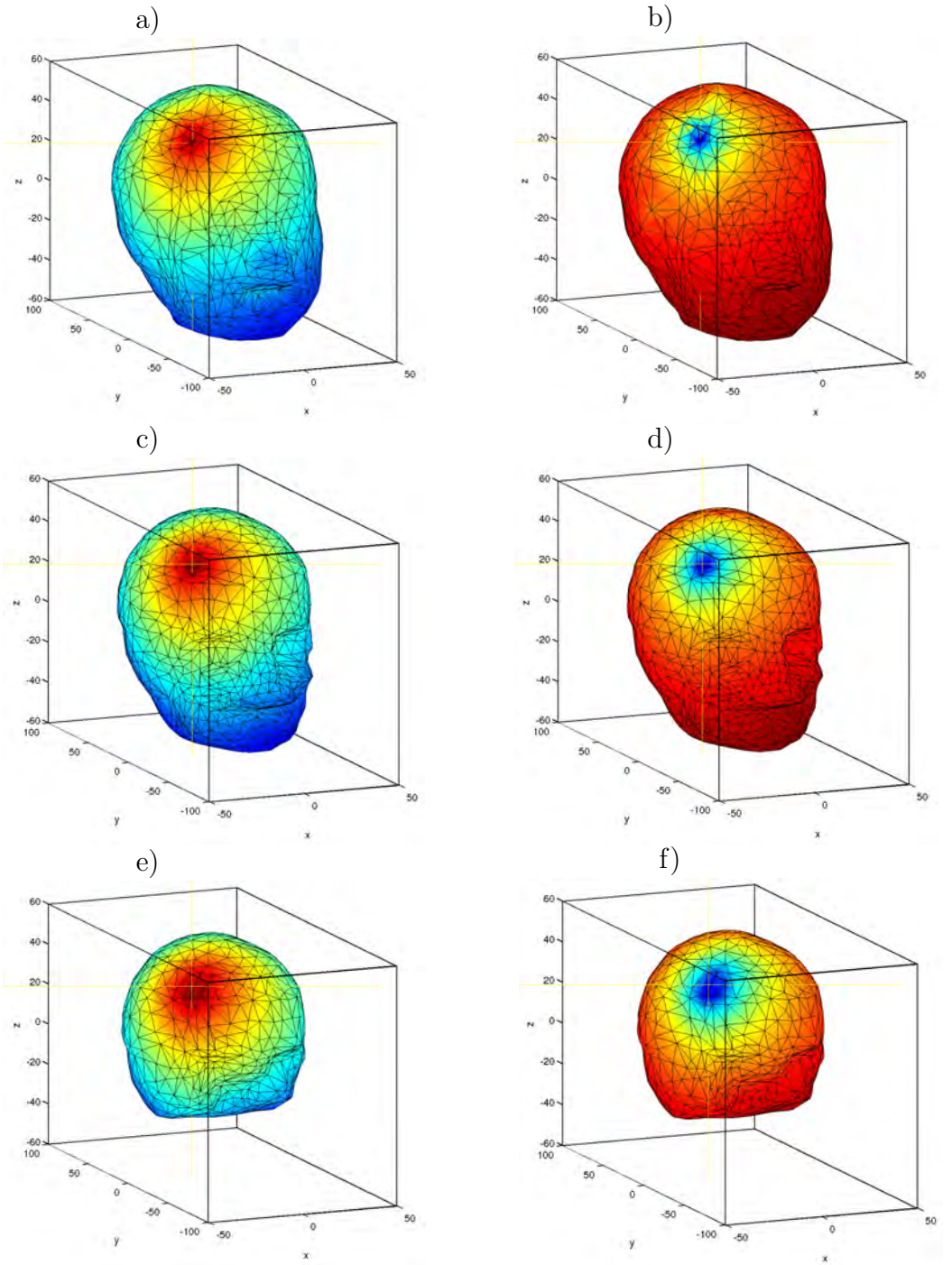


Figure 6.8: Photon density layout presented in logarithmic scale on: a) the head surface $\left(\Gamma^{(1)}\right)$, c) the skull surface $\left(\Gamma^{(1:2)}\right)$, e) the brain surface $\left(\Gamma^{(2:3)}\right)$; the left column presents the amplitude and the right column the phase shift

Bibliography

[1] S.R. Arridge, M. Cope, D.T. Delpy: The theoretical basis for the determination of optical pathlengths in tissue: temporal and frequency analysis, Physics of Medical Biology, Vol. 37, No. 7, 1992, pp. 1531-1560.

[2] C.A. Brebbia: The boundary element method for engineers, Wiley, 1978.

[3] K. Fogel: Producing Open Source Software: How to Run a Successful Free Software Project, Publisher: O'Reilly, 2005.

[4] M. Fowler: UML Distilled: A brief guide to the standard object modelling language, Addison-Wesley, ed. 3, 2003.

[5] J. Mackerle: Object-oriented programming in FEM and BEM: a bibliography (1990-2003), Advances in Engineering Software, Vol. 35, 2004, pp. 325-336.

[6] W.H. Press, P.B. Flannery, S.A. Teukolsky, W.T. Vetterling: Numerical Recipes in FORTRAN: The Art of Scientific Computing: LU Decomposition and Its Applications, Cambridge University Press, 1992, pp. 34-42.

[7] Y. Saad, M.H. Schultz: GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM Journal on Scientific and Statistical Computing, Vol. 7, No. 3, 1986, pp. 856-869.

[8] M. Schanz, O. Steinbach: Boundary element analysis: Mathematical aspects and applications, Springer-Verlag, Berlin, 2007.

[9] M. Schweiger, S.R. Arridge, M. Hiraoka, D.T. Delpy: Finite element method for the propagation of light in scattering media: boundary and source conditions, Medical Physics, Vol. 22, No. 11, 1995, pp. 1779-1792.

[10] M. Schweiger, S.R. Arridge: Finite element method for the propagation of light in scattering media: frequency domain case, Medical Physics, Vol. 24, No. 6, 1997, pp. 895-902.

[11] J. Sikora: Boundary Element Method for Impedance and Optical Tomography, Oficyna Wydawnicza Politechniki Warszawskiej, 2007.

[12] J. Starzyński: Laboratorium podstaw elektromagnetyzmu, Oficyna Wydwanicza Politechniki Warszawskiej, Warszawa, 2005, (in Polish).

[13] T. Tarvainen and M. Vauhkonen and V. Kolehmainen and J.P. Kaipio and S.R. Arridge: Utilizing the radiative transfer equation in optical tomography, PIERS Online, http://piers.mit.edu, Vol. 4, No. 6, 2008, pp. 655-660.

[14] O.C. Zienkiewicz, R.L. Taylor: The finite element method, Butterworth-Heinemann, ed. 6, 2000.

[15] P. Wieleba, J. Sikora: Open Source BEM Library, Advances in Engineering Software, 2009, issn=0965-9978, Vol. 40, No. 8, 2008, pp. 564-569.

[16] P. Wieleba, J. Sikora: Open Source Boundary Element Method Library for Diffusion Optical Tomography, Electrotechnical Review, Vol. II/2007, 2007.

[17] BEMLAB homepage, http://bemlab.org/

[18] CSparskit2 homepage, http://bemlab.org/csparskit2/

[19] Sparskit2 homepage, http://www-users.cs.umn.edu/saad/software/SPARSKIT/sparskit.html

[20] MediaWiki homepage, http://mediawiki.org/

Part III

Optimization

Chapter 7

Review of modern optimization methods

V. Harbarchuk, P. Kisala

7.1 Introduction

What is optimization? What is optimal decision? It is principal problem of this review. In the simplest case, this means solving problems in which one seeks to minimize or maximize a real function by systematically choosing the values of real or integer variables from within an allowed set [1]. But it is correct only for classical mathematics and non correct for problems of very much directions of modern mathematics, at the same time – for modern engineering problems as problems of applied mathematic. – What is optimal decision in the game theory? – What is optimal decision in economic plans? – What is optimal construction of car, computer, house, ship, aircraft, criptosystem and so on? – About this problems of optimization is thousands scientific published works. For example, some specific applied and theoretical problems of modern optimization is published [2-37].Very topical problems of multiobjective and combinatorial optimization published in [38-98]. The problems multicriterial optimization, genetic algorithms and so on – in [99-152]. Any theoretical problems and problems optimal decisions at

base Pareto-principles published in [153-196]. Some theoretical methods of optimization – in [197-225] and any applied problems and methods of optimal decisions is demonstrate in [225-236].

This formulation, using a scalar, real-valued objective function, is probably the simplest example; the generalization of optimization theory and techniques to other formulations comprises a large area of applied mathematics. More generally, it means finding 'best available' values of some objective function given a defined domain, including a variety of different types of objective functions and different types of domains [1-37].

Other important mathematicians in the optimization field include: Richard Bellman, Ronald A. Howard, Leonid Kantorovich, William Karush, Leonid Khachiyan, Bernard Koopman, Harold Kuhn, Joseph Louis Lagrange, László Lovász, Arkadii Nemirovskii, Yurii Nesterov, John von Neumann, Boris Polyak, Lev Pontryagin, James Renegar, Kees Roos, Naum Z. Shor, Michael J. Todd, Albert Tucker.

7.2 What is an optimum?

We have already said that global optimization is about finding the best possible solutions for given problems. Thus, it cannot be a bad idea to start out by discussing what it is that makes a solution optimal. In the above we introduced functions of the form $f(a;Y)$ which measure the fit of a model instance with n parameters a to some set of data Y. We are interested in the optimal choice of parameters, those which give the best fit to the data. This involves finding the optimum (maximum or minimum) of the function $f(a;Y)$ with respect to a. For notational simplicity we will use $f(a) = f(a;Y)$. Since any maximum of $f(a)$ is a minimum of $f(a)$ we will only consider minimization. Formally a is a minimum point of $f(a)$ if there exists a region about a of radius ϵ such that:

$$f(a+x) > f(a) \quad \forall \quad |x| < \epsilon\,. \tag{7.1}$$

The maxima and minima of a function can either be global (the highest or lowest value over the whole region of interest) or local (the highest or lowest value over some small neighborhood). We are usually most interested in finding the global optimum (such as the model parameters which give

the best match to some image data), but this can be very difficult. Often a problem will have many local optima (perhaps caused by image noise or clutter) which means that locating the single global optima can be tricky. The most suitable methods to locate minima depend upon the nature of the function we are dealing with. There are two broad classes of algorithms. If a good estimate of the position of the minimum exists we need only use a local minimizer to improve it and find the optimum choice of parameters. If no such estimate exists some global method must be used. The simplest would be to generate a set of possible start points, locally optimize each and choose the best. However, this may not be the most efficient approach. Often an application will require both local and global methods. For instance, in a tracking problem initializing a model on the first frame may require a global search, but subsequent frames would only require a local search about the current best estimate. The choice of which local minimization technique to use will depend upon:

- whether a is one or many-dimensional,
- $f(a)$ can be differentiated efficiently,
- how noisy $f(a)$ is.

In the following we will give an overview of some of the methods for locating both global and local minima. For a more comprehensive survey, including algorithmic details.

7.3 Single objective functions

In the case of optimizing a single criterion f, an optimum is either its maximum or minimum, depending on what we are looking for. If we own a manufacturing plant and have to assign incoming orders to machines, we will do this in a way that minimizes the time needed to complete them. On the other hand, we will arrange the purchase of raw material, the employment of staff, and the placing of commercials in a way that maximizes our profit. In global optimization, it is a convention that optimization problems are most often defined as minimizations and if a criterion f is subject to maximization, we simply minimize its negation $(-f)$. A global optimum is

an optimum of the whole domain X while a local optimum is an optimum of only a subset of X.

Definition 1. (**Global Optimum**). A global optimum $x^* \in X$ of one (objective) function $f : X \to R$ is either a global maximum or a global minimum [8].

Global optimizers are useful when the search space is likely to have many minima, making it hard to locate the true global minimum. In low dimensional or constrained problems it may be enough to apply a local optimizer starting at a set of possible start points, generated either randomly or systematically (for instance at grid locations), and choose the best result. However this approach is less likely to locate the true optimum as the ratio of volume of the search region to number of starting points increases.

Correctly applied, the Simulated Annealing and Genetic Algorithm approaches described below can explore the search space better than a grid search for a given number of function evaluations, and are more likely to find the true global minimum. Note that both these approaches involve a stochastic element and so may fail to find the true minimum. In addition, since they are better at global searching than local optimization, it is usually worthwhile to 'polish' any final solution using one of the local optimizers above. Such problems regard for example design optimization, data mining, or creating long-term schedules for transportation crews. These optimization processes will usually be carried out only once in a long time. Before doing anything else, one must be sure about to which of these two classes the problem to be solved belongs.

Even a one-dimensional function $f : X = R \to R$ may have more than one global maximum, multiple global minima, or even both in its domain X. Take the cosine function for example: It has global maxima x_i at $x_i = 2i\pi$ and global minima x_i at $x_i = (2i + 1)\pi$ for all $i \in Z$.

The correct solution of such an optimization problem would then be a set $X?$ of all optimal inputs in X rather than a single maximum or minimum. Furthermore, the exact meaning of optimal is problem dependent. In single-objective optimization, it either means minimum or maximum.

Definition 2. (**Optimal Set**). The optimal set X^* is the set that contains all optimal elements [8].

There are normally multiple, often even infinite many optimal solutions. Since the memory of our computers is limited, we can find only a finite (sub-)set of them. We thus distinguish between the global optimal set X^* and the set X^* of (seemingly optimal) elements which an optimizer returns. The tasks of global optimization algorithms are:

- to find solutions that are as good as possible and
- that are also widely different from each other [8].

The second goal becomes obvious if we assume that we have an objective function $f\colon R \to R$ which is optimal for all $x \in [0, 10] \Leftrightarrow x \in X^*$. This interval contains uncountable many solutions, and an optimization algorithm may yield $X_1^* = \{0, 0.1, 0.11, 0.05, 0.01\}$ or $X_2^* = \{0, 2.5, 5, 7.5, 10\}$ as result. Both sets only represent a small subset of the possible solutions. The second result (X_2^*), however, gives us a broader view on the optimal set.

Even good optimization algorithms do not necessarily find the real global optima but may only be able to approximate them. In other words, $X_3^* = \{-0.3, 5, 7.5, 11\}$ is also a possible result of the optimization process, although containing two sub-optimal elements.

We will introduce different algorithms and approaches that can be used to maintain an optimal set or to select the optimal elements from a given set during an optimization process Multiple Objective Functions Global optimization techniques are not just used for finding the maxima or minima of single functions f. In many real-world design or decision making problems, they are rather applied to sets F consisting of $n = |F|$ objective functions f_i, each representing one criterion to be optimized [8] $F = f_i : X \to Y_i : 0 < i \leq n, Y_i \in R$.

Algorithms designed to optimize such sets of objective functions are usually na med with the prefix multi-objective. Multi-objective optimization often means to compromise conflicting goals. If we go back to our factory example, we can specify the following objectives that all are subject to optimization:

- minimize the time between an incoming order and the shipment of the corresponding product,
- maximize profit,

- minimize costs for advertising, personal, raw materials etc.,
- maximize product quality,
- minimize negative impact on environment.

The last two objectives seem to contradict clearly the cost minimization. Between the personal costs and the time needed for production and the product quality there should also be some kind of (contradictive) relation. The exact mutual influences between objectives can apparently become complicated and are not always obvious.

7.3.1 Artificial Ant Example 1

Example for such a situation is the Artificial Ant problem where the goal is to find the most efficient controller for a simulated ant. The efficiency of an ant should not only be measured by the amount of food it is able to pile. For every food item, the ant needs to walk to some point. The more food it piles, the longer the distance it needs to walk. If its behavior is driven by a clever program, it may walk along a shorter route which could not be discovered by an ant with a clumsy controller. Thus, the distance it has to cover to find the food or the time it needs to do so may also be considered in the optimization process. If two control programs produce the same results and one is smaller (i. e., contains fewer instructions) than the other, the smaller one should be preferred. Like in the faktory example, the optimization goals conflict with each other.

From these both examples, we can gain another insight: To find the global optimum could mean to maximize one function $f_i \in F$ and to minimize another one $f_j \in F, (i \neq j)$.

Hence, it makes no sense to talk about a global maximum or a global minimum in terms of multi-objective optimization. We will thus retreat to the notation of the set of optima elements $x^* \in X^* \subseteq X$.

Since compromises for conflicting criteria can be defined in many ways, there exist multiple approaches to define what an optimum is. These different definitions, in turn, lead to different sets X^*.

7.3.2 Weighted Sums (Linear Aggregation)

The simplest method to define what is optimal is computing a weighted sum $g(x)$ of all the functions $f_i(x) \in F$. Each objective f_i is multiplied with a weight w_i representing its importance. Using signed weights also allows us to minimize one objective and to maximize another. We can, for instance, apply a weight $w_a = 1$ to an objective function f_a and the weight $w_b = -1$ to the criterion f_b. By minimizing $g(x)$, we then actually minimize the first and maximize the second objective function. If we instead maximize $g(x)$, the effect would be converse and f_b would be minimized and f_a would be maximized. Either way,multi-objective problems are reduced to single-objective ones by this metod [8].

7.4 Optimization methods classification

For this classification we used [1,2,3,4,5,8,21,22,23,24-28]. Optimal state is one of the most fundamental principles in our Word [8]. It begins in the microcosm where atoms in physics try to form bonds1 in order to minimize the energy of their electrons When molecules form solid bodies during the process of freezing, they try to assume energy-optimal crystal structures. These processes, of course, are not driven by any higher intention but purely result from the laws of physics [8].

The same goes for the biological principle of survival of the fittest which, together with the biological evolution [8], leads to better adaptation of the species to their environment. Here, a local optimum is a well-adapted species that dominates all other animals In its surroundings. Homo sapiens have reached this level, sharing it with ants, bacteria, flies,cockroaches, and all sorts of other creepy creatures [8].

As long as human kind exists, we strive for perfection in many areas. We want to reach a maximum degree of happiness with the least amount of effort. In our economy, profit and sales must be maximized and costs should be as low as possible. Therefore, optimization is of the oldest of sciences which even extends into daily life [8]. If something is important, general, and abstract enough, there is always a mathematical discipline dealing with it.

The first optimization technique, which is known as steepest descent, goes

back to Gauss. Historically, the first term to be introduced was **linear programming**, which was invented by L. Kantorovich (1938) and G. Dantzig (1940). The term **programming** in this context does not refer to computer programming (although computers are nowadays used extensively to solve mathematical problems). Instead, the term comes from the use of program by the United States military to refer to proposed training and logistics schedules, which were the problems that Dantzig was studying at the time. Additionally, later on, the use of the term 'programming' was apparently important for receiving government funding, as it was associated with high-technology research areas that were considered important.

Convex programming studies the case when the objective function is convex and the constraints, if any, form a convex set. This can be viewed as a particular case of nonlinear programming or as generalization of linear or convex quadratic programming.

Linear programming (LP), [33,234] – a type of convex programming, studies the case in which the objective function f is linear and the set of constraints is specified using only linear equalities and inequalities. Such a set is called a polyhedron or a polytope if it is bounded.

The Simplex algorithm is an elegant method for linear programming which, though not strictly 'global' (in the sense that it searches all parameter space), is able to crawl out of some local minima to find better minima. It requires only function evaluations, not derivatives, but unlike Powell's method it neither uses line minimizations nor builds an implicit model of the derivative structure of the function.

This makes it the method of choice for noisy functions. Though slower than Powell's, it is more robust. A simplex is the geometrical figure in n dimensions consisting of $n+1$ vertices. In 2D it is a triangle, in 3D a tetrahedron. The Simplex Algorithm for minimization takes such a set of $n+1$ points and attempts to move them into a minimum. The simplex formed from the points should be **non-degenerate**, it should have a non-zero volume. If your initial guess is a_0 the other n points of the simplex can be initialized as

$$a_i = a_0 + \lambda_i \mathbf{e}_i \,, \tag{7.2}$$

where $\mathbf{e}_i$ are unit vectors ($i = 1 \ldots n$).

The algorithm then takes a series of steps. It will either:

- *reflect* the point with the highest function evaluation in the plane defined by the remaining points,
- *reflect* and *expand* to take larger steps,
- *contract* to shrink the overall volume when it has reached a valley floor with reflections and expansions.

The overall effect is for the simplex to crawl around the parameter space, creeping down valleys and shrinking to get to the very bottom of narrow valleys. **Second order cone programming** (SOCP) is a convex program, and includes certain types of quadratic programs.

Semidefinite programming (SDP) is a subfield of convex optimization where the underlying variables are semidefinite matrices. It is generalization of linear and convex quadratic programming.

Conic programming is a general form of convex programming. LP, SOCP and SDP can all be viewed as conic programs with the appropriate type of cone.

Geometric programming is a technique whereby objective and constraints expressed as posynomials and equality constraints as monomials can be transformed into a convex program.

Integer programming [18,19,22,24,69,223,227] – studies linear programs in which some or all variables are constrained to take on integer values. This is not convex, and in general much more difficult than regular linear programming.

Quadratic programming [50] – allows the objective function to have quadratic terms, while the set A must be specified with linear equalities and inequalities. For specific forms of the quadratic term, this is a type of convex programming.

Nonlinear programming [41] – studies the general case in which the objective function or the constraints or both contain nonlinear parts. This may or may not be a convex program. In general, the convexity of the program affects the difficulty of solving more than the linearity.

Stochastic programming studies the case in which some of the constraints or parameters depend on random variables.

Figure 7.1: Golden Section Serach [2]

Robust programming is, as stochastic programming, an attempt to capture uncertainty in the data underlying the optimization problem. This is not done through the use of random variables, but instead, the problem is solved taking into account inaccuracies in the input data.

An elegant and robust method of locating a minimum in such a bracket is the Golden Section Search. This involves evaluating the function at some point x in the larger of the two intervals (a, b) or (b, c). If $f(x) < f(b)$ then x replaces the midpoint b, and b becomes an end point. If $f(x) > f(b)$ then b remains the midpoint with x replacing one of the end points. Either way the width of the bracketing interval will reduce and the position of the minima will be better defined (Fig. 7.1). The procedure is then repeated until the width achieves a desired tolerance. It can be shown that if the new test point, x is chosen to be a proportion $3 - 5^{1/2}/2$ (hence Golden Section) along the larger sub-interval, measured from the mid-point **b**, then the width of the full interval (a, c) will reduce at an optimal rate.

The Golden Section Search requires no information about the derivative of the function. If such information is available it can be used to predict where best to choose the new point x in the above algorithm, leading to faster convergence.

Combinatorial optimization [18,19,24] – is concerned with problems where the set of feasible solutions is discrete or can be reduced to a discrete one.

Infinite-dimensional optimization studies the case when the set of feasible solutions is a subset of an infinite-dimensional space, such as a space of functions.

Disjunctive programming used where at least one constraint must be satisfied but not all. Trajectory optimization is the specialty of optimizing trajectories for air and space vehicles.

In a number of subfields, the techniques are designed primarily for optimization in dynamic contexts (that is, decision making over time):

- calculus of variations seeks to optimize an objective defined over many points in time, by considering how the objective function changes if there is a small change in the choice path,
- optimal control theory is a generalization of the calculus of variations,
- dynamic programming studies the case in which the optimization strategy is based on splitting the problem into smaller subproblems. The equation that describes the relationship between these subproblems is called the Bellman equation [9,10].

Mathematical programming with equilibrium constraints is where the constraints include variational inequalities or complementarities.

Multiobjective optimization: adding more than one objective to an optimization problem adds complexity. For example, if you wanted to optimize a structural design, you would want a design that is both light and rigid. Because these two objectives conflict, a trade-off exists. There will be one lightest design, one stiffest design, and an infinite number of designs that are some compromise of weight and stiffness. This set of trade-off designs is known as a **Pareto set**. The curve created plotting weight against stiffness of the best designs is known as the **Pareto frontier**. A design is judged to be Pareto optimal if it is not dominated by other designs: a Pareto optimal design must be better than another design in at least one aspect. If it is worse than another design in all respects, then it is dominated and is not Pareto optimal.

Optimization problems are often multi-modal, that is they possess multiple good solutions. They could all be globally good (same cost function value) or there could be a mix of globally good and locally good solutions. Obtaining all the multiple solutions is the goal of a multi-modal optimizer.

Classical optimization techniques due to their iterative approach do not perform satisfactorily when they are used to obtain multiple solutions, since it

is not guaranteed that different solutions will be obtained even with different starting points in multiple runs of the algorithm.

Evolutionary algorithms are however a very popular approach to obtain multiple solutions in a multi-modal optimization task. See Evolutionary multi-modal optimization.

Dimensionless optimization (DO) is used in design problems, and consists of the following steps:

- rendering the dimensions of the design dimensionless,
- selecting a local region of the design space to perform analysis on creating an I-optimal design within the local design space,
- forming response surfaces based on the analysis.

Optimizing the design based on the evaluation of the objective function, using the response surface models. An optimization problem can be represented in the following way: given a function $f : A \in R$ from some set A to the real numbers. Sought: an element x_0 in A such that $f(x_0) \leq f(x)$ for all x in A ('minimization') or such that $f(x_0) \geq f(x)$ for all x in A ('maximization').

Such a formulation is called an optimization problem or a mathematical programming problem (a term not directly related to computer programming, but still in use for example in linear programming). Many real-world and theoretical problems may be modeled in this general framework.

Problems formulated using this technique in the fields of physics and computer vision may refer to the technique as energy minimization, speaking of the value of the function f as representing the energy of the system being modeled. Typically, A is some subset of the Euclidean space R^n, often specified by a set of constraints, equalities or inequalities that the members of A have to satisfy. The domain A of f is called the search space or the choice set, while the elements of A are called candidate solutions or feasible solutions. The function f is called, variously, an objective function, cost function, energy function, or energy functional. A feasible solution that minimizes (or maximizes, if that is the goal) the objective function is called an optimal solution.

Generally, when the feasible region or the objective function of the problem does not present convexity, there may be several local minima and maxima,

where a local minimum x^* is defined as a point for which there exists some $\delta > 0$ so that for all x such that the expression holds; that is to say, on some region around x^* all of the function values are greater than or equal to the value at that point. Local maxima are defined similarly. A large number of algorithms proposed for solving non-convex problems – including the majority of commercially available solvers – are not capable of making a distinction between local optimal solutions and rigorous optimal solutions, and will treat the former as actual solutions to the original problem. The branch of applied mathematics and numerical analysis that is concerned with the development of deterministic algorithms that are capable of guaranteeing convergence in finite time to the actual optimal solution of a non-convex problem is called global optimization [8].

Optimization problems are often expressed with special notation. Here are some examples: this asks for the minimum value for the objective function $x_2 + 1$, where x ranges over the real numbers. The minimum value in this case is 1, occurring at $x = 0$.

This asks for the maximum value for the objective function $2x$, where x ranges over the reals. In this case, there is no such maximum as the objective function is unbounded, so the answer is 'infinity' or 'undefined'.

For the value (or values) of x in the interval that minimizes (or minimize) the objective function $x_2 + 1$ (the actual minimum value of that function does not matter). In this case, the answer is $x = -1$.

This asks for the (x, y) pair (or pairs) that maximizes (or maximize) the value of the objective function , with the added constraint that x lies in the interval $[-5, 5]$ (again, the actual maximum value of the expression does not matter). In this case, the solutions are the pairs of the form $(5, 2k\pi)$ and $(-5, (2k + 1)\pi)$, where k ranges over all integers.

The satisfiability problem, also called the feasibility problem, is just the problem of finding any feasible solution at all without regard to objective value. This can be regarded as the special case of mathematical optimization where the objective value is the same for every solution, and thus any solution is optimal.

Many optimization algorithms need to start from a feasible point. One way to obtain such a point is to relax the feasibility conditions using a slack variable; with enough slack, any starting point is feasible. Then minimize

that slack variable until slack is null or negative.

The extreme value theorem of Karl Weierstrass states conditions under which an optimum exists. How can an optimum be found? One of Fermat's theorems states that optima of unconstrained problems are found at stationary points, where the first derivative or the gradient of the objective function is zero. More generally, they may be found at critical points, where the first derivative or gradient of the objective function is zero or is undefined, or on the boundary of the choice set. An equation stating that the first derivative equals zero at an interior optimum is sometimes called a 'first-order conditions' [8].

Optima of inequality-constrained problems are instead found by the Lagrange multiplier method. This method calculates a system of inequalities called the 'Karush-Kuhn-Tucker conditions' or 'complementary slackness conditions', which may then be used to calculate the optimum.

While the first derivative test identifies points that might be optima, it cannot distinguish a point which is a minimum from one that is a maximum or one that is neither. When the objective function is twice differentiable, these cases can be distinguished by checking the second derivative or the matrix of second derivatives (called the Hessian matrix) in unconstrained problems, or a matrix of second derivatives of the objective function and the constraints called the bordered Hessian. The conditions that distinguish maxima and minima from other stationary points are sometimes called 'second-order conditions'.

How does the optimum change if the problem changes? The envelope theorem describes how the value of an optimal solution changes when an underlying parameter changes. The maximum theorem of Claude Berge (1963) describes the continuity of the optimal solution as a function of underlying parameters [8].

Optimization methods are crudely divided into two groups:

- SVO – Single-variable optimization,
- MVO – Multi-variable optimization.

For twice-differentiable functions, unconstrained problems can be solved by finding the points where the gradient of the objective function is zero (that

is, the stationary points) and using the Hessian matrix to classify the type of each point. If the Hessian is positive definite, the point is a local minimum, if negative definite, a local maximum, and if indefinite it is some kind of saddle point.

The existence of derivatives is not always assumed and many methods were devised for specific situations. The basic classes of methods, based on smoothness of the objective function, are:

- combinatorial methods,
- derivative-free methods,
- first-order methods,
- second-order methods.

Actual methods falling somewhere among the categories above include:

- bundle methods,
- conjugate gradient method,
- ellipsoid method,
- Frank-Wolfe metod.

Gradient descent aka steepest descent or steepest ascent interior point methods. Line search – a technique for one dimensional optimization, usually used as a subroutine for other, more general techniques:

- Nelder-Mead method aka the Amoeba metod,
- Newton's metod,
- Quasi-Newton methods,
- simplex metod.

Subgradient method – similar to gradient method in case there are no gradients. Global optimization is the branch of applied mathematics and numerical analysis that focuses on, well, optimization [8]. The goal of global optimization is to find the best possible elements x^* from a set X according to a set of criteria [8]:

$$F = \{f_1, f_2, \ldots, f_n\}. \tag{7.3}$$

These criteria are expressed as mathematical functions, the so-called objective functions.

Definition 3. (**Objective Function**). An objective function $f : X \to Y$ with $Y \subseteq R$ is a mathematical function which is subject to optimization [8].

The codomain Y of an objective function as well as its range must be a subset of the Real numbers ($Y \subseteq R$). The domain X of f is called problem space and can represent any type of elements like numbers, lists, construction plans, and so on. It is chosen according to the problem to be solved with the optimization process. Objective functions are not necessarily mere mathematical expressions, but can be complex algorithms that, for example, involve multiple simulations. Global optimization comprises all techniques that can be used to find the best elements x^* in X with respect to such criteria $f \in F$ [8]. Before digging any deeper into the matter, we will attempt to provide a classification of these algorithms as overview and discuss some basic use casus [8].

Generally, optimization algorithms can be divided in two basic classes: deterministic and probabilistic algorithms [4,5,8]. Deterministic algorithms are most often use dif a clear relation between the characteristics of the possible solutions and their utility for a given problem exists. Then, the search space can efficiently be explored using for example a divide and conquer scheme. If the relation between a solution candidate and its 'fitness' are not so obvious or too complicated, or the dimensionality of the search space is very high, it becomes harder to solve a problem deterministically. Trying it would possible result In exhaustive enumeration of the search space, which is not feasible even for relatively small problems [4,8,22].

Then probabilistic algorithms come into play. The initial work in this area which now has become one of most important research fields in optimization was started about 55 years ago. An especially relevant family of probabilistic algorithms are the Monte Carlo based approaches. They trade in guaranteed correctness of the solution for a shorter runtime. This does not mean that the results obtained using them are incorrect – they may just not be the

global optima. On the other hand, a solution a little bit interior to the best possible one is better than one which needs 10100 years to be fund [8].

Heuristics used in global optimization are functions that help decide which one of a set of possible solutions is to be examined next. On one hand, deterministic algorithms usually employ heuristics in order to define the processing order of the solution candidates. An example for such a strategy is informed search [4,5,8,21].

Probabilistic methods, on the other hand, may only consider those elements of the search space in further computations that have been selected by the heuristic [4,5,8].

Definition 4. (**Heuristic**). A heuristic is a part of an optimization algorithm that uses the information currently gathered by the algorithm to help to decide which solution candidate should be tested next or how the next individual can be produced [8].

Heuristics used in global optimization are functions that help decide which one of a set of possible solutions is to be examined next. On one hand, deterministic algorithms usually employ heuristics in order to define the processing order of the solution candidates.

Probabilistic methods, on the other hand, may only consider those elements of the search space in further computations that have been selected by the heuristic.

Definition 5. (**Metaheuristic**). A meta-heuristic is a method for solving very general classes of problems. It combines objective functions or heuristics in an abstract and hopefully efficient way, usually without utilizing deeper insight into their structure, i.e., by treating them as black-box-procedures [8].

This combination is often performed stochastically by utilizing statistics obtained from samples from the search space or based on a model of some natural phenomenon or physical process. Simulated annealing, for example, decides which solution candidate to be evaluated next according to the Boltzmann probability factor of atom configurations of solidifying metal melts. Evolutionary algorithms copy the behavior of natural evolution and treat solution candidates as individuals that compete in a virtual environment. An important class of probabilistic Monte Carlo meta-heuristics is Evolu-

tionary Computation. It encompasses all algorithms that are based on a set of multiple solution candidates (called population) which are iteratively refined. This field of optimization is also a class of Soft Computing as well as a part of the artificial intelligence area. Some of its most important members are evolutionary algorithms and Swarm Intelligence, which will be discussed in-depth in this book. Besides these nature-inspired and evolutionary approaches, there exist also methods that copy physical processes like the before-mentioned Simulated Annealing, Parallel Tempering, and Raindrop Method, as well as techniques without direct real-world role model like Tabu Search and Random Optimization.

The taxonomy just introduced classifies the optimization methods according to their algorithmic structure and underlying principles, in other words, from the viewpoint of theory. A software engineer or a user who wants to solve a problem with such an approach is however more interested in its 'interfacing features' such as speed and precision. Speed and precision are conflicting objectives, at least in terms of probabilistic algorithms. A general rule of thumb is that you can gain improvements in accuracy of optimization only by investing more time. Scientists in the area of global optimization try to push this Pareto frontier further by inventing new approaches and enhancing or tweaking existing ones.

Optimization Speed

When it comes to time constraints and hence, the required speed of the optimization algorithm, we can distinguish two main types of optimization cases.

Definition 6. (**Online Optimization**). Online optimization problems are tasks that need to be solved quickly in a time span between ten milliseconds to a few minutes. In order to find a solution in this short time, optimality is normally traded in for speed gains.

Examples for online optimization are robot localization, load balancing, services composition for business processes, or updating a factory's machine job schedule after new orders came in. From the examples, it becomes clear that online optimization tasks are often carried out repetitively – new orders will, for instance, continuously arrive in a production facility and need to be scheduled to machines in a way that minimizes the waiting time of all jobs.

Definition 7. (**Offline Optimization**). In offline optimization problems, time is not so important and a user is willing to wait maybe even days if it can be Number of Criteria [8].

Optimization algorithms can be divided in such which try to find the best values of single objective functions f and such that optimize sets F of target functions.

7.5 Multiobjective optimization

Optimization problems for real applications have often to consider many objectives and we thus have a multiobjective (MO) problem. A trade-off between the objectives exist and we never have a situation in which all the objectives can be in a best possible way be satisfied simultaneously. MO optimization provides the information of all possibilities of alternative solutions we can have for a given set of objectives. By analyzing the spectrum of solutions we have to decide which of these solutions is the most appropriate. The two steps are to solve the MO problem and decide what the optimal solution is. The MO problem (also called multicriteria optimization or vector optimization), can be defined as the problem of determining the following. A vector of decision variables which satisfies constraints and optimizes a vector function whose elements represent M objective functions. These functions form a mathematical description of performance criteria that are usually in conflict with each other. Hence optimizes means finding a solution which would give acceptable results as possible for the values of all objective functions.

We call decision variables x_j, $j = 1, \ldots, N$ for which values are to be chosen in an optimization problem. In order to know how good is a certain solution, we need to have some criteria for evaluation. These criteria are expressed as computable functions $f_1(x), \ldots, f_M(x)$ of the decision variables, which are called objective functions. These form a vector function $f = (f_1(x), \ldots, f_M(x)$. In general, some of these will be in conflict with others, and some will have to be minimized while others are maximized. The multiobjective optimization problem can be now defined as the problem to find the vector $x = (x_1, x_2, \ldots, x_N)$, i.e. solution which optimize the vector function f.

The constraints define the feasible region X and any point x in X defines a feasible solution. The vector function $f(x)$ is a function that maps the set

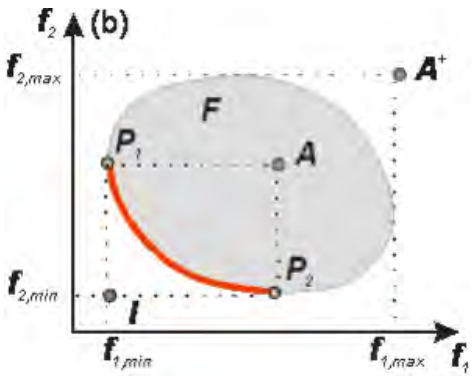


Figure 7.2: Example of a bi-objective space (f_1, f_2) [235]

X in the set F that represents all possible values of the objective functions. Normally we never have a situation in which all the $f_i(x)$ values have an optimum in X at a common point x. We therefore have to establish certain criteria to determine what would be considered an optimal solution. One interpretation of the term optimum in multiobjective optimization is the Pareto optimum, see Fig. 7.2.

A solution x_1 dominates a solution x_2 if and only if the two following conditions are true:

1. x_1 is no worse than x_2 in all objectives, i.e. $f_j(x_1) < f_j(x_2)$ $j = 1, \ldots, M$.
2. x_1 is strictly better than x_2 in at least one objective, i.e. $f_j(x_1) < f_j(x_2)$ for at least one $j \in 1, \ldots, M$.

We assume a minimization problem:

a) The Pareto front is the boundary between the points P_1 and P_2 of the feasible set F. Solutions 1 and 3 are non-dominated Pareto optimal solutions. Solution 2 is not Pareto optimal as solution 1 has simultaneously smaller values for both objectives. There is no reason why solution 2 should be accepted rather than solution 1. Therefore the aim of MO optimization is to obtain a representative set of non-dominated solutions.

b) The ideal and anti-ideal or *nadir* point I and A respectively. The point A^+ defined sometimes as anti-ideal is defined over the whole F range, whereas the points I and A are defined by the set of efficient points.

We assume, without loss of generality, that this is a minimization problem. The x_1 is said to be non-dominated by x_2 or x_1 is *non-inferior* to x_2 and x_2 is dominated by x_1.

Among a set of solutions P, the non-dominated set of solutions P' are those that are not dominated by any other member of the set P, see Fig. 7.1. When the set P is the entire feasible search space then the set P' is called the *global Pareto optimal set*. If for every member x of a set P there exists no solution in the neighborhood of x then the solutions of P form a *local Pareto optimal set*. The image $f(x)$ of the Pareto optimal set is called the *Pareto front*, see Fig. 7.1. The Pareto optimal set is defined in the parameter space, while the Pareto front is defined in the objective space.

The *ideal point* and the *nadir* or *anti-ideal point* are characterized by the components of the best and worse objective values of efficient points respectively.

Single objective optimization

In single objective optimization only one objective function is considered. Although many problems are multiobjective problems the majority of optimization algorithms used for their solution are single objective optimization algorithms. The M objective functions $f_m(x)$ are combined into a single objective function $f(x)$, e.g. by using a weighted sum of all objectives.

$$f(x) = \sum_{m=1}^{M} w_m f_m(x)\,, \tag{7.4}$$

The weights w_m are also known as importance factors and are considered as a measure of the significance of each objective in the optimization process. A representative convex part of the Pareto set can be sampled by running a single objective optimization algorithm each time with a different vector of importance factors.

We call a set of importance factors vectors normalized and uniformly distributed if each importance factor of each objective takes one of the following values: $[l/k, l = 0, 1, \ldots, k]$, where k is the sampling parameter and $\sum_{j=1}^{M} w_j = 1, w_j \geq 0, \forall j$ for each vector $\mathbf{w}$.

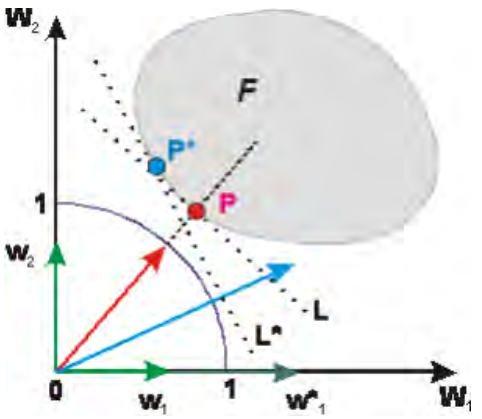


Figure 7.3: Single objective weighted sum optimization $w_1 f_1 + w_2 f_2$ for a bi-objective problem [235]

The vector sum $\mathbf{w} = \sum_{j=1}^{M} w_j \, \mathbf{e}_j$ where $\mathbf{e}_j, j = 1, \ldots, M$ are unit vectors of the objective space. For two objectives the weighted sum is given by:

$$y = w_1 f_1(x) + w_2 f_2(x) \,, \quad \text{i.e.} \quad f_2(x) = -\frac{w_1}{w_2} f_1(x) + \frac{y}{w_2} \,. \tag{7.5}$$

The minimization of the weighted sum can be interpreted as finding the value of y for which the line with slope $-\frac{w_1}{w_2}$ just touches the boundary of F as it proceeds outwards from the origin. If x^* is a Pareto optimal solution then there exists a weight vector $\mathbf{w} = (w_1, w_2, \ldots, w_M)$, such that x^* is a solution of the multiobjective convex optimization problem.

Figure 7.3 shows an example for a weighted sum optimization for a bi-objective problem. The solution obtained by (w_1, w_2) is the non-dominated solution P. If the value f_1 is not satisfactory then we can increase the importance factor w_1 to w_1^*. The new set of importance factors w_1^*, w_2 specify a new direction and the solution is P^*. The value of f_1 of P^* is smaller but the value $f_2(P^*)$ is larger than $f_2(P)$. If we are not satisfied by the value of $f_2(P)$ we can try another importance factor w_1^{**} with $w_1^* > w_1^{**} > w_1$.

In a trial and error method the optimization is repeated with different importance factors until the optimization result is acceptable. The designer by a trial and error method determines the structure of the Pareto front.

While for two objectives a solution very close to the optimal can be found it is more difficult as more objectives are considered. The complexity of the Pareto front increases rapidly and also the combinatorial complexity.

To determine the dependence of the results on the importance factors with a sufficient accuracy requires repeating the optimization a large number of times with different importance factor combinations. The solution which is obtained in the conventional weighted sum approach depends on the shape of the Pareto front and the importance factors used.

For a given set of importance factors the vector sum of w_1, w_2, if we consider these as vectors, specifies a direction S shown by the dashed line. The optimization provides a solution which is the point P of a line perpendicular to the direction S that will touch the Pareto front as the line is moved away from the origin along S. Solution P^* will be obtained if we replace w_1 by a larger importance factor w_1^*.

Multiobjective optimization requires a decision making process as there is not a single solutions but a set of non-dominated solutions out of which the best must be chosen. Three strategies can be used.

An a priori method. The decision making (DCM) is specified in terms of a scalar function and an optimization engine is used to obtain the corresponding solution.

An a posteriori method. An optimization engine exists which finds all solutions. Decision making is applied at the end of the optimization **manually**, or using a decision engine. This method decouples the solutions from the decision making process. A new decision is possible without having to repeat the optimization.

Mixture of a priori and a posteriori methods. During the optimization periodically information can be used which may be used to reformulate the goals as some of these can physically not be achieved.

The non-dominated solutions provide information on the trade-off between the objectives. This trade-off is described by the form of the Pareto front. In Fig. 7.4 two different forms of trade-offs are shown.

In Fig. 7.4a) there is a strong trade-off between the objective f_1 and f_2. The smaller the f_1 value is that we want the larger is the corresponding f_2 value. We can see how much this depends on the choice of f_1. There is a weak trade-off between f_1 and f_2 in Fig. 7.4b). It is possible to minimize f_1 significant and close to the ideal point $(f_{1,\min}, f_{2,\min})$. Only very close to $f_{1,\min}$ we see a rapid increase of f_2. This is a case for a set of parameters for

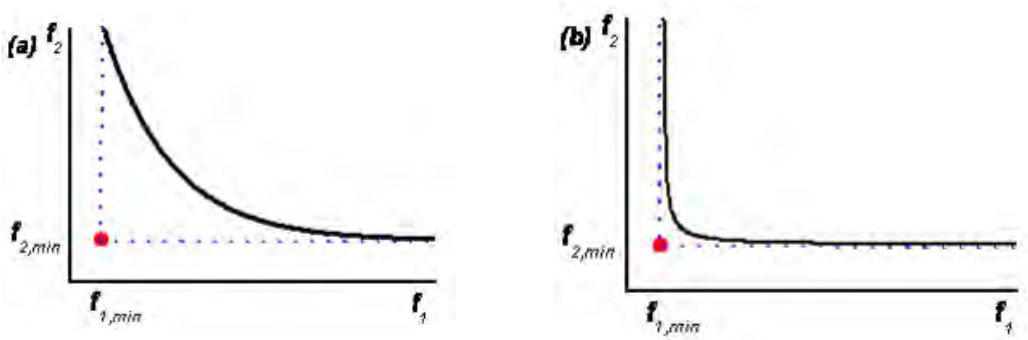


Figure 7.4: Trade-off between two objectives of a bi-objective problem: a) strong and b) weak trade-off between f_1 and f_2 [235]

which we can obtain simultaneously almost the individual optimal values for f_1 and f_2. The Pareto front provides also ranges of objective values.

For more than two objectives the trade-off can be difficult to analyze graphically. A sensitivity analysis can be performed numerically such that it considers the local slope of the Pareto front as a measure of the trade-off.

- Obtaining a representative set of non-dominated solutions,
- selecting a solution from this set, i.e. the decision making process.

Re-optimization

Re-optimization is understood literally as starting from the optimal solution of one problem to find an optimal solution for the next problem in the sequence. The multiobjective optimization using deterministic gradient based algorithms can be seen as an application of the algorithm many times based on a sequence of importance factors sets.

One method to obtain a representative sample of solutions faster, could the use of the result of a previous optimization as the starting point for the optimization with another set of importance factors $\mathbf{w}^*$ which is close to $\mathbf{w}$, see Fig. 7.5. With this approach it should be possible to reduce the number of the iterations required if the solutions are also close in objective space.

The method which produces sets of uniformly or randomly distributed importance factors has been modified using a rearrangement of the order of the

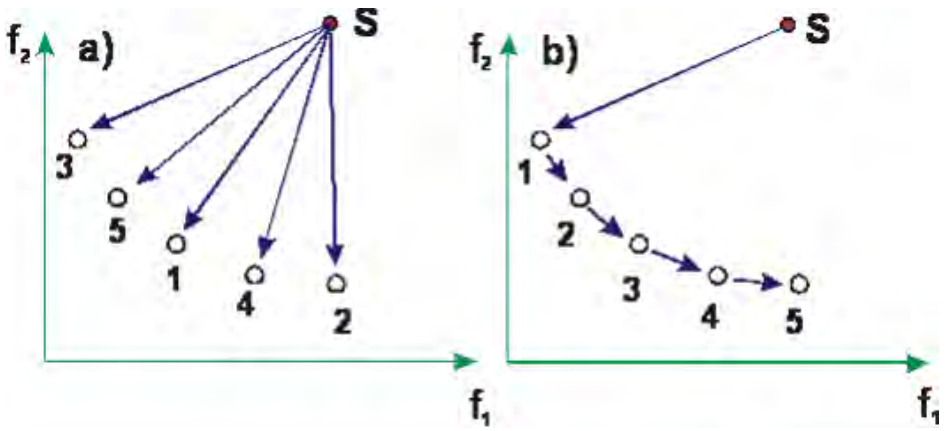


Figure 7.5: Optimization methods to obtain a consecutive number of solutions ordered in objective space: a) Cold Start b) Warm Start. The numbers show the order in which each of the five solutions is processed. This can be seen schematically in a) and b) if the lengths of all the arrows are summed. The total length is smaller in b) than in a) [235]

importance factors such that a spatial proximity of consecutive sets in importance factors space is obtained. This method has been used to test whether there are trapping regions. The rearrangement is shown in Fig. 7.6a) for randomly distributed importance factors and in Fig. 7.6b) for uniformly distributed importance factors for 3 objectives. Even for the initial uniformly distributed importance factors we have a reduction of the Euclidean distance in importance space.

Traditionally, application of Pareto Optimality principles have been applied in the *detailed design* phase of engineering design. However, Mattson and Messac (2002) are using Pareto fronts to aid *concept* selection. Pareto curves are generated for concept alternatives, which exist within feasible regions.

7.6 Pareto-optimization

The mathematical foundations for multi-objective optimization which considers conflicting criteria in a fair way has been laid by Vilfredo Pareto 110 years ago. Pareto optimality became an important notion in economics, game theory, engineering, and social sciences [2,6,8,99,209,211,223]. It defines the frontier of solutions that can be reached by trading-off conflicting objectives in an optimal manner. From this front, a decision maker (be it a human or

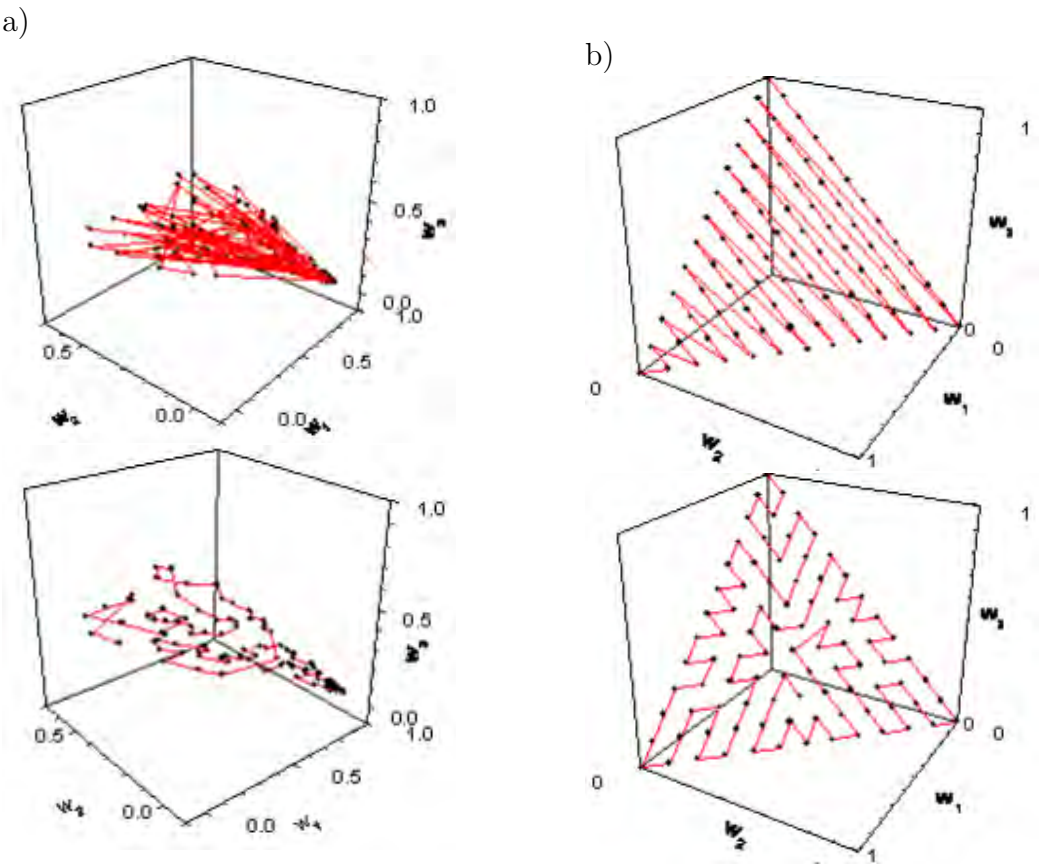


Figure 7.6: Rearrangement of solutions in importance space: a) solutions using randomly distributed importance factors before and after sorting b) uniformly distributed importance factors before and after sorting [235]

an algorithm) can finally choose the configurations that, in his opinion, suit best. The notation of optimal in the Pareto sense is strongly based on the definition of domination.

Definition 8. (**Domination**). An element x_1 dominates (is preferred to) an element x_2 if x_1 is better than x_2 in at least one objective function and not worse with respect to all other objectives. Based on the set F of objective functions f, we can write:

$$x_1 \vdash x_2 \Leftrightarrow \forall i \; i : 0 < i \leq n \Rightarrow w_i f_i(x_1) \leq w_i f_i(x_2) \wedge \\ \exists j : 0 < j \leq n : w_j f_j(x_1) < w_j f_j(x_2)\,, \tag{7.6}$$

where $w_i = 1$ if f_i should be minimized, or -1 if f_i should be maximized.

Different from the weights in the weighted sum approach, the factors w_i only carry sign information which allows us to maximize some objectives and to minimize some other criteria. The Pareto domination relation defines a strict partial order on the space of possible objective values. In contrast, the weighted sum approach imposes a total order by projecting it into the real numbers R.

Definition 9. (**Pareto Optimal**). An element $x^* \in X$ is Pareto optimal (and hence, part of the optimal set X^*) if it is not dominated by any other element in the problem space X. In terms of Pareto optimization, X^* is called the Pareto set or the Pareto Frontier.

$$x^* \in X \Leftrightarrow Ax \in X : x \vdash x* \,. \tag{7.7}$$

We assume again that f_1 and f_2 should both be maximized and hence, $w_1 = w_2 = -1$. The areas shaded with dark gray are Pareto optimal and thus, represent the optimal set:

$$X^* = [x_2, x_3] \cup [x_5, x_6] \,, \tag{7.8}$$

which here contains infinite many elements. All other points are dominated, i.e. not optimal.

The points in the area between x_1 and x_2 (shaded in light gray) are dominated by other points in the same region or in $[x_2, x_3]$, since both functions f_1 and f_2 can be improved by increasing x. If we start at the leftmost point in X (which is position x_1), for instance, we can go one small step to the right and will find a point x_1+ dominating x_1 because:

$$f_1(x_1 + \beta) > f_1(x_1) \quad \text{and} \quad f_2(x_1 + \beta) > f_2(x_1) \,. \tag{7.9}$$

We can repeat this procedure and will always find a new dominating point. At x_3 however, f_2 steeply falls to a very low level. A level lower than $f_2(x_5)$. Since the f_1 values of the points in $[x_5, x_6]$ are also higher than those of the points in $(x_3, x_4]$, all points in the set $[x_5, x_6]$ (which also contains the global maximum of f_1) dominate those in $(x_3, x_4]$. For all the points in the white area between x_4 and x_5 and after x_6, we can derive similar relations. All of them are also dominated by the non-dominated regions that we have just discussed. The higher this number, the worst is the element x in terms of Pareto optimization. His Pareto ranking approach is also used in many

optimization algorithms as part of the fitness assignment scheme. A non-dominated element is, as the name says, not dominated by any other solution candidate. These elements are Pareto optimal and have a domination-count of zero [8].

Problems of Pure Pareto Optimization

The complete Pareto optimal set is often not the wanted result of an optimization algorithm. Usually, we are rather interested in some special areas of the Pareto front only.

7.6.1 Artificial Ant Example 2

We can again take the Artificial Ant example to visualize this problem [8].

1. Maximize the amount of food piled.
2. Minimize the distance covered or the time needed to find the food.
3. Minimize the size of the program driving the ant.

Pareto optimization may now yield for example:

1. a program consisting of 100 instructions, allowing the ant to gather 50 food items when walking a distance of 500 length units,
2. a program consisting of 100 instructions, allowing the ant to gather 60 food items when walking a distance of 5000 length units,
3. a program consisting of 10 instructions, allowing the ant to gather 1 food item when walking a distance of 5 length units,
4. a program consisting of 0 instructions, allowing the ant to gather 0 food item when walking a distance of 0 length units.

The result of the optimization process obviously contains two useless but non-dominated individuals which occupy space in the population and the non-dominated set. We also invest processing time in evaluating them, and even worse, they may dominate solutions that are not optimal but fall into the

space behind the interesting part of the Pareto front. Furthermore, memory restrictions usually force us to limit the size of the list of non-dominated solutions found during the search. When this size limit is reached, some optimization algorithms use a clustering technique to prune the optimal set while maintaining diversity.

On one hand, this is good since it will preserve a broad scan of the Pareto frontier. In this case on the other hand, a short but dumb program is of course very different from a longer, intelligent one. Therefore, it will be kept in the list and other solutions which differ less from each other but are more interesting for us will be discarded.Furthermore, non-dominated elements have a higher probability of being explored further. This then leads inevitably to the creation of a great proportion of useless offspring. In the next generation, these useless offspring will need a good share of the processing time to be evaluated. Thus, there are several reasons to force the optimization process into a wanted direction.

7.6.2 Constraint Handling

Such a region of interest is one of the reasons for one further extension of the definition of optimization problems: in many scenarios, p inequality constraints g and q equality constraints may be imposed additional to the objective functions. Then, a solution candidate x is feasible, if and only if $g_i(x) \geq 0 \ \forall \ i = 1, 2, \ldots, p$ and $h_i(x) = 0 \ \forall \ i = 1, 2, \ldots, q$ holds. Obviously, only a feasible individual can be solution, i.e., an optimum, for a given optimization problem [8].

Death Penalty

Probably the easiest way of dealing with constraints is to simply reject all infeasible solution candidates right away and not considering them any further in the optimization process. This death penalty [8] can only work in problems where the feasible regions are very large and will lead the search to stagnate in cases where this is not the case. Also, the information which could be gained from the infeasible individuals is discarded with them and not used during the optimization.

Penalty Functions

Maybe one of the most popular approach for dealing with constraints, especially in the area of single-objective optimization, goes back to Courant, who introduced the idea of penalty functions in 1943. Here, the constraints are combined with the objective function f, resulting in a new function f' which is then actually optimized. The basic idea is that this combination is done in a way which ensures that an infeasible solution candidate has always a worse f'-value than a feasible one with the same objective values.

There are practically no limits for the ways in which a penalty for infeasibility can be integrated into the objective functions. Several researchers suggest dynamic penalties chich incorporate the index of the current iteration of the optimizer or adaptive penalties which additionally utilize population statistics [8].

Constraints as Additional Objectives

Another idea for handling constraints would be to consider them as new objective functions. The minimum is needed since there is no use in maximizing g further than 0 and hence, after it reached 0, the optimization pressure must be removed. An approach similar to this is Deb's Goal Programming method [8].

External Decision Maker

All approaches for defining what optima are and how constraints should be considered are rather specific and bound to certain mathematical constructs. The more general concept of an External Decision Maker which (or who) decides which solution candidates prevail has been introduced by Fonseca and Fleming [8]. One of the ideas behind 'externalizing' the assessment process on what is good and what is bad is that Pareto optimization imposes only a partial order on the solution candidates. In a partial order, elements may exists which neither succeed nor precede each other.

Most fitness assignment processes, however, require some sort of total order, where each individual is either better or worse than each other (except for the case of identical solution candidates which are, of course, equal to each other). The fitness assignment algorithms can create such a total order by themselves.

While this method of ordering is a good default approach able of directing the serach into the direction of the Pareto frontier and delivering a broad scan of it, it neglects the fact that the user of the optimization most often is not interested in the whole optimal set but has preferences, certain regions of interest [8,22,24,28]. What the user wants is a detailed scan of these areas, which often cannot be delivered by pure Pareto optimization.

Here comes the External Decision Maker as an expression of the user's preferences [8] into play. The task of this decision maker is to provide a cost function $u : Y \to R$ (or utility function, if the underlying optimizer is maximizing) chich maps the space of objective values Y (which is usually R_n) to the space of real numbers R. Since there is a total order defined on the real numbers, this process is another way of resolving the 'incomparability-situation'. The structure of the decision making process u can freely be defined and may incorporate any of the previously mentioned methods. Furthermore, it may even incorporate forms of artificial intelligence, other forms of multi-criterion Decision Making, and even interaction with the user. This technique allows focusing the search onto solutions which are not only optimal in the Pareto sense, but also feasible and interesting from the viewpoint of the user.

Fonseca and Fleming make a clear distinction between fitness and cost values. Cost values have some meaning outside the optimization process and are based on user preferences. Fitness values on the other hand are an internal construct of the search with no meaning outside the optimizer. If External Decision Makers are applied in evolutionary algorithms or other search paradigms that are based on fitness measures, these will be computed using the values of the cost function instead of the objective functions.

Prevalence Optimization

We have now discussed various approaches which define optima in terms of multi-objective optimization and steer the search process into their direction. Let us subsume all of them in general approach. From the concept of Pareto optimization to the Method of Inequalities, the need to compare elements of the problem space in terms of their quality as solution for a given problem winds like a read thread through this matter. Even the weighted sum approach and the External Decision Maker do nothing else than mapping multi-dimensional vectors to the real numbers in order to make them comparable.

If we compare two solution candidates x_1 and x_2, either x_1 is better than x_2, vice versa, or both are of equal quality. Hence, there are three possible relations between two elements of the problem space. These two results can be expressed with a comparator function cmpF.

It is easy to see that we can define Pareto domination relations and Method of Inequalities-based comparisons, as well as the weighted sum combination of objective values based on this notation. Together with the fitness assignment strategies which will be introduced later in this book, it covers many of the most sophisticated multi-objective techniques that are proposed, for instance in [8].

By replacing the Pareto approach with prevalence comparisons, all the optimization algorithms (especially many of the evolutionary techniques) relying on domination relations can be used in their original form while offering the new ability of scanning special regions of interests of the optimal frontier.

7.6.3 Artificial Ant Example 3

With the prevalence comparator, we can also easily solve the problem by no longer encouraging the evolution of useless programs for Artificial Ants while retaining the benefits of Pareto optimization. The comparator function simple can be defined in a way that they will always be prevailed by useful programs. It therefore may incorporate the knowledge on the importance of the objective functions. Let f_1 be the objective function with an output proportional to the food piled, f_2 would denote the distance covered in order to find the food, and f_3 would be the program length. Demonstrates one possible comparator function for the Artificial Ant problem [8].

7.7 The Structure of Optimization

After we have discussed what optima are and have seen a crude classification of global optimization algorithms, let us now take a look on the general structure common to all optimization processes. This structure consists of a number of well-defined spaces and sets as well as the mappings between them.

Based on this structure of optimization, we will introduce the abstractions fitness landscapes, problem landscape, and optimization problem which will lead us to a more thorough definition of what optimization is.

7.7.1 Spaces, Sets, and Elements

In this section, we elaborate on the relation between the (possibly different) representations of solution candidates for search and for evaluation. We will show how these representations are connected and introduce fitness as a relative utility measures defined on sets of solution candidates. You will find that the general model introduced here applies to all the global optimization methods mentioned in this book, often in a simplified manner.

The Problem Space and the Solutions therein Whenever we tackle an optimization problem, we first have to define the type of the possible solutions. For deriving a controller for the Artificial Ant problem, we could choose programs or artificial neural networks as solution representation. If we are to find the root of a mathematical function, we would go for real numbers R as solution candidates and when configuring or customizing a car for a sales offer, all possible solutions are elements of the power set of all optional features. With this initial restriction to a certain type of results, we have specified the problem space X [8].

Definition 10. (**Problem Space**). The problem space X (phenome) of an optimization problem is the set containing all elements x which could be its solution [8].

Usually, more than one problem space can be defined for a given optimization problem. A few lines before, we said that as problem space for finding the root of a mathematical function, the real number R would be fine. On the other hand, we could as well restrict ourselves to the natural numbers N or widen the search to the whole complex plane . This choice has major impact: On one hand, it determines which solutions we can possible find. On the other hand, it also has subtle influence on the search operations. Between each two different points in R, for instance, there are infinitely many other numbers, while in N, there are not.

In dependence on genetic algorithms, we often refer to the problem space synonymously phenome. The problem space X is often restricted by:

1. logical constraints that rule out elements which cannot be solutions, like programs of zero length when trying to solve the Artificial Ant problem and

2. practical constraints that prevent us, for instance, from taking all real numbers into consideration in the minimization process of a real function. On our off-the-shelf CPUs or with the Java programming language, we can only use 64 bit floating point numbers.

With these 64 bit, it is only possible to express numbers up to a certain precision and we cannot have more than 15 or so decimals.

Definition 11. (**Solution Candidate**). A solution candidate x is an element of the problem space X of a certain optimization problem [8].

In the context of evolutionary algorithms, solution candidates are usually called phenotypes. In this book, we will use both terms synonymously. Somewhere inside the problem space, the solutions of the optimization problem will be located (if the problem can actually be solved, that is).

Definition 12. (**Solution Space**). We call the union of all solutions of an optimization problem its solution space S [8].

This solution space contains (and can be equal to) the global optimal set. There may exist valid solutions which are not elements of the especially in the context of constraint optimization.

7.7.2 The Objective Space and Optimization Problems

After the appropriate problem space has been defined, the search space has been selected and a translation between them (if needed) was created, we are almost ready to feed the problem to a global optimization algorithm. The main purpose of such an algorithm obviously is to find as many elements as possible from the solution space – We are interested in the solution candidates with the best possible evaluation results. This evaluation is performed by the set F of n objective functions f, each contributing one numerical value describing the characteristics of a solution candidate x.

Definition 13. (**Objective Space**). The objective space Y is the space spanned by the codomains of the objective functions.

$$F = \{f_i : X \to Y_i : 0 < i \leq n,\ Y_i \subseteq R\} \Rightarrow Y = Y_1 \times Y_2 \times \ldots \times Y_n\,. \quad (7.10)$$

The set F maps the elements x_j of the problem space X to the objective space Y and, by doing so, gives the optimizer information about their qualities as solutions for a given problem, $j = 1, 2, \ldots, m$ [8].

7.8 The algorithms of optimization

Throughout human being history, mankind has faced optimization problems and made great efforts to solve them. Loosely speaking, optimization is the process of finding the best way to use available resources, while at the same time not violating any of the constraints that are imposed. More accurately,we may say that we wish to define a system mathematically, identify its variables and the conditions they must satisfy, define properties of the system, and then seek the state of the system (values of the variables) that gives the most desirable (largest or smallest) properties. This general process is referred to as optimization. It is not our purpose here to define a system. This is the central problem of various disciplines which are sciences or are struggling to become sciences. Our concern here is, given a meaningful system, what variables will make the system have the desirable properties.

There might be better formulation of objective and constraint functions to describe a particular optimization problem. Any knowledge about the optimization problem should be worked into the objective and constraint functions. Good objective and constraint functions can make all the difference.

7.8.1 Optimization Parameters

Optimization parameters x_j are critical for an optimization problem. They affect the value of objective and constraint functions. If there are no optimization parameters, we cannot define the objective and constraint functions. In the investment fund management problem, the optimization parameters are the amounts of money invested in each fund. In experimental data fitting problems, the optimization parameters are the parameters that define

the model. In the radome design problem, the optimization parameters might include the material index in the materiale database, material thickness, and some other parameters.

An optimization parameter can be continuous, discrete, or even symbolic. For instance, in the investment fund management problem, the fund manager wants to maximize the return. In fitting experimental data to a user-defined model, we might minimize the total deviation of observed data from predictions based on the model. In the radome design problem, we have to maximize the strength and minimize the distortion and cost.

Almost all optimization problems have objective functions. However, in some cases, such as the design of integrated circuit layouts, the goal is to find optimization parameters that satisfy the constraints of the model. The user does not particularly want to optimize anything, so there is no reason to define an objective function. This type of problem is usually called a feasibility problem. On the other hand, in some optimization problems, there is more than one objective function. For instance, in the radome design problem, it would be nice to minimize weight and maximize strength simultaneously.

Optimization has a consistent track record across a wide range of science, engineering, industry and commerce. In fact, many optimization problems come directly from real-world applications. A simple search on Google with the keywords 'optimization' and 'application' will get numerous hits. Publications on optimization that do not mention applications are very rare. There is no need for us to prove the usefulness of optimization by presenting a long list of fields of application in which optimization has been involved. Space considerations also do not permit us to give an exhaustive and in-depth review on applications of optimization. Therefore,no further discussion on applications of optimization will be given here.

7.8.2 Enumeration of basic of Optimization Algorithms

Optimization has long been the subject of intensive study. Numerous optimization algorithms have been proposed. In general, these algorithms can be divided into two major categories, deterministic and stochastic. Hybrid algorithms which combine deterministic and stochastic features are stochastic in essence and regarded as such. However, it is acceptable to treat them as a third category from the point of view of purity.

1. **Deterministic Optimization Algorithms**

   A deterministic optimization algorithm will always get the same solution with the same number of objective function evaluations regardless of the time it is started, if the search space, starting-point, and termination conditions are unchanged. If the algorithm is run multiple Times on the same computer, the search time for each run will be exactly the same. In other words, deterministic optimization is clonable. interpolation algorithm, and the Brent algorithm. If the minimum of the objective function $f(x)$ is known, a nonlinear equation can be formulated. In this case, the Secant algorithm for nonlinear equations is applicable.

2. **Exhaustive Search Algorithm**

   The exhaustive search algorithm samples the search space $[a, b]$ at m points. Usually, the sample points are equally spaced within $[a, b]$. The minimum value of the objective function at each and every sample point is regarded as the optimum and the corresponding sample point is regarded as the optimal solution. The exhaustive search algorithm is also known as enumeration algorithm or brute force algorithm.

3. **Dichotomous Algorithms**

   There are many different schemes for obtaining y_1 and y_2 within $[bL, bU]$, such as the equal interval, Fibonacci, and golden section schemes.

4. **Parabolic Interpolation Algorithm**

   A local quadratic approximation to the objective function $f(x)$ is useful because the minimum of a quadratic is easy to compute.

5. **Brent Algorithm**

   The Brent algorithm is a hybrid of the parabolic interpolation algorithm and the golden section algorithm. The objective function in each iteration is approximated by an interpolating parabola through three existing points. The minimum point of the parabola is taken as a guess for the minimum point. It is accepted and used to generate a smaller interval if it lies within the bounds of the current interval. Otherwise, the algorithm falls back to an ordinary golden section step.

6. **Secant Algorithm for Nonlinear Equation**

   For an objective function $f(x)$ with known optimum, it is straightforward to formulate a nonlinear equation $f(x)\, L'0$. Its truncated Taylor

series $f(xh)$ is a linear function of h that approximates $f(x)$ near a given x. Taylor series is only an approximation to the nonlinear function $f(x)$, its root xh does not equal to the root of $f(x)$. Therefore, this process has to be repeated until an acceptable root is located. This motivates the update scheme of the Newton algorithm for nonlinear equation [8].

7. **Higher-Order One-Dimensional Deterministic Optimization Algorithms**

   Besides objective function evaluation and comparison, higher-order algorithms directly make use of derivatives. Three algorithms in this category are commonly used.

8. **Newton Algorithm**

   Another way to obtain a local quadratic approximation to the objective function $f(x)$ is to use a truncated Taylor series expansion. The minimum point of the parabola, a new estimate of the minimum point of the objective function, will replace one of the three previous points. This process is repeated until termination conditions are fulfilled.

   Dimension is a good criterion for classifying deterministic optimization algorithms. Deterministic optimization algorithms are accordingly divided into one-dimensional and multi-dimensional deterministic optimization algorithms. Some multi-dimensional deterministic algorithms need the help of one-dimensional deterministic optimization algorithms.

9. **One-Dimensional Deterministic Optimization Algorithms**

   Use of the derivative is a good choice for distinguishing one-dimensional optimization algorithms.

10. **Zeroth-Order One-Dimensional Deterministic Optimization Algorithms**

    Zeroth-order algorithms involve objective function evaluation and comparison only. Prominent algorithms include the exhaustive search algorithm, dichotomous algorithms.

11. **Secant Algorithm**

    The Secant algorithm here is equivalent to the Secans algorithm for nonlinear equation $f0(x)\,L'0$. Finite difference is applied to approximate the second-order derivative.

12. **Cubic Interpolation Algorithm**

    This is another polynomial approximation algorithm in which the objective function $f(x)$ is approximated by a local third-order polynomial $p_3(x)$. The basic logic is similar to that of the paraboli interpolation algorithm. However, in this instance, evaluation of both objective function and its derivative at each point is required. Consequently, the approximation polynomial can be constructed using fewer points.

13. **Multi-Dimensional Deterministic Optimization Algorithms**

    Similarly, the use of gradient, Hessian matrix or even higher-order partial derivatives is a good way to distinguish multi-dimensional optimization algorithms.

14. **Zeroth-Order Multi-Dimensional Deterministic Optimization Algorithms**

    Likewise, zeroth-order algorithms here also involve objective function evaluation and comparison only. Prominent algorithms include the grid search algorithm, univariate search algorithms, the pattern search algorithm, Powell's conjugate direction algorithm, and the downhill simplex algorithm. If the minimum of the objective function $f(x)$ is known, a nonlinear equation can be formulated. In this case, Broyden's algorithm for nonlinear equation is applicable.

15. **Grid Search Algorithm**

    The grid search algorithm is the simplest algorithm for finding the minimum and the corresponding minimum solution point of objective function $f(x)$. The D_i the search space of x_i, is sampled at mi points to form solution point. Obviously this approach soon becomes prohibitive as the dimension N and the number of sample points m_i for each dimension increase. A more efficient approach starts from a grid point, evaluates the objective function values at the surrounding $3N_1$ grid points, selects the grid point with the smallest objective function value as the new starting-point, and repeats this process until termination conditions are fulfilled.

16. **Univariate Search Algorithms**

    The guiding idea behind univariate serach algorithms is to change one optimization parameter at a time so that the function is optimized in each of the coordinate directions.

Univariate search algorithms are regarded as zeroth-order optimization algorithms for multi-dimensional optimization problems because the gradient of the objective function is not explicitly involved, although the derivative with respect to l might be involved in the inner one-dimensional optimization algorithm.

Powell proposed a simple but vastly superior variation. It gets the name Powell's conjugate direction algorithm because it chooses conjugate directions to move when applied to an objective function of quadratic form.

17. **Downhill Simplex Algorithm**

The downhill simplex algorithm for minimizing $f(x)$ is due to Nelder and Mead. A non-degenerate N-dimensional simplex is a geometrical figure consisting of $N1$ distinct vertices. For example, a two-dimensional simplex is a triangle and three-dimensional simplex is a tetrahedron. Suppose we have obtained an N-dimensional simplex.

18. **Physical Algorithms**

Stochastic optimization algorithms in this category are inspired by physical phenomena. The Monte Carlo algorithm and the simulated annealing algorithm [13,14] are two of the most prominent algorithms in this category

19. **Monte Carlo Algorithm**

The Monte Carlo algorithm, named for a famous casino city in Monaco, relies on repeated random sampling to find the optimal solution. The use of randomness and the repetitive nature of the process are analogous to the activities conducted at a casino. It was originally practiced under more generic names such as statistical sampling. In the 1940s, physicists working on nuclear weapons projects at the Los Alamos National Laboratory coined the present name.

20. **Simulated Annealing Algorithm**

The simulated annealing algorithm imitates the annealing process in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce defect. By analogy with this physical process, each step of the simulated annealing algorithm replaces the current solution by a new solution with a probability that depends on the difference of objective function values at

the two solution points and the temperature. The most often implemented probability distribution function is the Boltzmann probabilisty distribution.

21. **Evolutionary Algorithms**

    Evolutionary algorithms were inspired by Darwin's theory of evolution. Natural selection is the foundation of Darwin's theory of evolution. The study of evolutionary algorithms began in the 1960's. A number of creative researchers independently came up with the idea of mimicking the biological evolution mechanism and developed three mainstream evolutionary algorithms, namely, genetic algorithms, evolutionary programming and evolution strategies.

Outstanding Features

Stochastic optimization algorithms have many interesting features. Some of these features are controversial. Nevertheless, stochastic optimization has been gaining more and more popularity and acceptance.

Randomness

As mentioned earlier, deterministic optimization is clonable. In contrast, as the name indicates, results obtained from a stochastic optimization algorithm are in general unpredictable due to randomness. In practice, one may never be able to get identical optimal solutions, although the solutions obtained may differ only very slightly.

However, from the point of view of practical application, two mathematically different results are regarded as identical if both of them meet the tolerance requirement imposed by the practical application. A controversy accompanying stochastic optimization algorithms is their proof of absolute success, either theoretically or numerically. No stochastic optimization algorithm guarantees absolute success, although the failure percentage might be very small. A search by a stochastic optimization algorithm may miss the optimal solution. This constitutes a major challenge for the entire stochastic optimization community. Mathematicians working on it are a long way from a successful conclusion.

Simplicity

Stochastic optimization algorithms are in general mathematically simpler than deterministic algorithms. Usually, neither an exact nor an approximate derivative is involved. Most stochastic optimization algorithms generate their initial solution through an inherent initialization process, and thus avoid being troubled by choosing a starting-point. Relief from heavy reliance on trial and error or a priori knowledge in guessing starting-points is a tremendous advantage in the eyes of many optimization practitioners.

Another controversy as regards stochastic optimization algorithms is their rigorous mathematical foundation. Most of the stochastic optimization algorithms are inspired by natura phenomena which mankind has been struggling to understand differentiability of objective and constraint functions are the most common implications assumed by deterministic optimization algorithms. Unfortunately, many real-world application problems do not satisfy even one of these assumptions.

It is obvious that most deterministic optimization algorithms demand one or more starting points. Good starting-points are critical for the success of deterministic optimization algorithms. Poor starting-points may have a significant adverse effect on deterministic optimization algorithms' efficiency, or even cause them to fail This approach causes unnecessary confusion and should be abandoned. The original algorithms are strictly followed here unless explicitly stated otherwise Each stochastic optimization algorithm has at least one intrinsic control parameter. The performance of stochastic optimization algorithms more or less depends on these intrinsic control parameters. It is well known that tuning these intrinsic control parameters for better performance is usually very hard. In this sense, stochastic optimization algorithms are not as simple as people have believed.

Efficiency

Stochastic optimization algorithms usually require more objective function evaluations to find the optimal solution than deterministic optimization algorithms, given that both of them succeed. In other words, they are computationally more expensive or less efficient.

Robustness

This is the third controversy as far as stochastic optimization algorithms are concerned. Stochastic optimization algorithms may occasionally miss the optimal solution even if everything is favorable. On the other hand, stochastic optimization algorithms are usually capable of bracketing a quasi-optimal solution within a wide search space, while deterministic optimization algorithms usually fail to do so in the same situation. In most practical applications, a quasi-optimal solution is welcome. Very often, it is immediately accepted. In the event that it is not acceptable, certain measures can be taken to refine it.

Versatility

Most stochastic optimization algorithms do not impose restrictions on optimization problems. In addition, many stochastic optimization algorithms apply to discrete or even symbolic optimization parameters as well as real ones. In this regard, stochastic optimization optimizations are versatile.

Classification

Stochastic optimization algorithms can be divided into two major categories according to their origins: physical algorithms and evolutionary algorithms. Some people regard artificial neural networks and artificial immune systems as stochastic optimization algorithms. Indeed, they can be used to solve certain optimization problems. However, before solving the optimization problem at hand, training has to be carried out. These algorithms cannot accomplish the optimization by themselves without the help of training sets. For this reason, this autor personally would not regard them as stochastic optimization algorithms.Once again, we do not regard hybrids of two or more stochastic optimization alg orithms asa separate category.

Bibliography

[1] A. Goran: Introduction to Optimization – North Karolina State University, SAMSI NDHS Undegraund workshop, 2006.

[2] Anyong Qing: An Introduction to Optimization. Differential Evolution. John Wiley & Sons (Asia) Pte Ltd. 2009.

[3] M. Gilli: Numerical Optimization and Simulation. Department of Econometrics University of Geneva and Swiss Finance Institute, Spring 2008.

[4] B.T. Poliak: Vvedenie v optimizaciju. –M., Nauka, 1983, (in Russian).

[5] P. Klein: Numerical optimization. University of Western Ontario, paul.klein@uwo.ca , www.ssc.uwo.ca/economics/faculty/klein/ , 2007.

[6] A.H. Suhariev, A.V. Timohov, V.V. Fiodorov: Kurs metodov optimizacii. –M., Nauka, 1986, (in Russian).

[7] B. Pshenichny, J. Danilin: Chislennyje metody v ekstremalnych zadachah. –M., Nauka, 1975, (in Russian).

[8] T. Weise: Global Optimization Algorithms – Theory and Application, http://www.it-weise.de/ 2009-06-26.

[9] R. Bellman: Dinamicheskoje programirovanie. –M., Nauka, 1960, (in Russian).

[10] R. Bellman, S. Drejfus: Prikladnyje Zadachi dinamicheskogo programirovania. –M., Nauka, 1965, (in Russian).

[11] Cooper L., Steinberg D.: Introduction to Methods of Optimization, W.B. Saunders, Philadelphia, 1970.

[12] J. Branke, K. Deb, K. Miettinen, R.E. Steuer editors: Practical Approaches to Multi-Objective Optimization, No. 04461 in Dagstuhl Seminar Proceedings, Nov. 7-12, 2004, Dagstuhl, Germany. Internationales Begegnungs und Forschungszentrum fur Informatik (IBFI), Schloss Dagstuhl, Germany IBFI. 2005.

[13] J. Branke, K. Deb, K. Miettinen, R. Slowinski editors: Practical Approaches to Multi-Objective Optimization, No. 06501 in Dagstuhl Seminar Proceedings, December 10-15, 2006.

[14] Lam Thu Bui, Sameer Alam editors: Multi-Objective Optimization in Computational Intelligence: Theory and Practice. Premier Reference Source. Idea Group Publishing, 2008.

[15] H.G. Beyer: Design optimization of a linear accelerator using evolution strategy: solving a TSP-like optimization problem. In Handbook of Evolutionary Computation, pp. G4.2:1-8. Institute of Physics Publishing and Oxford University Press, 1997.

[16] R. Chiong editor: Nature-Inspired Algorithms for Optimisation, Vol. 193 of Studies in Computational Intelligence. Springer, April 30, 2009.

[17] I.V. Sergijenko: Matematicheskie modeli i metody reshenija zadach diskretnoj optimizacii. –Kijev, Naukova dumka, 1988, (in Russian).

[18] I.V. Sergijenko, T. Lebiedieva, V. Roschin: Priblizonnyje metody reshenija diskretnych zadach optimizacii. –Kijev, Naukova dumka, 1980, (in Russian).

[19] Pareto optimality. Georgia Institut of Technology. Systems Realization Laboratory, 2010.

[20] B.J. Frédéric et al.: Numerical optimization: Theoretical and practical aspects, Berlin: Springer-Verlag, 2006, pp. xiv+490.

[21] I.V. Sergijenko, V.P. Shylo: Zadachi diskretnej optimizacii: Problemy, metody, reshenija, issledovanije. –Kijev, Naukova dumka, 2003, (in Russian).

[22] V.M. Glushkov: O sistemnoj optimizacii.–Kibernetika, No. 5, 1980, pp. 3-5, (in Russian).

[23] I.V. Sergijenko, L. Kozerackaja, T. Lebedijeva: Issledovanije ustojchivosti i parametricheskij analiz diskretnych optimizacionnych zadach. –Kijev, Naukova dumka, 1995, (in Russian).

[24] N. Moisejev, J. Ivanilov, E. Stolarov: Metody optimizacii, –M., Nauka, 1978, (in Russian).

[25] A. Bronstein, M. Bronstein: Numerical Optimization, 2008: tosca.cs.technion.ac.il

[26] Sawaragi Y., Nakayama H., Tanino T.: Theory of Multiobjective Optimization (Vol. 176 of Mathematics in Science and Engineering), Orlando, FL: Academic Press Inc., 1985.

[27] T.W. Petersen: An Introduction to Numerical Optimization Methods and Dynamic Programming using C++. June 5th 2001, University of Copenhagen: toke.ward.petersen@econ.ku.dk.

[28] J. Sloan, D. Kesler, R. Kumar: A Numerical Optimization-based Methodology for Application Robustification. University of Illinois, Urbana-Champaign, jsloan,dkesler2,rakeshk@illinois.edu

[29] J. Sarich, T. Munson, J. More: Numerical Optimization and the Toolkit for Advanced Optimization. Mathematics and Computer Science Division, Argonne National Laboratory, 2009.

[30] J. Arneric, A. Rozga: Numerical Optimization within Vector of Parameters Estimation in Volatility Models, World Academy of Science, Engineering and Technology, 2009.

[31] J. Sloan, D. Kesler, R. Kumar: A Numerical Optimization-based Methodology for Application Robustification. University of Illinois, Urbana-Champaign, jsloan,dkesler2,rakeshk@illinois.edu

[32] J. Nocedal, J. Stephen: Wright Numerical Optimization. Second Edition Mathematics Subject Classification, 2000.

[33] F.P. Vasiliev: Chislennyje metody reshenia ekstremalnyh zadach. –M., Nauka, 1988, (in Russian).

[34] P.E. Gill, W. Murray, M.H. Wright: Numerical Linear Algebra and Optimization, Vol. 1, Addison Wesley, 1991.

[35] M.J.D. Powell: A Fast Algorithm for Nonlinearly Constrained Optimization Calculations. Numerical Analysis, G.A. Watson ed., Lecture Notes in Mathematics, Springer Verlag, Vol. 630, 1978.

[36] M.J.D. Powell: Variable Metric Methods for Constrained Optimization. Mathematical Programming: The State of the Art, (A. Bachem, M. Grotschel, B. Korte eds.) Springer Verlag, pp. 288-311, 1983.

[37] A. Neumaier: Global optimization and constraint satisfaction. In I. Bomze, I. Emiris, Arnold Neumaier, and L. Wolsey, editors, Proceedings of GICOLAG workshop (of the research project Global Optimization, Integrating 84. Convexity, Optimization, Logic Programming and Computational Algebraic Geometry), December 2006.

[38] M. Montaz Ali, Charoenchai Khompatraporn, Z.B. Zabinsky: A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. Journal of Global Optimization, 31(4):635-672, 2005.

[39] Peter John Angeline: Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In EP'98: Proceedings of the 7th International Conference on Evolutionary Programming VII, pp. 601-610, 1998.

[40] M. Avriel: Nonlinear Programming: Analysis and Methods. Dover Publishing Inc., September 16, 2003.

[41] N. Baba: Convergence of a random optimization method for constrained optimization problems. Journal of Optimization Theory and Applications, 33(4):451-461, 1981.

[42] V. Nissen, M. Krause: Constrained combinatorial optimization with an evolution strategy. In Proceedings of Fuzzy Logik, Theorie und Praxis, 4. Dortmunder Fuzzy-Tage, pp. 33-40, June 1994, Dortmund, Germany, Springer-Verlag, London.

[43] T.F. Coleman, Y. Li: On the Convergence of Reflective Newton Methods for Large-Scale Nonlinear Minimization Subject to Bounds. Mathematical Programming, Vol. 67, No. 2, pp. 189-224, 1994.

[44] T.F. Coleman, Y. Li: An Interior. Trust Region Approach for Nonlinear Minimization Subject to Bounds. SIAM Journal on Optimization, Vol. 6, pp. 418-445, 1996.

[45] T.F. Coleman, Y. Li: A Reflective Newton Method for Minimizing a Quadratic Function Subject to Bounds on some of the Variables. SIAM Journal on Optimization, Vol. 6, No. 4, pp. 1040-1058, 1996.

[46] Sorensen D.C.: Minimization of a Large Scale Quadratic Function Subject to an Ellipsoidal Constraint. Department of Computational and Applied Mathematics, Rice University, Technical Report TR94-27, 1994.

[47] E. Zitzler: Evolutionary algorithms for multiobjive optimization. Institute of Technology (ETH) Zurich, Gloriastr, Switzerland, http://www.tik.ee.ethz.ch/_zitzler

[48] Jouni Pousi, Kai Virtanen, Raimo P. Hamalainen: Multiple Criteria Optimization and Analysis in the Planning of Effects-Based Operations (EBO). Helsinki University of Technology jouni.pousi@hut.fi, kai.virtanen@hut.fi, raimo@hut.fi

[49] M.A. Branch, T.F. Coleman, Y. Li: A Subspace, Interior, and Conjugate Gradient Method for Large-Scale Bound-Constrained Minimization Problems. SIAM Journal on Scientific Computing, Vol. 21, No. 1, pp. 1-23, 1999.

[50] Biggs M.C.: Constrained Minimization Using Recursive Quadratic Programming. Towards Global Optimization (L.C.W.Dixon and G.P.Szergo eds.), North-Holland, pp. 341-349, 1975.

[51] R.K. Brayton, S.W. Director, G.D. Hachtel, L.Vidigal: A New Algorithm for Statistical Circuit Design Based on Quasi-Newton Methods and Function Splitting. IEEE Transactions on Circuits and Systems, Vol. CAS-26, pp. 784-794, Sept. 1979.

[52] Broyden C.G.: The Convergence of a Class of Double-rank Minimization Algorithms. J. Inst. Maths. Applics., Vol. 6, pp. 76-90, 1970.

[53] R.H. Byrd, R.B. Schnabel, G.A. Shultz: Approximate Solution of the Trust Region Problem by Minimization over Two-Dimensional Subspaces. Mathematical Programming, Vol. 40, pp. 247-263, 1988.

[54] I. Hong et al.: Power Optimization of Variable Voltage Core-Based Systems. IEEE Trans. Computer Aided Design, 18(12):1702-14, Dec. 1999.

[55] S. Martello, P. Toth: Knapsack Problems: Algorithms and Computer Implementations. John Wiley and Sons, 1990.

[56] P. Mejia-Alvarez, E. Levner, D. Mosse: Power-Optimized Scheduling Server for Real-Time Tasks. In Proc. the 8th IEEE Real-Time and Embedded Technology and Applications Symp., 2002.

[57] D. Pisinger: Algorithms for Knapsack Problems. PhD thesis, Dept. of Computer Science, University of Copenhagen, Denmark, 1995.

[58] Y. Shin, K. Choi, T. Sakurai: Power Optimization of Real-Time Embedded Systems on Variable Speed Processors. In Int. Conf. Computer-Aided Design, pp. 365-368, 2000.

[59] N. Ascheuer: Hamiltonian path problems in the on-line optimization of flexible manufacturing systems. Technische Universitat Berlin, Germany, 1995.

[60] L. Bianchi, L.M. Gambardella, M. Dorigo: An ant colony optimization approach to the probabilistic traveling salesman problem. In Proceedings of PPSN-VII, Seventh Inter 17 national Conference on Parallel Problem Solving from Nature, Lecture Notes in Computer Science. Springer Verlag, Berlin, Germany, 2002.

[61] C. Blum, M. Sampels: Ant colony optimization for FOP shop scheduling: a case study on different pheromone representations, in Proceedings of the 2002 congress on Evolutionary Computation, Honolulu, USA.

[62] T. Back, H.P. Schwefel: An overview of evolutionary algorithms for parameter optimization, Evolutionary Computation 1(1), pp. 1-23, 1993.

[63] M. den Besten, T. Stutzle, M. Dorigo: Ant colony optimization for the total weighted tardiness problem, Parallel Problem Solving from Nature: 6th international conference, September 2000. Springer Verlag.

[64] A. Colorni, M. Dorigo, V. Maniezzo: Distributed optimization by ant colonies, Proceedings of ECAL'91, European Conference on Artificial Life, Elsevier Publishing, Amsterdam, 1991.

[65] O. Cordon, I. Fernandez de Viana, F. Herrera, L. Moreno: A new ACO model integrating evolutionary computation concepts: the best-worst ant system, In Proceedings of ANTS 2000 – from ant colonies to artificial ants, Universite Libre de Bruxelles.

[66] M. Dorigo: Optimization, learning and natural algorithms, Ph.D. Thesis, Politecnico di Milano, Milano, 1992.

[67] M. Dorigo, V. Maniezzo, A. Colorni: The ant system: an autocatalytic optimizing process, Technical Report TR91-016, Politecnico di Milano, 1991.

[68] M. Dorigo, G. Di Caro: The Ant Colony Optimization Meta-Heuristic. In D. Corne, M. Dorigo and F. Glover editors, New Ideas in Optimization, McGraw-Hill, 1999, pp.11-32.

[69] M. Dorigo, G. Di Caro, L.M. Gambardella: Ant Algorithms for Discrete Optimization. Artificial Life, 5(2) 1999, pp. 137-172.

[70] M. Dorigo, V. Maniezzo, A. Colorni: The ant system: optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man and Cybernetics-Part B 26(1), 1996, pp. 29-41.

[71] M. Dorigo, T. Stutzle: The ant colony optimization metaheuristic: Algorithms, applications and advances. In F. Glover and G. Kochenberger editors, Handbook of Metaheuristics. Kluwer Academic Publishers, 2002.

[72] T.A. Feo, M.G.C. Resende: Greedy randomized adaptive search procedures, Journal of Global Optimization 6, 1995, pp. 109-133.

[73] W.J. Gutjahr: ACO algorithms with guaranteed convergence to the optimal solution. Information Processing Letters 82(3):145-153, 2002.

[74] L.M. Gambardella, E. Taillard, G. Agazzi: Ant colonies for vehicle routing problems, Vol. New Ideas in Optimization, McGraw-Hill, London, 1999.

[75] B. Hajek: Cooling schedules for optimal annealing, Math. of OR, 13, 1988, pp. 311-329.

[76] H.M. Botee, E. Bonabeau: Evolving ant colony optimization, (SFI Working Paper Abstract), 1999.

[77] Price K.V., Storn R.M., Lampinen J.A.: Differential Evolution: A Practical Approach to Global Optimization, Springer, Berlin, 2005.

[78] Joshi M.C., Moudgalya K.M.: Optimization: Theory and Practice, Alpha Science International, Harrow, 2004.

[79] Reklaitis G.V., Ravindran A., Ragsdell K.M.: Engineering Optimization: Methods and Applications, John Wiley & Sons, Inc., New York, 1983.

[80] Kirkpatrick S., Gelatt C.D., Vecchi M.P.: Optimization by simulated annealing. Science, 220(4598), 1983, pp. 671-680.

[81] Goldberg D.E.: Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.

[82] Schwefel H.P., Corne D., Dorigo M., Glover F. (eds.): Evolution and Optimum Seeking, John Wiley & Sons, Inc., New York. 1999.

[83] Pardalos P.M., Resende M.G.C. (eds.): Handbook of Applied Optimization, Oxford University Press, Oxford, 2002.

[84] Onwubolu G.C., Babu B.V. (eds.): New Optimization Techniques in Engineering, Springer, Berlin, 2004.

[85] Dorigo M.: Optimization, Learning and Natural Algorithms, PhD thesis, Politecnico di Milano, Italy, 1992.

[86] Pham D.T., Ghanbarzadeh A., Koc E. et al.: The bees algorithm – a novel tool for complex optimization problems, in Intelligent Production Machines and Systems: 2nd I_PROMS Virtual Conference (eds. D.T. Pham, E.E. Eldukhri and A.J. Soroka), Elsevier, Oxford, 2006, pp. 454-461.

[87] Babu B.V., Chaturvedi G.: Evolutionary computation strategy for optimization of an alkylation reaction. International Symposium 53rd Annual Session IIChE, Science City, Calcutta, Dec. 2000, pp. 18-21.

[88] Babu B.V. and Munawar S.A.: Differential evolution for the optimal design of heat exchangers. All India Seminar Chemical Engineering Progress R, 2000.

[89] Jozwiak S.F.: Improving structural optimization programs using artificial intelligence concepts, Engineering Optimization 12: 1987, pp. 155-162.

[90] S. Banerjee, L.N. Hazra: Thin lens design of cooke triplet lenses: application of a global optimization technique. In K.D. Bell, M.K. Powers and J.M. Sasian editors, Proceedings of SPIE, Society of Photo-Optical Instrumentation Engineers (SPIE) Conference – Novel Optical Systems and Large-Aperture Imaging, Vol. 3430, 1998, pp. 175-183.

[91] S. Boettcher: Extremal optimization of graph partitioning at the percolation threshold. Journal of Physics A: Mathematical and General, 32(28):5201-5211, July 16, 1999.

[92] S. Boettcher, A.G. Percus: Extremal optimization: Methods derived from co-evolution. In GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conferenc, 1999.

[93] S. Boettcher, A.G. Percus: Extremal optimization for graph partitioning. Physical Review E, 64(2), July 2001.

[94] S. Boettcher, A.G. Percus: Optimization with extremal dynamics. Physical Review Letters, 86(23):5211-5214, June 4, 2001.

[95] J. Branke: Evolutionary Optimization in Dynamic Environments. PhD thesis, Universitat Karlsruhe (TH), Fakultat fur Wirtschaftswissenschaften, Institut AIFB, D-76128 Karlsruhe, 2000.

[96] J. Branke: Evolutionary Optimization in Dynamic Environments, Vol. 3 of Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, Dec. 2001.

[97] A. Torn, A. Zilinskas: Global Optimization, Vol. 350/1989 of Lecture Notes in Computer Science (LNCS). Springer-Verlag, Berlin Heidelberg, New York, 1989.

[98] P.E. Gill, W. Murray, M.A. Saunders, M.H. Wright: Procedures for Optimization Problems with a Mixture of Bounds and General Linear Constraints. ACM Trans. Math. Software, Vol. 10, 1984, pp. 282-298.

[99] A. Messac, C.A. Mattson: Normal constraint method with guarantee of even representation of complete Pareto frontier. AIAA journal, Vol. 42, No. 10, 2004, pp. 2101-2111.

[100] D. Mueller-Gritschneder, H. Graeb, U. Schlichtmann: A Successive Approach to Compute the Bounded Pareto Front of Practical Multiobjective Optimization Problems. SIAM Journal on Optimization, Vol. 20, Issue 2, 2009, pp. 915-934.

[101] T. Erfani, S.V. Utyuzhnikov: Directed Search Domain: A Method for Even Generation of Pareto Frontier in Multiobjective Optimization. Journal of Engineering Optimization, 2010.

[102] Coello C.A., Lamont G.B., Van Veldhuizen D.A.: Evolutionary Algorithms for Solving Multi-Objective Problems Springer, 2007.

[103] Das S., Panigrahi B.K.: Multi-objective E33. Steuer R.E.: Multiple Criteria Optimization: Theory, Computations, and Application. New York: John Wiley & Sons, 1986.

[104] Sawaragi Y., Nakayama H., Tanino T.: Theory of Multiobjective Optimization (Vol. 176 of Mathematics in Science and Engineering). Orlando, FL: Academic Press Inc., 1985.

[105] D. Craft, T. Halabi, H. Shih, T. Bortfeld: Approximating convex Pareto surfaces in multiobjective radiotherapy planning. Medical Physics, 33(9):3399-3407, 2006.

[106] Fleming P.J.: Application of Multiobjective Optimization to Compensator Design for SISO Control Systems. Electronics Letters, Vol. 22, No. 5, 1986, pp. 258-259.

[107] Fleming P.J.: Computer-Aided Control System Design of Regulators using a Multiobjective Optimization Approach. Proc. IFAC Control Applications of Nonlinear Porg. and Optim., Capri, Italy, 1985, pp. 47-52.

[108] N. Padhye, S. Kalia, K. Deb: Multi-Objective Optimization and Multi-Criteria Decision Making For FDM Using Evolutionary Approaches, 2009, Indian Institute of T echnology Kanpur, Kanpur-208016, U.P., India. npdhye@gmail.com, subodh@iitk.ac.in

[109] A. Abraham, L.C. Jain, R. Goldberg editors: Evolutionary Multiobjective Optimization – Theoretical Advances and Applications. Advanced Information and Knowledge Processing. Springer, Berlin, July 30, 2005.

[110] D. Chafekar, Jiang Xuan, K. Rasheed: Constrained multi-objective optimization using steady state genetic algorithms. In Proceedings of Genetic and Evolutionary Computation GECCO 2003.

[111] C.M. Fonseca, P.J. Fleming: Multiobjective optimization and multiple constraint handling with evolutionary algorithms. In Practical Approaches to Multi-Objective Optimization, 2004.

[112] X. Gandibleux, M. Sevaux, K. Sorensen, V. T'kindt editors: Metaheuristics for Multiobjective Optimisation, Volume 535 of Lecture Notes in Economics and Mathematical Systems. Springer, Berlin, January 2004.

[113] C.S. Chang, Chung Min Kwan: Evaluation of evolutionary algorithms for multiobjective train schedule optimization. In AI 2004: Advances in Artificial Intelligence, Vol. 3339/2004 of Lecture Notes in Artificial Intelligence, subseries of Lecture Notes in Computer Science (LNCS), pp. 803-815, Springer-Verlag, 2004.

[114] H. Muhlenbein, D. Schlierkamp-Voosen: Predictive models for the breeder genetic algorithm i: Continuous parameter optimization. Evolutionary Computations, 1(1):25-49, Spring 1993.

[115] R. Poli: Parallel distributed genetic programming. In New Ideas in Optimization, pp. 403-432, McGraw-Hill Education, 1999.

[116] K.V. Price, R.M. Storn, J.A. Lampinen: Differential Evolution – A Practical Approach to Global Optimization. Natural Computing Series. Springer, New York, October 2005.

[117] J.D. Schaffer: Multiple objective optimization with vector evaluated genetic algorithms. In Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications, pp. 93-100, 1985.

[118] H.P. Schwefel: Evolution and Optimum Seeking: The Sixth Generation. John Wiley & Sons, Inc., Wiley Interscience, New York, NY, USA, 1993.

[119] W.W. Siedlecki, J. Sklansky: Constrained genetic optimization via dynamic reward-penalty balancing and its use in pattern recognition. In Proceedings of the third international conference on Genetic algorithms, pp. 141-150, 1989.

[120] W.W. Siedlecki, J. Sklansky: Constrained genetic optimization via dynamic reward-penalty balancing and its use in pattern recognition. In C.H. Chen, L.F. Pau and P.S.P. Wang editors, Handbook of Pattern Recognition and Computer Vision, chapter 1.3.3, pp. 108-124, World Scientific, 1993.

[121] T. Blackwell: Particle swarm optimization in dynamic environments. In Evolutionary Computation in Dynamic and Uncertain Environments, chapter 2, pp. 29-52, Springer, 2007.

[122] T. Back, H.P. Schwefel: An overview of evolutionary algorithms for parameter optimization. Evolutionary Computation, 1(1):1-23, Spring 1993.

[123] A.M. Baltar, D.G. Fontane: A generalized multiobjective particle swarm optimization solver for spreadsheet models: application to water quality. In Proceedings of AGU Hydrology Days 2006, March 20-22, 2006, pp. 1-12.

[124] F.T.S. Chan, M.K. Tiwari editors: Swarm Intelligence – Focus on Ant and Particle Swarm Optimization. I-Tech Education and Publishing, Vienna, Austria, Dec. 2007.

[125] S. Chandran and R.R. Rhinehart: Heuristic random optimizer-version ii. In Proceedings of the American Control Conference, Vol. 4, pp. 2589-2594. IEEE, Piscataway NJ, US, 2002.

[126] Kalyanmoy Deb, Samir Agrawal, Amrit Pratab, T. Meyarivan: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In Proceedings of the Parallel Problem Solving from Nature VI Conference, 2000, pp. 849-858.

[127] V. Cerny: Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of Optimization Theory and Applications, 45(1):41-51, 1985.

[128] Hajime Kita, Yasuhito Sano: Genetic algorithms for optimization of noisy fitness functions and adaptation to changing environments. In 2003 Joint Workshop of Hayashibara Foundation and 2003 Workshop on Statistical Mechanical Approach to Probabilistic Information Processing (SMAPIP), July 14-15, 2003, Okayama, Japan.

[129] D. Montana, J. Redi: Optimizing parameters of a mobile ad hoc network protocol with a genetic algorithm. In GECCO'05: Proceedings of the 2005 conference on Genetic and evolutionary computation, 2005, pp. 1993-1998.

[130] P. Moscato: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P 826, Caltech Concurrent Computation Program 158-79, California Institute of Technology, Pasadena, USA, Pasadena, CA, 1989.

[131] H. Muhlenbein: Parallel genetic algorithms in combinatorial optimization. In Osman Balci, Ramesh Sharda and Stavros A. Zenios editors: Selected papers from the Conference Computer Science and Operations Research: New Developments in Their Interfaces, pp. 441-456, Jan. 8-10, 1992, Williamsburg, Virginia, Pergamon Press.

[132] Samantha Y. Chong, M. Tremayne: Combined optimization using cultural and differential evolution: application to crystal structure solution from powder diffraction data. Chemical Communication, 39:4078-4080, October 2006.

[133] Hosung Choo, A. Hutani, L.C. Trintinalia, Hao Ling: Shape optimisation of broadband microstrip antennas using genetic algorithm. Electronics Letters, 36(25):2057-2058, Dec. 7, 2000.

[134] R. Chiong editor: Nature-Inspired Algorithms for Optimisation, Vol. 193 of Studies in Computational Intelligence. Springer, April 30, 2009.

[135] Yuelin Gao, Yuhong Duan: An adaptive particle swarm optimization algorithm with new random inertia weight. In Advanced Intelligent Computing Theories and Applications. With Aspects of Contemporary Intelligent Computing Techniques, Vol. 2, 2007, pp. 342-350.

[136] Yuelin Gao, Zihui Ren: Adaptive particle swarm optimization algorithm with genetic mutation operation. In Proceedings of the Third International Conference on Natural Computation (ICNC 2007), Vol. 2, 2007, pp. 211-215.

[137] J.H. Holland: Adaptive plans optimal for payoff-only environments. In Proceedings of the Second Hawaii International Conference on System Sciences, Periodicals, pp. 917-920. North Holland, Jan. 22-24, 1969.

[138] D. Holstein, P. Moscato: Memetic algorithms using guided local search: a case study. In New ideas in optimization, McGraw-Hill Ltd., 1999, pp. 235-244.

[139] R.C. Holte: Combinatorial auctions, knapsack problems and hill-climbing search. In Proceedings of the 14th Biennial Conference of the Canadian Society on Computational Studies of Intelligence: Advances in Artificial Intelligence, Vol. 2056/2001 of Lecture Notes in Computer Science (LNCS), pp. 57. Springer Berlin /Heidelberg, June 7-9, 2001, Ottawa, Kanada.

[140] J. Horn, N. Nafpliotis, D.E. Goldberg: A niched Pareto-genetic algorithm for multiobjective optimization. In Proceedings of the First IEEE Conference on Evolutionary Computation, Vol. 1, 1994, pp. 82-87.

[141] Deb K., Pratap A., Agarwal S., Meyarivan T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, Vol. 6, No. 2, 2002, pp. 182-197.

[142] Deb K.: Multi-Objective Optimization using Evolutionary Algorithms. Wiley, 2002.

[143] Matthias John, Max J. Ammann: Optimisation of a wide-band printed monopole antenna using a genetic algorithm. In Loughborough Antennas and Propagation Conference (PAPC), pp. 237-240. Loughborough University, April 11-12, 2006.

[144] P. Korosec, J. Silc: Real-parameter optimization using stigmergy. In Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Application, BIOMA 2006, Oct. 2006, pp. 73-84.

[145] M. Meissner, M. Schmuker, G. Schneider: Optimized particle swarm optimization (opso) and its application to artificial neural network training. BMC Bioinformatics, 7(125), 2006.

[146] E. Mezura-Montes, C.A. Coello Coello: Using the evolution strategies' self-adaptation mechanism and tournament selection for global optimization. Intelligent Engineering Systems through Artificial Neural Networks (ANNIE'2003), 13:373-378, Nov. 2003.

[147] E. Mezura-Montes, J. Velazquez-Reyes, C.A. Coello Coello: A comparative study of differential evolution variants for global optimization. In GECCO'06: Proceedings of the 8th annual conference on Genetic and evolutionary computation, 2006, pp. 485-492.

[148] Z. Michalewicz: A step towards optimal topology of communication networks. In Vibeke Libby, editor, Proceedings of Data Structures and Target Classification, the SPIE's International Symposium on Optical Engineering and Photonics in Aerospace Sensing, Vol. 1470 of SPIE, pp. 112-122. SPIE – The International Society for Optical Engineering, April 1-5, 1991.

[149] Z. Michalewicz: Genetic algorithms, numerical optimization, and constraints. In Proceedings of the Sixth International Conference on Genetic Algorithms, 1995, pp. 151-158.

[150] Z. Michalewicz, G. Nazhiyath: Genocop iii: A co-evolutionary agorithm for numerical optimization with nonlinear constraints. In Proceedings of the Second IEEE International Conference on Evolutionary Computation, Vol. 2, 1995, pp. 647-651.

[151] B.L. Miller, M.J. Shaw: Genetic algorithms with dynamic niche sharing for multimodal function optimization. IlliGAL Report 95010, Department of General Engineering, University of Illinois at Urbana-

Champaign, 117 Transportation Building, 104 South Mathews Avenue, Urbana, IL 61801, USA, Dec. 1, 1995.

[152] G. Ochoa, I. Harvey, H. Buxton: Error thresholds and their relation to optimal mutation rates. In European Conference on Artificial Life, 1999, pp. 54-63.

[153] Guanyu Pan, Quansheng Dou, Xiaohua Liu: Performance of two improved particle swarm optimization in dynamic optimization environments. In ISDA'06: Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06), Vol. 2, pp. 1024-1028, 2006, Jinan, China.

[154] J. Kennedy, R.C. Eberhart: Particle swarm optimization. In Proceedings of IEEE International Conference on Neural Networks, 1995, Vol. 4, pp. 1942-1948, Nov. 27-Dec. 1, 1995, Perth, WA, Australia.

[155] S. Kirkpatrick, C.D. Gelatt Jr., M.P. Vecchi: Optimization by simulated annealing. Science, 220(4598):671-680, May 13, 1983.

[156] J.P.C. Kleijnen: Experimental design for sensitivity analysis, optimization, and validation of simulation models. In Handbook of Simulation, chapter 6, pages 173-223. Wiley & Sons, 1998.

[157] P.M. Pardalos, Ding-Zhu Du: Handbook of Combinatorial Optimization, Vol. 1-3 of Combinatorial Optimization. Kluwer Academic Publishers, Springer Netherlands, October 31, 1998.

[158] K.E. Parsopoulos and M.N. Vrahatis: Recent approaches to global optimization problems through particle swarm optimization. Natural Computing, 1(2-3):235-306, June 2002.

[159] Kwang Mon Sim, Weng Hong Sun: Multiple ant-colony optimization for network routing. In First International Symposium on Cyber Worlds: Theory and Practices (CW'02), pp. 277-281, Nov. 6-8, 2002. IEEE Computer Society, Los Alamitos, CA, USA.

[160] D. Simon: Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches. Wiley-Interscience, June 2006.

[161] M.C. Sinclair: Minimum cost topology optimisation of the cost 239 european optical network. In Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, 1995, pp. 26-29.

[162] Gulshan Singh, K. Deb: Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In GECCO'06: Proceedings of the 8th annual conference on Genetic and evolutionary computation, 2006, pp. 1305-1312.

[163] J.C. Spall: Introduction to Stochastic Search and Optimization. Estimation, Simulation, and Control – Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, first edition, June 2003.

[164] Peter Stagge and Christian Igel: Structure optimization and isomorphisms. In Theoretical Aspects of Evolutionary Computing, pp. 409-422. Springer, 2000.

[165] R.E. Steuer: Multiple Criteria Optimization: Theory, Computation and Application. Krieger Pub Co, reprint edition, August 1989.

[166] R. Storn: On the usage of differential evolution for function optimization. In M. Smith, M. Lee, J. Keller and J. Yen, editors, 1996 Biennial Conference of the North American Fuzzy Information Processing Society, pp. 519-523, 1996. IEEE Press, Piscataway, NJ.

[167] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.P. Chen, Anne Auger, S. Tiwari: Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. KanGAL Report 2005005, Kanpur Genetic Algorithms Laboratory, KanGAL, Indian Institute of Technology Kanpur, India, May 2005.

[168] Tetsuyuki Takahama, Setsuko Sakai: Constrained optimization by applying the α constrained method to the nonlinear simplex method with mutations. IEEE Transactions on Evolutionary Computation, 9(5):437-451, October 3, 2005

[169] Ioan Cristian Trelea: The particle swarm optimization algorithm: convergence analysis and parameter selection. Information Processing Letters, 85(6):317-325, March 31, 2003.

[170] Chao-Hsi Tsao, Jyh-Long Chern: Application of a global optimization process to the design of pickup heads for compact and digital versatile disks. Optical Engineering, 45(10):103001, 2006.

[171] R.K. Ursem: Models for Evolutionary Algorithms and Their Applications in System Identification and Control Optimization. PhD thesis, Department of Computer Science, University of Aarhus, Denmark, 2003.

[172] Frank J. Villegas, Tom Cwik, Yahya Rahmat-Samii, Majid Manteghi: A parallel electromagnetic genetic-algorithm optimization (ego) application for patch antenna design. IEEE Transactions on Antennas and Propagation, 52(9):2424-2435, Sept. 3, 2004.

[173] Fang Wang, Yuhui Qiu: Multimodal function optimizing by a new hybrid nonlinear simplex search and particle swarm algorithm. Proceedings of ECML 2005, the 16th European Conference on Machine Learning, volume 3720, pp. 759-766, Springer, October 3-7, 2005, Porto, Portugal.

[174] Fang Wang, Yuhui Qiu, Yun Bai: A new hybrid nm method and particle swarm algorithm for multimodal function optimization. Advances in Intelligent Data Analysis VI, 6th International Symposium on Intelligent Data Analysis, IDA 2005, volume 3646.

[175] Fang Wang, Yuhui Qiu, Naiqin Feng: Multi-model function optimization by a new hybrid nonlinear simplex search and particle swarm algorithm. In First International Conference on Advances in Natural Computation, Part III, 2005, pp. 562-565.

[176] Zhao-Xia Wang, Zeng-Qiang Chen, Zhu-Zhi Yuan: Qos routing optimization strategy using genetic algorithm in optical fiber communication networks. Journal of Computer Science and Technology, 19(2):213-217, March 2004.

[177] D.C. Wedge, D.B. Kell: Rapid prediction of optimum population size in genetic programming using a novel genotype-fitness correlation. In Genetic and Evolutionary Computation Conference, 2008.

[178] Nelson Wu: Differential evolution for optimisation in dynamic environments. Techncal Report, School of Computer Science and Information Technology, RMIT University, Nov. 2006.

[179] Yong L. Xiao, D.E. Williams: Game: Genetic algorithm for minimization of energy, an interactive program for three-dimensional intermolecular interactions. Computers and Chemistry, 18(2):199-201, 1994.

[180] K. Zielinski, R. Laur: Stopping criteria for a constrained single-objective particle swarm optimization algorithm. Informatica, 31(1): pp. 51-59, 2007.

[181] A. Zilinskas: Optimization of one-dimensional multimodal functions. Applied Statistics, 27(3):367-375, 1978.

[182] Esad Mulavdi: Multi-criteria optimization of construction technology of residential bulding upon the principles of sustainable development, thermal science: Vol. 9, No. 3, 2005, pp. 39-52.

[183] Salomon R.: Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions: a survey of some theoretical and practical aspects of genetic algorithms. Bio Systems, 39(3), 1996, pp. 263-278.

[184] Rajeev Kumar: Improved Sampling of the Pareto-Front in Multiobjective Genetic Optimizations by Steady-State Evolution: A Pareto Converging Genetic Algorithm, rkumar@cse.iitkgp.ernet in Indian Institute of Technology, Kharagpur, 2009.

[185] P. Rockett: Multiobjective Optimization Problems With Complicated Pareto Sets, p.rockett@shef_eld.ac.uk

[186] H. Piech: The compromise in multicriterial net optimization. Poland, hpiech@adm.pcz.czest.pl

[187] Peng Yang, Francky Catthoor: Pareto-Optimization-Based Run-Time Task Scheduling for Embedded Systems. IMEC, Kapeldreef 75, B3001 Leuven, Belgium, yangp, catthoor@imec.be

[188] Censor Y.: Pareto Optimality in Multiobjective Problems. Appl. Math. Optimiz., Vol. 4, 1977, pp. 41-59.

[189] Papalambros P., Wilde D.J.: Principles of Optimal Design: Modeling and Computation. Cambridge University Press, Cambridge. 2000.

[190] Pardalos P.M., Resende M.G.C. (eds.): Handbook of Applied Optimization. Oxford University Press, New York, 2002.

[191] Parmee I.C.: Towards an optimal engineering design process using appropriate adaptive search strategies. Journal of Engineering Design 7(4): 1996, pp. 341-362.

[192] Parmee I.C., Hajela P.: Optimization in Industry. Springer-Verlag, London, 2002.

[193] M. Hamm, U. Beissert, M. Konig: Simulation-based optimization of construction shedules by using Pareto simulated annealing. Bauhaus-University Weima, Germany, E-mail: matthias.hamm@uni-weimar.de

[194] Tae Hee Lee, Kwangki Lee, Kwang Soon Lee: Multi-criteria shape optimization of a finel in cathode ray tubes using response surface model. Gumi, 730-030, Korea, E-mail: thlee@hanyang.ac.kr

[195] Schwabacher M., Ellman T., Hirsh H.: Learning to set up numerical optimizations of engineering designs, Artificial Intelligence for Engineering Design, Analysis and Manufacturing 12: 1998, pp.173-192.

[196] G.A.P. Kindervater, D. Gelatt, M.P. Vecchi: Optimization by simulated annealing. Science 220 (1983), 671, Savelsbergh. M.W.P. Vehicle routing: handling edge exchanges, E.H. Aarts, J.K. Lenstra eds.: Local Search in Combinatorial Optimization. John Wiley & Sons, Chichester, UK. 1997, pp. 311-336.

[197] V. Maniezzo, A. Carbonaro: Ant Colony Optimization: an overview, in C. Ribeiro (eds.) Essays and Surveys in Metaheuristics. Kluwer, 2001, pp. 21-44.

[198] G. Navarro Varela, M. C. Sinclair: Ant Colony Optimization for virtual wavelength -path routing and wavelength allocations. Proceeding of the Congress on Evolutionary Computation (CEC'99), Washington DC, USA, 1999.

[199] R.S. Parpinelli, H.S. Lopes, A.A. Freitas: Data mining with ant colony optimization algorithm. IEEE Trasactions on Evolutionary Computation, Vol. 6, August 2002.

[200] T. Stutzle, M. Dorigo: Aco algorithms for the quadratic assignment problem. New Ideas in Optimization, McGraw-Hill, London, 1999, pp. 3-50.

[201] T. Stutzle, H.H. Hoos: MAX-MIN ant system. Future Generation Computer Systems, Vol. 16, 2000.

[202] E. Siegel, B. Denby, S. Le Hegarat-Mascle: Application of ant colony optimization to adaptive routing in a leo telecommunications satellite network. IEEE Trasactions on Networks, July 2000.

[203] T. Stutzle, H. Hoos: Ant system and local search for combinatorial optimization problems. S. Voss, S. Martello, I.H. Osman and C. Roucairol editors, Vol. Meta-Heuristics: Advanced and Trends in Local Search Paradigms for Optimization, Kluwer, Boston, 1998.

[204] T. Stutzle, H. Hoos: The MAX-MIN Ant System and Local Search for Combinatorial Optimization Problems: Towards Adaptive Tools for Combinatorial Global Optimization. In S. Voss, S. Martello, I.H. Osman and C. Roucairol editors, Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers, 1998, pp. 313-329.

[205] W. Ying, X. Jianying: Ant colony optimization for multicast routing. In the 2000 IEEE Asia-Pacific Conference on Circuits and Systems, 2000, Tianjin, China.

[206] M. Akbar et al.: Heuristic Solutions for the Multiple-Choice Multi-Dimension Knapsack Problem. In Int. Conf. Computational Science, June 2001, pp. 112-117.

[207] T. Givargis, F. Vahid, J. Henkel: System-level Exploration for Pareto-optimal Configurations. In Parameterized System-on-a-Chip, IEEE Trans. VLSI Syst., 10(4):579-592, Aug. 2002.

[208] Jouni Pousi, Kai Virtanen, Raimo P. Hamalainen: Multiple Criteria Optimization and Analysis. In the Planning of Effects-Based Operations (EBO), Helsinki University of Technology, jouni.pousi@hut.fi, kai.virtanen@hut.fi, raimo@hut.fi

[209] Steven M. Small, Benjamin Jeyasurya: Multi-Objective Reactive Power Planning: A Pareto Optimization Approach. The 14th Int. Conference on Intelligent System Applications to Power Systems, ISAP 2007, Nov. 4-8, 2007, Kaohsiung, Taiwan.

[210] M. Chiang: Optimization of Communication Systems, Lecture 16: Pareto Optimization and Nonconvex Optimization. March 16, 2007.

[211] B.L. Wessel: Analitical and numerical optimization of an implantable volume conduction antenna BS. University of Pittsbugh, 2002.

[212] K. Giannakoglou, D. Tsahalis, J. Periaux, K. Papailiou, T. Fogarty (Eds.): Evolutionary methods for design, optimization and control. Barcelona, Spain 2002.

[213] S.C. Liu, C.C. Lin: A heuristic method for the combined location routing and inventory problem, Int. Journ. Adv. Manuf. Technol., 2005, 26, pp. 372-381.

[214] C.F. Olson: A General Method for Geometric Feature Matching and Model Extraction. Int. Journal of Computer Vision 45(1), 2001, pp. 39-54.

[215] H. Tuy: Monotonic optimization: problems and solution approaches. SIAM J. Optim. 11(2), 2000, pp. 464-494.

[216] R.T. Dalimov: Modelling International Economic Integration: an Oscillation Theory Approach. Victoria, Trafford, 2008, 234 p.

[217] L.P. Kuptsov: Gradient, in Encyclopedia of Mathematics edited by M. Hazewinkel, Springer, 2001.

[218] S. Visnovsky: Matrix representations for vector differential operators in general orthogonal coordinates, Czech. J. Phys., 54, 2004, pp. 793-819.

[219] I. Das, J.E. Dennis: Normal-Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems. SIAM Journal on Optimization, 8:631-657, 1998.

[220] A. Chinchuluun, P.M. Pardalos, A. Migdalas, L. Pitsoulis editors: Pareto Optimality. Game Theory and Equilibria, Vol. 17 of Springer Optimization and Its Applications. Springer, New York, USA, 2008.

[221] A. Messac, A. Ismail-Yahaya, C.A. Mattson: The normalized normal constraint method for generating the Pareto frontier. Structural and multidisciplinary optimization, Vol. 25, No. 2, 2003, pp. 86-98.

[222] V.S. Mihalevich, A.I. Kuksa: Metody posledovatelnoj optimizacii. –M., Nauka, 1983, (in Russian).

[223] V.S. Mihalevich, V.L. Volkovich: Vichislitelnyje metody issledovania i projektirovanija sloznyh sistem. –M., Nauka, 1982, (in Russian).

[224] W. Garbarczuk, A. Świć: Podtawy ochrony informacji. 2005, p. 512, (in Polish).

[225] W. Garbarczuk, A. Świć, W. Wójcik: Metodologia ochrony informacji. Lublin, 2006, p. 358, (in Polish).

[226] W. Garbarczuk, A. Świć, W. Wójcik: Projektowanie systemów ochrony informacji. Lublin, 2006, p. 219, (in Polish).

[227] V. Harbarchuk: Matematicheskoje projektirovanije sloznyh sudovyh system. Leningrad, Sudostojenije, 1982, (in Russian).

[228] V. Harbarchuk: Modeli vektornyh igr dla optimizacii dejstvij v sytuacijah 'napadenije-zaschita', in book 'Tehnologija mikroelektronnych priborov i apparatury sredstv svijazi. –M., Nauka, 1988, (in Russian).

[229] P. Mirek, J. Jablonski, W. Nowak: Numerical Optimization of air flow in the plenum chamber of CFB boiler. Czestochowa University of Technology, Poland, 2008, pmirek@neo.pl, wnowak@is.pcz.czest.pl

[230] M. Dorigo: Ant colony optimization. http://iridia.ulb.ac.be/mdorigo/ACO/ACO.html.

[231] Corina Rotar: An Evolutionary Technique for Multicriterial Optimization Based on Endocrine Paradigm. "1 Decembrie 1918" University, Computer Science Department, Alba Iulia, Romania, crotar@uab.ro

[232] Shigeru Obayashi: Multidisciplinary design optimization of aircraft wing planform based on evolutionary algorithms. In Proceedings of the 1998 IEEE International Conference on Systems, Man and Cybernetics. IEEE Press, October 11-14, 1998, La Jolla, California, USA.

[233] P.D. Lezniuk: Metody optimizacii v elektroenrgetyci – Kriterialnyj Metod. Universum, Vinnucia, 2003, (in Ukrainian).

[234] L.V. Kantorovich, A.B. Horstko: Optimalnyje reshenija v ekonomike. –M., Nauka, 1972, (in Russian).

[235] M. Lahanas: Multiobjective optimization. Googl.pl, Multiobjectiveoptimization, 1010.

[236] N. Nedjah, L. de Macedo Mourelle editors: Systems Engineering using Particle Swarm Optimisation. Nova Publishers, February 28, 2007.

Chapter 8

Optimization methods for robot-manipulation systems modeling and controlling

I. Krak, I. Kryvonos, W. Wójcik, P. Komada

8.1 Optimization methods for planning problems solving and computer aided design of optimal kinematics spatial structure of manipulation robots

Let's define a formalism to describe manipulation systems for further reference. Let the two adjacent points of a broken chain that consists of solid non-morphed materials, form or rotational kinematic pair of telescopic type, having number of degrees of freedom is determined by independent movements of a possible joint with respect to another pair. Each kinematic pair is assumed to have a single degree of freedom. To overcome the assumption, additional artificial pairs of zero length are introduced. This way the number degrees of freedom of a system and number of kinematic pairs are equal. The

connected kinematic pairs constitute a kinematic scheme of a manipulation system.

Let's associate a Cartesian coordinate system having a center at the beginning of the link O_i and further call it, by convention, a center of i-th kinematic pair. The axis O_iX_i will be headed towards the center of $(i+1)$-th kinematic pair O_{i+1} [10,15]. Another axis will be a movement- or rotation-axis. The remaining axis will be chosen to make the coordinate system right-handed. Coordinate system of i-th joint is considered to be *connected* or *relative* The coordinate system of the starting (zero-indexed) joint is considered to be *global* coordinate system [18-20].

Then the state of an elemental joint of a manipulation system can be determined by that states of joints that connect it to the starting joint. A state of a joint of a manipulation system is described by some values, in particular location, approach and orientation.

Definition 1. Let vector $s(j) = (s_1, s_2, s_3)^T$ define a state of j-th joint of a manipulation system as corresponding radius-vector $A_j(l_j, 0, 0)$ in global coordinate system.

Definition 2. Let vector $k(j) = (k_1, k_2, k_3)^T$ define an approach to j-th joint of a manipulation system as corresponding vector $e_{j_1} = (1, 0, 0)^T$ in global coordinate system.

Definition 3. Let vector $r(j) = (r_1, r_2, r_3)^T$ define an orientation of j-th joint of a manipulation system as corresponding vector $e_{j_2} = (0, 1, 0)^T$ in global coordinate system.

Definition 4. Let the triple $(s(j), k(j), r(j))$ be called a complete state of j-th joint in the global coordinate system.

Vectors $k(j), r(j)$ should satisfy the following conditions: $\|r(j)\| = 1$, $\|k(j)\| = 1,$, $k^T(j)\, r(j) = k_1r_1 + k_2r_2 + k_3r_3 = 0$.

Typically, modern kinematic systems are constructed out of orthogonal or parallel pairs. They are proven to be effective for various kinematic problems (especially for analytic solutions). They provide a finite generic class of kinematic schemes. Optimal schemes with respect to certain criteria can be found in there. They have analogies in the nature. The state of a descendant joint can be set relatively to the current by two parameters: a variable to

define the shift by the kinematic pair and a constant vector that connects centers of adjacent kinematic pairs. The shift is referred to as a generic coordinate of a manipulation system that reflects the degree of freedom for the pair. The norm of the constant vector that connects adjacent kinematic pairs defines the lengths of the joint of the kinematic scheme.

Therefore, a kinematic scheme can be described using a sequence of biparametric transformations. Let's define a mapping $\xi : \{1, 2, ..., n\} \to \{1, 2, 3, 4\}$, that associates i-th kinematic pair its type using the following rule: $\xi\ (i) = k$, where $k \in \{1, 2, 3\}$ defines the type of rotation along a certain axis: *1* – rotation along O_iX_i*; 2* – rotation along O_iY_i*; 3* – rotation along O_iZ_i connected with the joint of the coordinate system; $\xi\ (i) = 4$ signifies the shifting type of the joint between i-th kinematic pair and $(i + 1) - $ th kinematic pair.

According to the formalism described above, in any given n-joint manipulation system the location of a joint in relation to other joints can be found using the following recursive formulae:

$$\begin{cases} a(i-1,j) = P_i^T(\theta_i)a(i,j) + p(\theta_i, l_{i-1}), \\ a(j,j) = r_j, j = \overline{1,n}, i = \overline{j,1}, \end{cases} \tag{8.1}$$

where $a(i, j)$ is a radius-vector of a point r_j of the$j$-th joint inside i-th coordinate system; $l_0 = 0$; $P_i(\theta) \in \mathrm{P} = \{P^1, P^2, P^3, P^4\}$ is a set of matrices of orthogonal rotation along coordinate axes $P^k, k = \overline{1,3}$ and $P^4 = E$is a matrix of the identity transformation, $p_i(\theta_i, l_{i-1}) = l_{i-1_X} e_{i-1_1} + (1 - \Delta(i))\theta_i b_i = $ $= l_{i-1} + (1 - \Delta(i))\theta_i b_i$, $b_i = \left\{-\tilde{P}_i^T e_{i_1}, -e_{i-1_1}\right\}$ – the first value, if i-th link is movable along axis e_{i_1}, and the second value, if the link is movable along axis e_{i-1_1}, $\tilde{P}_i$- is the constant orthonormal matrix that can be one of the rotation matrices according to the value of a generic coordinate θ, in particular, may correspond to the identity matrix E. Let the length of the i-th link be referred to as l_i. Obviously, $l_i = l_{i_X} e_{i_1}$ in i-th coordinate system, where $l_{i_X} = |O_iO_{i+1}|$.

Let's consider the problem of planning of manipulation systems modes, that arise from the necessity to convert system coordinates from dimensional Euclidean space to the value of generalized coordinates and vice versa. This is due to the fact that the object of study is given in absolute coordinates and control actions are formed according to values of generalized coordinates. Technically it's much easier to get the meaning of generalized coordinates, using transducers to measure, and then, having resolved the direct kinematics problem, to find the realities of manipulation system in an absolute coordinate system. Finding the values of generalized coordinates, which will meet

a certain (specified) mode of manipulation system in an absolute coordinate system (inverse kinematics problem) is far more complex task, which is not always the solution because of inaccessibility of the provisions or such solutions can be a whole set. In order to pose a problem of planning modes we will introduce some definitions. As an initial state we will choose the one where the units of manipulation system are elongated along specified axis (for example, OX) other axis are coordinated in the way to get the right-handed coordinate system. This can be achieved at the corresponding angle turns, introducing additional fictitious kinematic pairs. When grouped by growth from the base, the sequence of generalized coordinate forms the vector of generalized coordinates $\theta = (\theta_1, ..., \theta_n)$. In the future, we will consider that $\theta \in R^n$ where R^n-n-dimensional space, endowed with the standard Euclidean structure. Each such vector corresponds to a particular configuration of manipulation system in three-dimensional space, id est, there is a mapping of $\theta \in \Omega \subseteq R^n$ in $a \in \mathrm{A} \subseteq R^3$.

The problem of finding a total state (s, k, r) of n-th unit of manipulation system for a known vector of generalized coordinates $\theta = (\theta_1, ..., \theta_n)$ will be interpreted as the Cauchy problem [4,10] for the discrete control system, if we regard the number of chain as discrete moment of time, the state of every chain in base coordinate system – as phase state of the system, and generalized coordinates – as control parameters.

We get $s = a_1(0)$, $k = a_2(0)$, $r = a_3(0)$, where $a_1(0)$, $a_2(0)$, $a_3(0)$ – are solutions of the following Cauchy problems:

$$a_1(j) = P_{j+1}^T(\theta_{j+1})\ a_1(j+1) + p(\theta_{j+1}, l_j)\ , \tag{8.2}$$

$$a_2(j) = P_{j+1}^T(\theta_{j+1})\ a_2(j+1)\ , \tag{8.3}$$

$$a_3(j) = P_{j+1}^T(\theta_{j+1})\ a_3(j+1)\,, \quad j = \overline{n-1, 0}\,, \tag{8.4}$$

under initial conditions accordingly: $a_1(n) = l_n\ e_{n_1}$, $a_2(n) = e_{n_1}$, $a_3(n) = e_{n_2}$.

We put the problem of inverse planning of modes (inverse kinematics problem) as problem of optimal control of discrete systems with marginal conditions and under the following performance criterion:

$$F = \arg\min_{\theta \in \Theta} \left\{ \alpha_1 \left\| a_1(0) - s' \right\|^2 + \alpha_2 \left\| a_2(0) - k' \right\|^2 + \alpha_3 \left\| a_3(0) - r' \right\|^2 \right\}, \tag{8.5}$$

where $\Theta = \left\{ \theta_i : \theta_{i\min} \le \theta_i \le \theta_{i\max}, \quad i = \overline{1, n} \right\}$, – the range of legitimate values of generalized coordinates , $\theta_{i\min}, \theta_{i\max}$ – set values; $\alpha_i \ge 0, \quad \alpha_1 +$

$\alpha_2 + \alpha_3 = 1$ – weighting coefficient; $\left(s^{'}, k^{'}, r^{'}\right)$ – set goal state of n-th unit of manipulation system in the system of absolute coordinates.

Due to introduced interpretation of the problem of inverse mode planning as the problem of control theory, we will use principle of the maximum to find vector of generalized coordinates θ^*, which provides minimum of functional with gradient procedure:

$$\theta_i(k) = \Pi_\Theta \left\{ \theta_i(k-1) + \frac{\gamma_k}{\|grad H(\cdot)\|_{\theta_i(k-1)}} \times \right.$$

$$\left. \times \frac{\partial}{\partial \theta_i(k-1)} H\left(a_1(i), a_2(i), a_3(i), \theta_i(k-1), \varphi_1(i-1), \varphi_2(i-1), \varphi_3(i-1), i\right) \right\}$$

where $H(\cdot)$ – Hamiltonian-Pontryagin function that looks like:

$$\begin{aligned} & H(a_1(i), a_2(i), a_3(i), \theta_i(k-1), \varphi_1(i-1), \varphi_2(i-1), \varphi_3(i-1), i) = \\ = \quad & \varphi_1^T(i-1) \left[P_i^T(\theta_i(k-1)) \; a_1(i) + p_i(\theta_i(k-1), l_{i-1}) \right] + \\ + \quad & \varphi_2^T(i-1) \; P_i^T(\theta_i(k-1)) \, a_2(i) + \varphi_3^T(i-1) \, P_i^T(\theta_i(k-1)) \, a_3(i), \end{aligned}$$

and functions $\varphi_j(i), \quad j = \overline{1,3}$, are solutions for the following conjugate systems

$$\varphi_1(i) = P_i(\theta_i(k-1)) \, \varphi_1(i-1), \quad \varphi_1(0) = \alpha_1 \left(s^{'} - a_1(0, \theta(k-1)) \right), \tag{8.6}$$

$$\varphi_3(i) = P_i(\theta_i(k-1)) \, \varphi_3(i-1), \quad \varphi_3(0) = \alpha_3 \left(k^{'} - a_3(0, \theta(k-1)) \right), \tag{8.7}$$

$$\varphi_3(i) = P_i(\theta_i(k-1)) \, \varphi_3(i-1), \quad \varphi_3(0) = \alpha_3 \left(r^{'} - a_3(0, \theta(k-1)) \right), \tag{8.8}$$

$i = \overline{1,n}; k = \overline{1,N}; \theta(0) = \theta, \theta(N) = \theta^*$; where γ_k – iteration step; N – number of iterations, which is set or determined by the conditions of accuracy; Π_Θ – operation of projection onto set Θ:

$$a_1(0, \theta(k-1)),$$

$a_2(0, \theta(k-1)), a_3(0, \theta(k-1))$ – according to solution of Cauchy problems under $\theta(k-1) = (\theta_1(k-1), ..., \theta_n(k-1))$.

Note that the weight factors that are functional, provide an opportunity to manage the sequence of changes of generalized coordinates, providing, thus, a certain 'quality' of trajectory.

It is clear that, when the vector of generalized coordinates θ^* delivers minimum of functional, the value $I = \sum_{i=1}^{n} |\theta_i - \theta_i^*|$ [11], known as traffic volume, takes the minimum value.

Thus, the interpretation of the problem of inverse planning of modes as the problem of optimal control of discrete systems and application of Hamiltonian-Potryagin function allowed to find gradient of functional with a number of arithmetic operations which linearly depends on the number of generalized coordinates of manipulation system and this is important for solving these problems in real-time mode.

Simplicity and clarity of description and minimum of necessary parameters for constructing models should be referred to the advantages of specific formalism. In the sequel, this simplified representation will allow to set problems on studies of certain properties of manipulation system using only necessary parameters to solve this problem. For example, for analysis and optimization of attainability domains of manipulation system modes, we used only those parameters that characterize the length of chains and types of kinematic connections; for construction of dynamic equations we used knowledge about the types of connections and dynamic characteristics of chains, such as weight, about static and dynamic moments of weight distribution and so forth.

Using above mentioned formalism and methods of solving mode planning problems, let's put the problem of determining the optimal geometrical parameters of the manipulator. In designing, the most important is the choice of kinematic patterns and geometrical dimensions of the manipulator chains, because they define the principal possibility of robotic application in specific production. Therefore, the major requirement for the manipulator can be formulated as follows: its geometry (number of chains, the type of kinematic pairs, the length of parts) must be necessary and sufficient to perform operations on this set. For mathematical statement let's use correlation for the state of manipulation system, which we will overwrite the following way:

$$a\,(i-1) = \sum_{j=1}^{4} v_j(i) P_j^T\,(\theta_i)\, a\,(i) + \sum_{k=1}^{s} l_k u_k(i) i_1 + \sum_{j=1}^{4} v_j(i) \sigma\,(\theta_i)\;\; i_1, \quad (8.9)$$

$$b\,(i-1) = \sum_{j=1}^{4} v_j(i) P_j^T\,(\theta_i)\, b\,(i)\,,\; c\,(i-1) = \sum_{j=1}^{4} v_j(i) P_j^T\,(\theta_i)\, c\,(i)\,, \quad (8.10)$$

where $v_j(i) \geq 0, \sum_{j=1}^{4} v_j(i) = 1$; $u_k(i) \geq 0, \sum_{k=1}^{s} v_k(i) = 1$; $\sigma = 0, j = 1, 2, 3$; $\sigma = -\theta,\; j = 4$.

Assume that chain lengths belong to the set $L = \{l : l_1, ..., l_N\}$ where $s \leq N$ – predetermined number.

This representation allows to define the problems of designing of optimal manipulator geometry, as changing the parameters $u_k(i), v_j(i)$ you can build various schemes of manipulators.

In general, the task of determining the optimal geometrical characteristics of the manipulator, that maximizes the size of working area in given directions and always gets in a given set of points of location with given approach and orientation vectors, we will express in the following way: to define

$$\max_{n,v,u,\theta} \sum_{i=1}^{6} I_i, \tag{8.11}$$

where $I_1 = \eta^T a(0,\theta)$, $I_2 = \mu^T b(0,\theta)$, $I_3 = \xi^T c(0,\theta)$, $I_4 = -\left\|a_1 - a(0,\theta)\right\|^2$, $I_5 = -\left\|b_1 - b(0,\theta)\right\|^2$, $I_6 = -\left\|c_1 - c(0,\theta)\right\|^2$.

Here η, μ, ξ – vectors from given sets according to vectors of position, approach and orientation of manipulator, a_1, b_1, c_1 – fixed position, approach and orientation vectors from appropriate sets in which the projectable spatial manipulator must necessarily enter.

New mathematical methods, based on methods of control theory, were developed in order to solve the problem of optimal design of manipulator with desired characteristics.

Usage of methods developed for design and research of workspace of manipulation robots, experiments on concrete manipulators, showed their efficiency and effectiveness for tasks solution of the given level.

8.2 Numerical methods and algorithms of automatic construction of dynamic models for the manipulated robots

In various problems of optimal control of complex manipulation systems it is necessary to make an optimal choice (in a certain sense) between contradictive demands, such as adequacy of the mathematical model to the object

of control and the necessity to compute the controlling actions based on this model in the real time. The latter problem, that is, the problem of creation mathematical models for manipulative systems in real time is found to be very complex and hard [5,6,9,16-20,25,30,40]. Therefore, there is a need to use mathematical models of various complexity according to the kind of the requirements of technical objectives, the robustness of computers and other factors. Therefore, in order to conduct an effective control it is necessary to have a set of mathematical models which vary in complexity and adequacy. Depending on the concrete application, different mathematical models should be used.

One of the primary problems in constructing mathematical models for manipulative systems is constructing a mathematical model of a dynamic system or creation of equations for dynamic systems, so-called dynamic models. The dynamic model allows to set dependencies between a given function of movement in time, that is, functions of coordinate change in time, their first and second derivatives and the controlling function, that is the actions to be taken. In general case, a dynamic model can be represented as follows:

$$u = P(\theta, \dot{\theta}, \ddot{\theta}, \xi), \tag{8.12}$$

where $u = (u_1, \ldots, u_n)^T$ are generic controlling powers; $P(\cdot)$ is a non-linear vector-function of dimension n, that depends on generic coordinates $\theta = (\theta_1, \ldots, \theta_n)^T$, their first and second derivatives, and also on weight-impulse parameters of a manipulator ξ; n is a degree of freedom of the system, the exact number of joints in the manipulative system too.

In manipulative system, the kinematics joints of rotational kind have force impulses as the generic controlling function whereas the kinematic joints of telescopic (movement) kind have plain forces as the generic controlling function. Due to the fact that the kinematic joints under consideration consist of combinations of primitive controlling joints, the rank of the vectors of generalized coordinates correspond to each other, moreover, the generic coordinate θ_i corresponds to the generic force u_i.

Without specification of the vector-function $P(\cdot)$ the equations of dynamics of kind (8.12) allow to solve the reverse problem of dynamics only, that is, allow to determine the generic controlling forces necessary to make the needed movement. But this generic representation is not sufficient for implementation of a considerable amount of ways of controlling, that's why there are additional forms of representation of equations for dynamic manipulative systems, which have explicitly present first and second derivatives of the

generic coordinates:

$$u = H(\theta, \xi)\ddot{\theta} + h(\theta, \dot{\theta}, \xi), \tag{8.13}$$

$$u = H(\theta, \xi)\ddot{\theta} + \dot{\theta}^T Q(\theta, \xi)\dot{\theta} + g(\theta, \xi), \tag{8.14}$$

where $H(\theta, \xi)$ – a matrix of rank $n \times n$, $Q(\theta, \xi)$ – a matrix of rank $n \times n \times n$, $g(\theta, \xi)$ – is a vector of rank $n \times 1$.

Models that are defined as (8.13),(8.14) can allow to solve both, diverse and direct dynamic problems – id est, finding a path of motion according to defined by the laws of change control actions, that is more universal. Dynamic model of form (8.14) is the most universal, because it explicitly allocates all dynamic components of manipulation system – matrix of inertia, centrifugal force and coriolis forces, gravitational forces.

Knowledge of the physical meaning of elements of the equations of motion allows more efficient use of these equations or their separate parts to develop new approaches and methods of control such systems. Therefore, further studies will be aimed at creating of dynamic models in the form (8.14).

An important criterion when comparing methods of constructing models of appearance (8.12)-(8.14) is the number of arithmetic operations of multiplication (division), adding (subtracting) with floating point that are necessary for the generation algorithm. This criterion is most significant in terms of applying of the appropriate method for the problems of analysis and synthesis of control systems in real-hand time.

One of the first results of mathematical models of manipulation systems that were used for the solution of both, direct and diverse dynamic problems, was the one received by R. Paul [18,41] with the method of Lagrange. It was developed several algorithms, the common property of which are description of models in closed form without using of the recurrent relations. The number of operations of multiplication/addition in these methods is of order $O(n^4)$, which makes it impossible to use them in real time.

The methods for modeling hinge-connected solid materials that are based on equations of Newton-Euler and take into account the principle of Dalambert [3,5-9,12,16-20], also have the algorithmic implementation that allows to solve the direct as well as reversed problems of dynamics by using expressions of elements of the matrices in a closed form. But, similarly to the previous method, the amount computations for solving the problem is huge. Using the recursive equations allows to create more effective computational algorithms

of forming the equations of dynamics. The algorithmic approximation of this algorithm is $O(n^3)$.

Let's represent the obtained recursive formulae for the equations of dynamics of a manipulative system using the form (8.12). Let's assume the following: a) the joints of the system are uniformly solid materials with weights m_i and matrices of joints inertia relatively to the main axes of inertia J_i; b) axes of the coordinate system for a joint correspond to the main axes of intertia; c) vectors $\tilde{e}_i, \tilde{r}_{ii}, \tilde{r}_{ii+1}$ are set relatively to the joint's coordinate system (let's use the notion of $\tilde{}$ for specifying the relevance to the joint's coordinate system hereafter).

Theorem. The dynamic model of form (8.12) is set as follows in case all the values are relative the corresponding joint's coordinate systems:

$$u_i = (\tilde{Q}_i,\ \tilde{e}_i)\Delta_i + (\tilde{R}_i, \tilde{e}_i)(1-\Delta_i),$$
$$\tilde{R}_i = \tilde{R}_{i+1} - \tilde{G}_i - \tilde{F}_i,$$
$$\tilde{Q}_i = \tilde{Q}^{(i)}_{i+1} + \tilde{r}_i \times \tilde{R}_i - \tilde{r}_{ii+1} \times \tilde{R}^{(i)}_{i+1} - \tilde{M}_i,$$

where:

$$\tilde{M}_i = -J_i\tilde{\varepsilon}_i + \tilde{\omega}_i \times J_i\tilde{\omega}_i,$$
$$\tilde{F}_i = -m_i\tilde{w}_i,$$
$$\tilde{\varepsilon}_i = B_i^T\varepsilon_i, \quad \tilde{\omega}_i = B_i^T\omega_i, \quad \tilde{w}_i = B_i^T w_i,$$
$$B_i = B_{i-1}P_i^T(\theta_i),\ B_0 = E.$$
$$\tilde{G}_i = B_i^T G_i, \quad \tilde{R}^{(i)}_{i+1} = P^T_{i+1}(\theta_{i+1})\tilde{R}_{i+1}, \quad \tilde{Q}^{(i)}_{i+1} = P^T_{i+1}(\theta_{i+1})\tilde{Q}_{i+1},$$
$$\omega_i = \omega_{i-1} + e_i\dot{\theta}_i\Delta_i,$$
$$\varepsilon_i = \varepsilon_{i-1} + (e_i\ddot{\theta}_i + \omega_{i-1} \times e_i\dot{\theta}_i)\Delta_i,$$
$$w_i = w_{i-1} - \varepsilon_{i-1} \times r_{i-1i} - \omega_{i-1} \times (\omega_{i-1} \times r_{i-1i}) +$$
$$+\varepsilon_i \times (r_{ii} + e_i\theta_i(1-\Delta_i)) + (e_i\ddot{\theta}_i + 2\omega_{i-1} \times e_i\dot{\theta}_i)(1-\Delta_i),$$
$$\tilde{r}_i = \tilde{r}_{ii} + \tilde{e}_i\theta_i, \quad e_i = B_i\tilde{e}_i, \quad r_{ii+1} = B_i\tilde{r}_{ii+1},$$

$\tilde{R}_i, \tilde{Q}_i, \tilde{R}_{i+1}, \tilde{Q}_{i+1}$– vectors of reaction of connection on the 'left' and 'right' ends of a joint respectively, $\tilde{G}_i$ – vector of the gravitation.

Theorem. The dynamic model of form (8.12) is set as follows in case all the values are relative the primary coordinate system:

$$u_i = (Q_i,\ e_i)\Delta_i + (R_i,\ e_i)(1-\Delta_i),$$

$$R_i = R_{i+1} - G_i - F_i,$$
$$Q_i = Q_{i+1} + r_i \times R_i - r_{ii+1} \times R_{_{i+1}} - M_i.$$

$$F_i = -m_i w_i,$$
$$M_i = -J_i \varepsilon_i + \omega_i \times J_i \omega_1.$$

It's worth to notice that the amount of computing operations necessary to construct equations of dynamics in form (8.12) is $O(n)$, that is in linear dependence from the number of degrees of freedom of the manipulative system as well as it does not depend on coordinate system in which the model is formed.

Next follows the formulae for construction of equations of dynamics in form (8.13), that is, with explicit matrix of joint's inertia.

Theorem. A dynamic model in form (8.13) is set up by the following equations in case all the values are defined in the global coordinate system.

$$H_{ij} = (e_i, \varphi_{ij}), \quad h_i = (e_i, \phi_i), \quad i, j = \overline{1, n},$$
$$\varphi_{ij} = \mu_{ij}\Delta_i + \eta_{ij}(1 - \Delta_i),$$
$$\phi_i = g_i\Delta_i + p_i(1 - \Delta_i);$$
$$\varphi_{ij} = \mu_{ij}\Delta_i + \eta_{ij}(1 - \Delta_i)$$
$$\eta_{ij} = \eta_{i+1j} - b_{ij}, \quad \eta_{n+1j} = 0, b_{ij} = 0, i > j;$$
$$\mu_{ij} = \mu_{i+1j} + (r_{ii} - r_{ii+1}) \times \eta_{ij} - r_{ii+1} \times b_{ij} - a_{ij},$$
$$\mu_{n+1j} = 0, b_{ij} = a_{ij} = 0, i > j;$$
$$p_i = p_{i+1} - b_i - G_i, \quad p_{n+1} = R_{n+1};$$
$$g_i = g_{i+1} + (r_{ii} - r_{ii+1}) \times p_i - r_{ii+1} \times (b_i + G_i) - a_i,$$
$$g_{n+1} = Q_{n+1} + (r_{nn} - r_{nn+1}) \times R_{n+1};$$
$$b_{ij} = -m_i\beta_{ij},$$
$$\beta_{ij} = \beta_{i-1j} + \alpha_{i-1j} \times (r_{ii} - r_{i-1i}), i > j,$$
$$\beta_{ii} = e_i \times r_i\Delta_i + e_i(1 - \Delta_i),$$
$$a_{ij} = -B_i J_i B_i^T \alpha_{ij},$$
$$\alpha_{ij} = \beta_{i-1j}, \quad \alpha_{ii} = e_i\Delta_i, \quad b_i = -m_i\beta_i,$$
$$\beta_i = \beta_{i-1} + \alpha_{i-1} \times (r_i - r_{i-1i}) - \omega_{i-1} \times (\omega_{i-1} \times r_{i-1i}) +$$
$$+\omega_i \times (\omega_i \times r_i) + \dot{\theta}_i(\omega_{i-1} \times e_i) \times r_i\Delta_i + 2\dot{\theta}_i(\omega_{i-1} \times e_i)(1 - \Delta_i),$$
$$a_i = -B_i J_i B_i^T \alpha_i + \omega_i \times B_i J_i B_i^T \omega_i,$$
$$\alpha_i = \alpha_{i-1} + \dot{\theta}_i(\omega_{i-1} \times e_i)\Delta_i.$$

The same results can be obtained for forming equations of dynamics in case all the values are set in relative coordinate systems. With regard to computing complexity, usage of global coordinate system is proved to be more effective.

Therefore, using the recursive nature of the connections between the joints, the mathematical models of manipulative systems have been obtained and require $O(n^2)$ operations in terms of computational complexity.

To conclude, new numerical methods and algorithms have been developed which allowed to reduce the number of computational operations significantly for the problem of automatic creation of dynamic models of manipulative robots using recursive equations.

8.3 Manipulation of dynamics equations with minimum calculable complication

The problem of mathematical dynamics models construction with the help of robots manipulations was solved with minimum calculable complication. New numerical-analytical approach is developed for forming equalizations of dynamics in a kind with the obviously selected constituents' forces which operate on the system. It is proved, that at such approach every kinematics chart of the manipulation system will have own calculable complication and this complication will be minimum. The problems of planning and analysis of kinematics schemes which can be practically realized, and also influency of every constituent part on equalizations of dynamics, are considered on calculable complication of all system is explored.

Dynamic equations of manipulation system with the n degrees of freedom, which are coincide with vector of the generalized co-ordinates $\theta = (\theta_1, \ldots, \theta_n)^T$, we will form in a kind (8.14) $u = H(\theta, \xi)\ddot{\theta} + \dot{\theta}^T Q(\theta, \xi)\dot{\theta} + g(\theta, \xi)$, where $H(\theta, \xi)$ is a matrix of dimension for $n \times n$, what represents inertial forces of the rings; $Q(\theta, \xi)$=$(Q^1(\theta, \xi), \ldots, Q^n(\theta, \xi))^T$, $Q^i(\theta, \xi)$, $i = \overline{1, n}$ – $n \times n$ matrices of centrifugal forces and coriolis forces; $g(\theta, \xi)$ is a n-vector of gravity forces; u- n-vector of the generalized managing forces; ξ are mass and momentum parameters of links. For the construction of dynamical equation of the manipulation systems we will use Lagrange equation II type for the systems with the n degrees of mobility:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_J}\right) - \frac{\partial L}{\partial \theta_j} = u_j, j = \overline{1, n}, \tag{8.15}$$

where L=(T-P) – is Lagrangian; T – kinetic energy; P – is potential energy of the system; u_j- leading force, which is added to the j-link of the system.

Consequently, for the construction of motion equation it is necessary to define and explore power descriptions of the system. Here out, to receive first two members of dynamics equation in form (8.14) it is needed to find kinetic energy of the system, and for the vector of gravitation forces – potential.

Let us formulate new result, on the basis of which algorithms will be developed for forming motion equation of the manipulations systems with minimum complication.

Theorem. Let the manipulation system from n turning degrees of mobility are know the matrices of rotation about set axes and vectors of displacements and system's rings are undeformed solids with the known density of division of the masses. Then for complete kinetic energy T of the manipulation system is true such presentation:

$$T = \sum_{j=1}^{n} \sum_{i=1}^{j} tr F_i^T F_j K_{ij}, \tag{8.16}$$

where:

$$\begin{gathered} F_k = \textstyle\sum_{i=1}^{k} F_{ki}\dot{\theta}_i = \\ = \textstyle\sum_{i=1}^{k} P_1^T(\theta_1) \cdot ... \cdot P_{i-1}^T(\theta_{i-1}) \cdot \frac{\partial}{\partial \theta_i} P_i^T(\theta_i) \cdot P_{i+1}^T(\theta_{i+1}) ... \cdot P_k^T(\theta_k)\dot{\theta}_i, \end{gathered} \tag{8.17}$$

$$K_{ij} = \begin{cases} \frac{1}{2}(I_i + p_i p_i^T \sum_{k=j+1}^{n} m_k), i = j, \\ M_j p_i^T + p_j p_i^T \sum_{k=j+1}^{n} m_k, i < j. \end{cases}, \tag{8.18}$$

I, M, m – accordingly dynamic and static moments and mass of i-link, $p_i = p_i(\theta_i, l_{i-1})$.

Theorem. Let the terms of previous theorem for this theorem and manipulation system which is executed has as turning, so forward (telescopic) degrees of mobility. Then complete kinetic energy of such manipulation system can be represented in such way as:

$$\begin{gathered} T = \textstyle\sum_{i=1}^{n} \sum_{j=1}^{i} tr F_i^T F_j K_{ij} + \sum_{i=1}^{n} \sum_{j=1}^{n} tr \tilde{F}_i^T F_j K'_{ij} + \\ + \textstyle\sum_{i=1}^{n} \sum_{j=1}^{n} tr \tilde{F}_i^T \tilde{F}_j K''_{ij}, \end{gathered}$$

where:

$$K'_{ij} = (1 - \Delta_j) \begin{cases} b_j l_j^T \sum_{r=j}^{n} m_r, i < j, \\ b_j M_i^T + b_i l_j^T \sum_{r=i+1}^{n} m_r, i \geq j. \end{cases}$$

$$K''_{ij} = \frac{1}{2}(1-\Delta_j)(1-\Delta_i) b_j b_i^T \sum_{r=\max(i,j)}^{n} m_r.$$

$$\tilde{F}_k = \sum_{i=1}^{k-1} \left(P_1^T(\theta_1) \cdot ... \cdot P_{i-1}^T(\theta_{i-1}) \frac{\partial}{\partial \theta_i} P_i^T(\theta_i) P_{i+1}^T(\theta_{i+1}) \cdot ... \cdot P_k^T(\theta_k) \theta_k \right) \dot{\theta}_i + \\ + P_1^T(\theta_1) \cdot ... \cdot P_{k-1}^T(\theta_{k-1}) \dot{\theta}_k = \sum_{i=1}^{k} \tilde{F}_{ki} \dot{\theta}_i.$$

Let's calculate potential energy of manipulation system links.

Theorem. Let the manipulation system will be set with the turning degrees of freedom. Then complete potential energy of such system is calculated on a formula:

$$\Pi = \sum_{k=1}^{n} tr\, A(0,k)\, L_k, \tag{8.19}$$

where:

$$L_k = -g\, (M_k + p_{k+1} \sum_{i=k+1}^{n} m_i)\, e_{i1}^T, \tag{8.20}$$

where g – acceleration of the free falling.

Thus, the proved theorems enable to get new presentation of kinetic and potential energies of the manipulation system, in which constituents, depended upon the mass-and moments' parameters of rings and expressions, depended upon the generalized co-ordinates, are selected. First factors for each of the manipulation systems can be calculated preliminary, as they depend on the certain known or calculated parameters of the system, memorized and in future could be used as permanent sizes. It allows considerably decreasing the amount of arithmetic operations at direct forming of equalizations of dynamics.

Presentations of kinetic and potential energies of the manipulation system got enable to take advantage of dividing method into permanent and dependent parts upon generalized co-ordinates in expressions for finding elements of matrices of inertial, centrifugal, coriolis's and forces of gravitations.

Theorem. Let's get the sated n – dimensional manipulation system with turning types of kinematics' pairs. Then, for the $h_{ij}, i, j = \overline{1,n}$ elements of inertia $H(\theta, \xi)$ matrix the dynamics equation (8.14) will be true following

presentation:

$$h_{km} = \sum_{j=m}^{n} \sum_{i=k}^{j} tr(F_{ik}^T F_{jm} + F_{im}^T F_{jk}) K_{ij}, \tag{8.21}$$

where $F_{ik} = P_1^T(\theta_1) \cdot ... \cdot P_{k-1}^T(\theta_{k-1}) \cdot \frac{\partial}{\partial \theta_k} P_k^T(\theta_k) \cdot P_{k+1}^T(\theta_{k+1}) ... \cdot P_i^T(\theta_i)$.

From Lagrange equation, taking into account representing model (8.14), we will received, that $\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} = \frac{d}{dt}(H(\theta,\xi)\dot{\theta}) = H(\theta,\xi)\ddot{\theta} + \frac{dH(\theta,\xi)}{dt}\dot{\theta}$.

Consequently, for finding matrices of centrifugal and coriolis forces $Q(\theta,\xi)$, it is necessary to calculate $\frac{dH(\theta,\xi)}{dt}\dot{\theta} - \frac{\partial T}{\partial \theta}$.

As $H(\theta,\xi)$ matrix is symmetric and its overhead three-cornered part is calculated only, $Q^i(\theta,\xi)$matrices are also more comfortable to depict in a overhead three-cornered manner, setting elements of such matrix correlations:

$$q_{kl}^i = \begin{cases} q_{kl}^i + q_{lk}^i, k < l, \\ q_{kk}^i, k = l, k, l = \overline{1,n}. \end{cases} \tag{8.22}$$

Theorem. Let the n dimension manipulation system be set with the turning types of kinematics pairs. Then $q_{k\_l}^i, k, l = \overline{1,n}$ elements of $Q^i(\theta,\xi)$, $i = \overline{1,n}$ matrices, which characterize centrifugal and coriolis forces in dynamic equations have the following presentation:

$$q_{kl}^i = \Delta_{kl} \sum_{j=\max(i,k,l)}^{n} \sum_{m=\min(i,k,l)}^{j} tr(F_{mi}^T F_{jkl} + F_{mkl}^T F_{ji}) K_{mj},$$

where $F_{ikl} = P_1^T(\theta_1) \cdot ... \cdot \frac{\partial}{\partial \theta_l} P_l^T(\theta_l) \cdot ... \cdot \frac{\partial}{\partial \theta_k} P_k^T(\theta_k) \cdot ... \cdot P_i^T(\theta_i)$, $\Delta_{kl} = \begin{cases} 1, k = l, \\ 2, k < l \end{cases}$.

Notice that the result of increase of $\dot{\theta}^T Q^i(\theta,\xi)\dot{\theta}$ on the $Q^i(\theta,\xi)$ matrix's diagonal elements will be centrifugal force of i-link of manipulator of $\sum_{k=1}^{n} q_{kk}^i \dot{\theta}_k^2$, and on strictly superdiagonal (remember, that the $Q^i(\theta,\xi)$ matrix has a overhead three-cornered structure) is coriolis force of this link.

Theorem. Gravitation forces, which operate on the manipulation system with turning kinematics pairs are determined after next formulas:

$$g_i = \sum_{k=i}^{n} F_{ki} L_{ki}, \ i = \overline{1,n}\,, \tag{8.23}$$

where:

$$L_{ki} = -g\,(M_k + p_{k+1}\sum_{i=k+1}^{n} m_i)\,e_{i_1}^T\,. \tag{8.24}$$

Thus, for the motion equations elements of formulas, in which pointed part is dependent upon generalized co-ordinates and part, which is dependent upon links' mass and moment parameters of the manipulation system, are got. As was shown previous, the second part can be calculated beforehand, while the first part needs to be calculated at every new values of the generalized co-ordinates.Consequently, a problem was taken to research of trigonometric functions products properties and effective algorithms of their calculations construction.

Presentation of dynamics equations elements on the basis of division method, leads, in essence, a task to research and construction in the obvious type of $F_{kj}^T F_{mi}$ $i,j = \overline{1,n}$; $i = \overline{1,k}$; $j = \overline{1,m}$. matrices' products . $F_{kj} = P_1^T \cdot ... \cdot P_{j-1}^T \cdot \dot{P}_j^T \cdot P_{j+1}^T \cdot ... \cdot P_k^T$, $F_{kj}^T = P_k \cdot ... \cdot \dot{P}_j \cdot ... \cdot P_1$, where $P_i = P_i(\theta_i), i = \overline{1,n}$ – are matrices of turns in relation to the axes of kinematics pairs, $P_i \in \left\{P^l, l = \overline{1,3}\right\}$ are orthogonal matrices of turn round proper axis, $\dot{P}_i = \frac{\partial P_i(\theta_i)}{\partial \theta_i}, i = \overline{1,n},$, so for finding products of image will get certain properties of orthogonal matrices turns .

Theorem. Elements of matrices $F_{kj}^T F_{mi}$ products, are the functions of the generalized co-ordinates, beginning from the number l+1, where $l = \min(i,j)$, and do not depend on the generalized co-ordinates θ_1,...,θ_l:

$$F_{kj}^T F_{mi} = \Phi\left(\theta_{l+1}, \theta_{l+2}, \ldots, \theta_{\max(k,m)}\right).$$

Total matrices products of type $F_{kj}^T F_{mi}$, necessary for calculation one of elements h_{ij}, will be depicted as such charts, stated below.

Diagonal elements:

$$h_{ii}, i = \overline{1,n} \quad F_{ki}^T F_{mi},, k = \overline{i,n}, m = \overline{i,k};$$

$$\begin{array}{ccccc}
F_{ii}^T F_{ii} & & & & \\
F_{i+1i}^T F_{ii} & F_{i+1i}^T F_{i+1i} & & & \\
F_{i+2i}^T F_{ii} & F_{i+2i}^T F_{i+1i} & F_{i+2i}^T F_{i+2i} & & \\
\vdots & \vdots & \vdots & \ddots & \\
F_{ni}^T F_{ii} & F_{ni}^T F_{i+1i} & F_{ni}^T F_{i+2i} & \cdots & F_{ni}^T F_{ni}
\end{array}$$

Off-Diagonal elements:

$$h_{ij}, j = \overline{2,n}, i = \overline{1,j-1} \quad F_{kj}^T F_{mi,}, k = \overline{j,n}, m = \overline{i,n};$$

$$\begin{array}{cccc} F_{jj}^T F_{ii} & F_{jj}^T F_{i+1i} & F_{jj}^T F_{i+2i} & \cdots & F_{jj}^T F_{ni} \\ F_{j+1j}^T F_{ii} & F_{j+1j}^T F_{i+1i} & F_{j+1j}^T F_{i+2i} & \cdots & F_{j+1j}^T F_{ni} \\ F_{j+2j}^T F_{ii} & F_{j+2j}^T F_{i+1i} & F_{j+2j}^T F_{i+2i} & \cdots & F_{j+2j}^T F_{ni} \\ \vdots & \vdots & \vdots & & \vdots \\ F_{nj}^T F_{ii} & F_{nj}^T F_{ii} & F_{nj}^T F_{i+i} & \cdots & F_{nj}^T F_{ni} \end{array}$$

Total number of products of the kind $F_{kj}^T F_{mi}$, which are necessary to calculate all elements inertia matrix $H(\theta, \xi)$ of an arbitrary n-link manipulation system equals $S = \sum_{k=1}^{n} k\frac{k^2+1}{2}$.

Let designate the s – $(s = 1, 2, 3)$ vector-column of matrix F_{kj}^Tthrough f_{kj}^s. Then the elements of matrix $F_{kj}^T F_{mi}$ can be given as an increase of vectors or as an operation of scalar products: $f_{kj}^{sT} f_{mi}^q = (f_{kj}^s, f_{mi}^q)$. Consequently, our task was taken to finding scalar products of these vectors. Offered procedure of scalar products determination consists of 4 stages. On the first stage, by analyzing types of kinematics pairs, we determine scalar products, which are equal to zero or unit. On the second stage, we find products of turns matrices in an analytical kind and, by certain rule, choose one member of every matrices. On the third stage, we carry out differentiation of this member according to generalized co-ordinates. And on the fourth stage, by analyzing received expressions, we find the value of scalar products.

Let's proof another property of motion equations elements, which will allow building radically new algorithms of forming property of motion equations elements. As manipulation system is described by the sequence of links, bound by pairs of V class, by selecting the certain continuous chain of this sequence also we will get, remaining within the formal descriptions framework, manipulation system, only with less dimension. Then there is a task: is it possible to take advantage dynamics equations elements of the systems of fewer dimensions for forming of dynamics equations of the initial (base) manipulation system? We will show that such property takes place and will point, based on this approach, effective algorithm of forming of dynamics equations.

We will show that matrices products $F_{kj}^T F_{mi}$ for the manipulations systems with fewer dimensions, abstracted from base system, are fully included in dynamics equations elements of the initial manipulation system.

We will define: $MR(1,n)=\left\{F_{kj}^T\, F_{mi},\ k,m=\overline{1,n},\ i=\overline{1,m},\ j=\overline{1,k}\right\}$ – range of matrices products, $F_{kj}^T F_{mi},\ i,j=\overline{1,n},\ k=\overline{1,i},\ m=\overline{1,j}$ for the n-dimension initial manipulation system:

$$MR(q,r)=\left\{F_{kj}^T\, F_{mi},\ k,m=\overline{q,r},\ i\le m,\ j\le k\right\}$$

– plural of trigonometric functions matrices products for the manipulation system of dimension of $r-q+1$ *(*$q\le r$, $1\le r\le n$*)*, what is by itself an indissoluble chain of base manipulation system from q – to the r – links.

Theorem. Let the manipulation system of dimension $r-q+1$ is set, so that its kinematics chart coincides with the chart of abstracted from the initial n-dimension system, and the generalized co-ordinates are evened:

$$\theta_1=\theta_q,\theta_2=\theta_{q+1},\ldots,\theta_{r-q+1}=\theta_r.$$

Then matrices products plurals of trigonometric functions of these systems of manipulations coincide, that $MR(1,r-q+1)=MR(q,r)$.

Theorem. Let – a set of products of matrices of trigonometric manipulation systems dimension respectively $r-q+1$ and $p-t+1$. Consider $p\le r\le t$. Then $MR(q,r)\cap MR(p,t)=MR(p,r)$.

These received properties allow to solve another important problem for its practical application, namely: if for a given kinematic scheme to build a set of manipulation, ie, find all the works are matrices of trigonometric functions of generalized coordinates, necessary for forming the equations of motion, and in this scheme to replace the one-level (degree of freedom) to another or you can use it and what elements of a set of motion for the construction of a new manipulation system. In practical terms this would mean that replacement of certain parts or rings of kinematic chains circuit elements of new system dynamic equations would not need to re-create all, only to find items, which depend on new generalized coordinates and make them into a new dynamic equation manipulation system.

Theorem. Let $MR(1,n)$ – be a set of matrices products for trigonometric functions of generalized coordinates for a given kinematic scheme of manipulation of the system. If in this kinematic scheme replace the i-link (degree of

freedom) to another, then $MR1(1,n)$ set of a new manipulation system will contain a $MR(1,i-1)$, $MR(i+1,n)$ set of manipulation systems, isolated from a given.

Using the results, we will build an algorithm to form the basic equations of the dynamics of manipulation systems with dedicated systems with its smaller dimensions. This algorithm will name signing algorithm elements of dynamic equations.

To construct the algorithm concluding elements of dynamic equations manipulation systems of different dimensions isolated from the base, let's consider the system of dimension n and *n-1.* There are two subsets $MR(1,n-1)$ and $MR(2,n)$, in the $MR(1,n)$set, that describe the handling of dimension *2 n-1.*These sets have common elements, as follows from Theorem 1.2, form a $MR(2,n-1)$ set. Thus, the set $MR(1,n)$ includes all the elements of $MR(2,n-1)$ set and part of elements of $MR(1,n-1)$ and $MR(2,n)$ set. In general, communication elements of dynamic equations of lower dimension manipulation systems with elements of dynamic equations manipulation systems of higher dimension can be presented in graph form. Elements of the set $MR(2,n-1)$ are placed in the center. Elements of the set $MR(1,n)$, that actually are needed to calculate, will be products of matrices $F_{i1}^T F_{nj}$, $i,j=\overline{1,n-1}$, $F_{ij}^T F_{n1}$, $i=\overline{1,n}$, $\quad j=\overline{2,i}$.

If for each vertex $MR(i,j)$, given in the Fig. 8.1, bind the matrix F_{ji}, we will find out, that in order to find these items, we have to find all results matrices' increases, which are located in the extreme left-top, extreme right of the matrix peaks and add the matrix F_{n1} products to the matrix of all vertices.

Hereof, the number of new elements of the set $MR(1,n)$ is:

$$k_n = k_{n-1} + 3(n-1) + 1, \quad k_0 = 1\,.$$

Thus, elements of dynamic equations can be divided into two groups. The first elements are already calculated for manipulation systems of lower dimension. The second group consists of items, that need to calculate for the manipulation system. Using the decomposition elements of dynamics equations, an efficient algorithm for the formation of members of inertia matrix H n-dimension manipulation system by n steps. Each member is formed by the iteration method, which is much simplifies the analysis and grouping of elements.

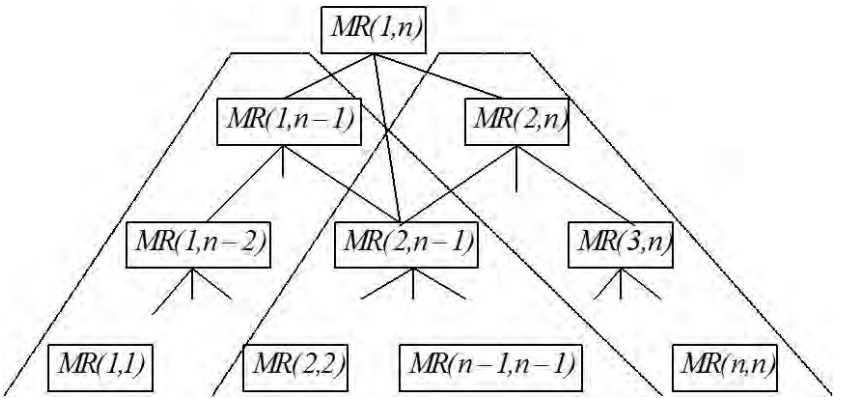


Figure 8.1: Kinematic scheme

Note, that the added elements may have common elements. That is why, to minimize the computational complexity were developed methods for grouping and structuring of these elements, that formed equations were optimized by the number of necessary arithmetic operations addition and multiplication. The basic idea which is used – dynamic representation of equations members in one image: $trAV_i$ or $trAU_i$ where is the matrix A depends on the generalized coordinates and parameters of links mass-and moments manipulation system and matrix – matrix of some special form.

Theorem. Let the given kinematic scheme of manipulation system and posed mass-and moments settings for the links. Then, for the diagonal elements of inertia matrix $H(\theta, \xi)$ of mathematical model (8.14), according to the method of decomposition, is right next image

$$h_{ii} = 2tr D_{i\,n-i+1} V_i, \tag{8.25}$$

where $D_{ij+1} = D_{ij} + C_{ij+1}$, $C_{ij+1} = P_{i+1}^T \cdots P_{i+j}^T K_{i\,i+j} + P_{i+1}^T C_{i+1j} P_{i+1}$, $i = \overline{1,n}$, $j = \overline{1,n-i+1}$; $D_{i1} = K_{ii}$, $C_{i1} = D_{i1}$.

Theorem. Let the conditions of the previous theorem be performed. Then for off-diagonal elements of inertia matrix $H(\theta, \xi)$, according to the method of decomposition, a fair representation of the following:

$$h_{ij} = tr\, B_{ij\,n-j+1} U_i,$$

where:

$$\begin{aligned}
B_{ij\,k} &= B_{ij\,k-1} + T_{ij\,k} + M_{ij\,k} - N_{ij\,k}, \quad k = \overline{1, n-j+1}\,, \\
N_{q\,j} &= P_q^T N_{q+1\,j}\, P_q, \quad q = \overline{j-1, i+1}\,, \\
N_{j\,j} &= R_j\, U_j . R_j = P_j^T C_{j\,j+k-1}\, P_j\,, \\
C_{m\,j+k-1} &= P_{m+1}^T \cdots P_{j+k-1}^T K_{m\,j+k-1} + P_{m+1}^T C_{m+1\,j+k-1}\; P_{m+1}\,, \\
m &= \overline{j+k-2, j}, \;\; C_{j+k-1\,j+k-1} = K_{j+k-1\,j+k-1}\;, \\
M_{q\,j} &= P_q^T \left(T_{q\,j} K_{q\,j+k-1} + M_{q+1\,j}\right)\, P_q\,, \\
M_{j\,j} &= U_j^T R_j . T_{q\,j} = P_q^T T_{q+1j} . T_{j\,j} = U_j^T P_j^T \cdots P_{j+k-1}^T\,, \\
N_{ij\,k} &= N_{i+1\,j}, \;\; M_{ij\,k} = M_{i+1\,j}, \;\; T_{ijk} = T_{i+1j} K_{i\,j+k-1}\,.
\end{aligned}$$

Similar results can be obtained for items coriolis and centrifugal forces. For gravitational components equations is fair next theorem.

Theorem. Let there be given circuit and kinematic parameters mass-and moments parameters for manipulation system. Then, elements of gravitational forces's vector are calculated using formulas

$$g_i = -tr\, C_{in} T_{1i}\, U_i\,, \tag{8.26}$$

where $C_{in} = L_{ii} + S_{i+1n}\,, \quad S_{jn} = P_j^T (L_{ji} + S_{j+1n})\,, \quad j = \overline{n, i+1}\,, \quad S_{nn} = P_n^T L_{ni}\,,$
$T_{1k} = T_{1\,k-1} P_k^T\,, \quad k = \overline{1, i}\,, \quad T_{10} = E\,.$

These received features have allowed to build, according to the method of decomposition, at each step computing for the minimum number of arithmetic operations of multiplication and addition, ie the computational complexity of this method is minimal.

Expertise and analyze the dependence of computational complexity of the dynamic equations formation based on developed new method for decomposition of manipulation systems. Under computational complexity, as already was observed above, we mean the number of arithmetic operations addition and multiplication, necessary for the formation model [6,29,33,34,40]. As computational complexity of the dynamic equations formation by proposed method depends on the kinematic scheme of manipulator and mass-and moments parameters, we will take base class manipulation robots with six (which is typical for most of the real manipulation robots) and less mobility.

As a result of such research we want to separate all possible optimal kinematic schemes, taking the computational complexity of optimality criteria and suitability for practical use in the field as widely as possible.

Let digit with 1-level around a rotation axis OX, with 2 – around the axis of the OY, found with 3 – around the axis OZ. Thus, the kinematic scheme, each ring of which has one degree of freedom, will record a sequence of numbers – 1,2 or 3. Then for six-rings manipulators total kinematic schemes in this class will be $3^6 = 729$. But, for the first time of such research, making analysis of the kinematic patterns, that are important from the standpoint of practical use only 42 kinematic schemes were received. For these schemes let's build momentum dynamic equation in numerical and analytical manner and research their computational complexity. Computational complexity characteristics of formation dynamic equations are presented in the table.

Table 8.1: Complexity of dynamic equations for the formation six-links manipulators

MP-6	+	*	MP-6	+	*
112131	169	173	113121	169	173
112231	167	153	113321	167	153
112331	178	155	113221	178	155
112321	204	200	113231	204	200
112311	165	154	113211	165	154
121131	196	198	131121	196	198
121331	247	251	131221	247	251
121311	239	253	131211	239	253
121321	283	308	131231	283	308
121231	283	308	131321	283	308
122311	232	224	133211	232	224
122321	278	283	133231	278	283
122131	243	256	133121	243	256
123121	306	334	132131	306	334
123131	306	334	132121	306	334
123211	290	295	132311	290	295
123311	246	219	132211	246	219
122331	240	216	133221	240	216
123231	341	362	132321	341	362
123221	293	285	132331	293	285
123321	287	271	132231	287	271

Here is obtained, based on the developed of new method, computational complexity of building dynamic equations of some industrial robots with three and six degrees of freedom: robot CL (consider modifications work individually with the three links of CL – three and six links of CL – 6), three-links robot PUMA, UMS-3B, manipulators TUR-10 and UEM-5 (see Table 8.2).

Table 8.2: Complexity of dynamic equations for the formation for industrial robots

	CL-3	PUMA	CL-6	UMS-3B	TUR-10	UEM-5
+	36	55	245	114	64	243
*	45	91	403	96	54	256

Thus, numerical-analytical method allows to form dynamic equations of manipulation robots with minimization of computational complexity and in the form, which allows the study of each specific kinematic scheme and using them to build a variety of new management methods, taking into account the dynamic properties of such systems. Knowledge of motion equations forming computational complexity manipulation is very important from a practical point of view, because gives an understanding of how to pick optimal computing means for building control systems in real time.

8.4 Analysis of modeling methods and motion control of manipulation systems

To construct the motion control of manipulation systems there usually use two approaches – technical and bionic The first one is based on applying of the known methods of control systems [2,6-9,16-19,22,23,36,41]. These methods can be divided into two groups based on the use of dynamic equations of manipulation systems. The first group consists methods which are based on using dynamics equations of system – optimal control, nonlinear control, linearization, variable structure control, and so on.

The second group includes methods that do not use the explicit exact dynamic equation of systems – recurrent adaptive estimation and adaptive control with standard model, control of learning type [3,12-15,21,24,28,33],

and others.

Feature of control systems of manipulation systems are real time running and a possibility to use information about the real state of the system by measuring the actual position, velocity and acceleration, force and moments of force. Thus, methods of constructing movements can be developed with possibilities of movement correction according to the real situation.

As it was mentioned above, during the movement mathematical models which describe this movement are changed. For example, motion in the positioning problem consists of three parts: acceleration, motion with constant speed, braking. At the stages of acceleration and braking, the full mathematical model should be used while at the stage of the motion with constant speed it is enough to calculate centrifugal and coriolis components. When moving with small velocity the contribution of these components will be insignificant and these components may not be calculated. Building mathematical models in numerical-analytical form allows to use effectively these properties by calculating only those elements of the model, which define this movement.

Cite examples of construction regulators for controlling manipulation of systems using dynamic models:

$$\begin{aligned} u(t) = H(\theta(t),\xi)\,[\,\ddot{\theta}^{pr}(t) + K_v\,[\,\dot{\theta}(t) - \dot{\theta}^{pr}(t))\,]\,+ \\ +K_p\,[\,\theta(t) - \theta^{pr}(t))\,]\,] + \dot{\theta}^T(t)Q_i(\theta(t),\xi)\dot{\theta}(t) + g(\theta(t),\xi), \end{aligned} \tag{8.27}$$

where $\theta^{pr}(t)$ – program trajectory of movement; K_v, K_p – given matrix; $Q_i(\theta(t),\xi)$ – diagonal matrix;

$$u(t) = H(\theta(t),\xi)\left[\ddot{\theta}^{pr}(t) + K_v\left[\dot{\theta}(t) - \dot{\theta}^{pr}(t)\right] + K_p\,[\theta(t) - \theta^{pr}(t)]\right] + g(\theta(t),\xi), \tag{8.28}$$

$$u(t) = H_i(\theta(t),\xi)\left[\ddot{\theta}^{pr}(t) + K_v\left[\dot{\theta}(t) - \dot{\theta}^{pr}(t)\right] + K_p\,[\theta(t) - \theta^{pr}(t)]\right] + g(\theta(t),\xi), \tag{8.29}$$

where $H_i(\theta(t),\xi)$ – diagonal elements of inertia matrix.

$$u(t) = \ddot{\theta}^{pr}(t) + K_v\left[\dot{\theta}(t) - \dot{\theta}^{pr}(t)\right] + K_p\,[\theta(t) - \theta^{pr}(t)] + g(\theta(t),\xi). \tag{8.30}$$

Also there exist different combinations of the control low mentioned above.

If meters efforts hinges is used then calculating the acceleration and the elements of the matrix of inertia, calculation of the vector $h(\theta(t), \dot{\theta}(t), \xi)$ in (8.13) can be avoided by calculating by the formula $h(\theta(t), \dot{\theta}(t), \xi) = u(t) - H(\theta(t), \xi)\ddot{\theta}(t)$ that allows to significantly reduce the number of calculations.

Thus, because of the considerable complicity and nonlinearity of the equations of dynamics the usage of classical methods of motion control of manipulation systems, as shown above, in real time causes significant difficulties.

Therefore, great attention is attracted an approach that let investigate the nature of processes of motion control of manipulator in order to simplify and highlight the most tangible elements of the movement and their usage in technical solutions of building control systems.

To build effective methods of control of complicated manipulation systems not only accurate knowledge of the parameters of the mathematical model is needed, but also the adequacy of the model of the object. In the previous section, mathematical models (dynamic equations) are based on the methods of classical mechanics, so constructed model will be assumed to be adequate to the object at the level of mechanics. In regard to parameters of model, to find or estimate their exact values is quite difficult. Moreover, the system as a whole is influenced by such actions as intractable backlash at the joints, the friction and, consequently, the elements of the correctional system heating, noise in the feedback channels, etc., which can lead to loss of the necessary control quality. That's why one of the most effective approach to organize control of manipulation work is a synthesis of global control, built on the basis of the system dynamics and the local, built on the basis of simple (usually linear) models. It is important for the latter to be easy to implement, use information about the current state and describe the process of movement with sufficient measure of the adequacy. To implement local control can be used adaptive mathematical models. Let's investigate methods of adaptive control of movement based on linear models with unknown parameters in more detail. Estimations of unknown model parameters are built by observations of the state of manipulation system in the process of motion. Then define control by these estimations optimizing some quality functional. Since the dynamics is described by equations of second order, then the discrete linear model can be performed as follows:

$$\theta(i+2) = A_1\theta(i+1) + A_2\theta(i) + A_3 + A_4u(i+1) + \varepsilon(i+2), \qquad (8.31)$$

where $u(i)$ is n-dimensional control vector at the i-th moment of time; A_1, A_2, A_4 – unknown matrix of $n \times n$ dimension, A_3 – unknown vector of $n \times 1$ dimension; $\varepsilon(i+2)$ – n-dimensional model error vector at $(i+2)$-th moment of time. Let's suppose that the trajectory of the system is given by a sequence of points in the space of generalized coordinates:

$$\theta^{pr}(i) \in R^n, \quad i = \overline{1,p}, \quad p < \infty. \tag{8.32}$$

Maintaining the programmed trajectory by the system will be taken for the functional of quality.

$$I_{i+1} = \left\|\theta^{pr}(i+2) - \theta(i+2)\right\|^2 + \delta \left\|u(i+1)\right\|^2, \tag{8.33}$$

where $\delta > 0$ – some constant.

Denote:

$$\begin{aligned} A &= (A_1, A_2, A_3, A_4) = (A_{(1)}, A_{(2)}, ..., A_{(n)}), \\ A_{(k)} &= (a_{k1}^{(1)}, ..., a_{kn}^{(1)}, a_{k1}^{(2)}, ..., a_{kn}^{(2)}, a_{k1}^{(3)}, a_{k1}^{(4)}, ..., a_{kn}^{(4)})^T, \\ \varphi(i+1) &= (\theta^T(i+1), \theta^T(i), 1, u^T(i+1))^T. \end{aligned}$$

Then the system (8.31) can be rewritten as follows:

$$\theta(i+2) = A^T\varphi(i+1) + \varepsilon(i+2). \tag{8.34}$$

Adaptive control algorithm consists of two stages. First estimates of unknown parameters are built by the measured states of the system then using these estimates control at the next cycle of motion is calculated. Therefore, we assume manipulation system to be in state $\theta(i+1)$, and will be seeking control $u(i+1)$, which on the $(i+1)$-th cycle will transfer manipulator in the state $\theta(i+2)$ provided that all previous conditions and controls are known. Estimates of unknown parameters of the matrix A are found by minimizing the error of modeling by least squares method. Procedure of estimation is written in the form of recurrence:

$$\hat{A}_{(k)}(i+2) = \hat{A}_{(k)}(i+1) + R(i+2)\varphi(i+1)\left[\theta_k(i+2) - \hat{A}_{(k)}(i+1)\varphi(i+1)\right],$$

$$\begin{aligned} R(i+2) = R(i+1) - R(i+1)\varphi(i+1)\varphi^T(i+1)R(i+1)\times \\ \times \quad \left[1 + \varphi^T(i+1)R(i+1)\varphi(i+1)\right]^{-1}, \end{aligned} \tag{8.35}$$

where $R(1) = E,\ k = \overline{1,n}, \quad R(i) - -(3n+1) \times (3n+1)$ – symmetric matrix.

Using estimations $\hat{A}(i+2)$ of matrix A estimation of control on the one cycle is can be found from the condition:

$$\left\| \theta^{pr}(i+2) - \hat{A}_1(i+2)\theta(i+1) - \hat{A}_2(i+2)\theta(i) - \hat{A}_3(i+2) - \hat{A}_4(i+2)u \right\|^2 + \delta \|u\|^2 \Rightarrow \min_u. \tag{8.36}$$

Hence:

$$\begin{aligned} u(i+1) = &\left(\hat{A}_4^T(i+2)\hat{A}_4(i+2) + \delta E \right)^{-1} \hat{A}_4^T(i+2) \times \\ &\times (\theta^{pr}(i+2) - \hat{A}_1(i+2)\theta(i+1) - \hat{A}_2(i+2)\theta(i) - \hat{A}_3(i+2)). \end{aligned} \tag{8.37}$$

Note that the number of computations required to implement the procedure (8.35)-(8.37) is large enough, but, in contrast to the control methods with using the equations of dynamics, it does not need to compute trigonometric functions of generalized coordinates.

Thus, the analysis of description methods, forming the equations of dynamics and control of complex manipulation systems show the challenges in building effective formalisms of representation such systems and the establishment of their mathematical models on the basis of these concepts and developing a new control methods, which would use non-linearity and complexity of the research object.

8.5 Construction of effective algorithms for dynamics and control problems solution

Given in the previous section algorithms of formation of dynamic equations in the numerical-analytical form allow not only to optimize the number of calculations or to allocate certain elements of the equations, but also effectively solve the problems of the dynamics in process of formation the dynamics equations. That is, if you want to have dynamics equation, then all the components which form these equations should be remembered. But if specific problem has to be solved, then do we need to form an equation first and then move on to the task?

Let's show the example of solving the inverse problem of dynamics, that instead of the traditional method – first find matrices $H(\theta,\xi)$, $Q(\theta,\xi)$ and vector $G(\theta,\xi)$ in the equations of dynamics (8.14) – the generalized control forces can be computed in the process of formation of the elements of matrices and vector.

Here, for simplicity, the algorithm for computing the first term in (8.14) is presented, namely, the product of inertia matrix of the vector of second derivatives of generalized coordinates$H(\theta,\xi)\ddot{\theta}$.

Step 1. In accordance with the first step of algorithm of forming the equations of dynamics based on decomposition properties, member h_{nn} is fully calculated.

Multiply it by $\ddot{\theta}_n$ and obtain the first conclusion of the n-th element of the vector of generalized forces u_n:

$$u_n^{(1)} = h_{nn}\,\ddot{\theta}_n.$$

Later in this task the member h_{nn} is not used and it can be set to zero.

Step 2. In the second step of the algorithm of forming the matrix $H(\theta,\xi)$, members h_{n-1n-1},h_{n-1n} are fully found. Multiplying h_{n-1n-1}, h_{n-1n} on $\ddot{\theta}_{n-1}$ and $\ddot{\theta}_n$ respectively, we find the first conclusion of the vector of forces u_{n-1}, and multiplying h_{n-1n} – $\ddot{\theta}_{n-1}$ a second conclusion for the vector of forces u_n:

$$\begin{aligned} u_n^{(2)} &= u_n^{(1)} + h_{n-1n}\,\ddot{\theta}_{n-1}\,, \\ u_{n-1}^{(1)} &= h_{n-1n-1}\,\ddot{\theta}_{n-1} + h_{n-1n}\,\ddot{\theta}_n\,. \end{aligned}$$

Elements h_{n-1n-1}, h_{n-1n} are no further used and can be destroyed, thus saving the resources of computing devices.

Step $k(k=3,...,n)$. Let $i=n-k+1$. On this step members h_{ii}, h_{ii+1},. . . ,h_{in} are calculated by the algorithm of forming the matrix of inertia forces, provide

relevant conclusions of generalized forces

$$\begin{aligned} u_n^{(k)} &= u_n^{(k-1)} + h_{in}\,\ddot{\theta}_i \\ u_{n-1}^{(k-1)} &= u_{n-1}^{(k-2)} + h_{in-1}\,\ddot{\theta}_i \\ &\vdots \\ u_{i+1}^{(2)} &= u_{i+1}^{(1)} + h_{ii+1}\,\ddot{\theta}_i \\ u_{ii}^{(1)} &= h_{ii}\,\ddot{\theta}_i + h_{ii+1}\,\ddot{\theta}_{i+1} + \ldots + h_{in}\,\ddot{\theta}_n \end{aligned}$$

members $h_{ii}, h_{ii+1}, \ldots, h_{in}$ are withdrawn.

As a result of the algorithm in n steps there will be calculated all the elements of the vector of generalized control forces for the first term in equations (8.14), namely the product $H(\theta,\xi)\,\ddot{\theta}$. There won't be any elements of inertia matrix.

Similarly the algorithms for calculation of centrifugal, coriolis and gravitational forces can be constructed.

Let's consider the direct problem of dynamics. To solve it the inverse matrix to $H(\theta,\xi)$ must be found. It and its individual members are used in methods of building the control of manipulator. For example, the values inverted to the diagonal elements of inverse matrix to matrix $H(\theta,\xi)$ are called the effective moments of inertia referred to the generalized coordinates and have a clear physical meaning. For kinematic pair of swinging degree of mobility it is moment of inertia -th,..., n-th manipulator links and weight relatively to the axis of i-th connection, and for the progressive couple – the mass of these links and weight. Using the algorithm of forming the inertia matrix of links to the k-th step the elements of symmetric matrix will be formed as:

$$H_{k-1} = \left\| \begin{array}{ccc} h_{i+1i+1} & \ldots & h_{i+1n} \\ \ldots & \ldots & \ldots \\ h_{ni+1} & \ldots & h_{nn} \end{array} \right\|,$$

where $i = n - k$.

At the k-step the elementsh_{ii}, h_{ii+1},. . . ,h_{in}, $i = n-k+1$ will be formed and, therefore, symmetrical matrix can be written as:

$$H_k = \left\| \begin{array}{cccc} h_{ii} & h_{ii+1} & \ldots & h_{in} \\ h_{i+1i} & & & \\ \ldots & & H_{k-1} & \\ h_{ni} & & & \end{array} \right\|. \qquad (8.38)$$

Therefore, matrix is formed by framing matrix H_{k-1}. It is clear that $H(\theta, \xi) = H_n$.

As the inertia matrix is proposed to be formed as (8.38), it is natural to calculate the inverse matrix $H_k^{-1}$from known matrixH_{k-1}^{-1}. It is important to not use a direct matrixH_{k-1}, because in our approach this matrix is withdrawn, releasing resources and reducing overall computational cost. That is, in fact, we will have a matrix on the k-th step:

$$H_k = \left\| \begin{array}{cccc} h_{ii} & h_{ii+1} & ... & h_{in} \\ h_{i+1i} & & & \\ ... & & H_{k-1}^{-1} & \\ h_{ni} & & & \end{array} \right\|. \tag{8.39}$$

Bring the general formula for finding the inverse matrix by partitioning of cells and when one cell of this partition is the inverse matrix. Let the output matrix has the structure $\left\| \begin{array}{cc} a & b \\ c & d \end{array} \right\|$, where *a, d*- a square matrices of some dimension, *b, c*- rectangular of relevant dimension. It is desirable to find the inverse matrix to this one so that in the resulting formulas expressions with d^{-1} will exist. Inverse matrix should also be found in the block form: $\left\| \begin{array}{cc} A & B \\ C & D \end{array} \right\|$.

Then:

$$\left\| \begin{array}{cc} a & b \\ c & d \end{array} \right\| \left\| \begin{array}{cc} A & B \\ C & D \end{array} \right\| = \left\| \begin{array}{cc} E_1 & O \\ O & E_2 \end{array} \right\|, \tag{8.40}$$

where E_1, E_2 – identity matrices.

From (8.40) we obtain a system of matrix equations relatively to unknown matrices *A, B, C, D*:

$$\begin{array}{l} aA + bC = E_1, \\ cA + dC = 0, \\ cB + dD = E_2, \\ aB + bD = 0. \end{array} . \tag{8.41}$$

From the second equation of system (8.41) we have:

$$C = -d^{-1}cA\,. \tag{8.42}$$

Substituting this expression in the first equation we get $aA - bd^{-1}cA = (a - bd^{-1}c)A = E_1$. Hence we have:

$$A = (a - bd^{-1}c)^{-1}\,. \tag{8.43}$$

Multiply the third equation in (8.41) by d^{-1} and solve it concerning the matrix D:

$$D = d^{-1} - d^{-1}cB\,. \tag{8.44}$$

Substituting this expression in the fourth equation we will have $aB + b(d^{-1} - d^{-1}cB) = 0$, or $(a - bd^{-1}c)B = -bd^{-1}$, and taking into account (8.43), finally we will get:

$$B = -Abd^{-1}\,. \tag{8.45}$$

Thus, we obtained the value for the matrix (8.42)-(8.45), which is the cells of the inverse matrix and in which use only matrix d^{-1}, nowhere presents matrix d.

Using the result we will write the algorithm of searching the inverse matrix $H^{-1}(\theta, \xi)$ in the process of forming the elements of inertia matrix $H(\theta, \xi)$.

Step 1. For the found element h_{nn}, $H_1^{-1} = h_{nn}^{-1}$ are calculated. Further save H_1^{-1} and withdraw h_{nn}.

Step $k (k = 2, ..., n)$. Denote $i = n - k + 1$. On this step we have members h_{ii},h_{ii+1},. . . ,h_{in} calculated by the algorithms of forming inertia forces matrix, and inverse matrix H_{k-1}^{-1}. Denote $v_k = (h_{ii+1}\ , \ldots , h_{in})^T$. Then matrix H_k can be written as:

$$H_k = \left\| \begin{array}{cc} h_{ii} & v_k^T \\ v_k & H_{k-1}^{-1} \end{array} \right\|.$$

Using formulas (8.40)-(8.45) inverse matrix to matrix H_k will be:

$$H_k^{-1} = \left\| \begin{array}{cc} (h_{ii} - v_k^T H_{k-1}^{-1} v_k)^{-1} & -v_k^T H_{k-1}^{-1}(h_{ii} - v_k^T H_{k-1}^{-1} v_k)^{-1} \\ -H_{k-1}^{-1} v_k (h_{ii} - v_k^T H_{k-1}^{-1} v_k)^{-1} & H_{k-1}^{-1} + H_{k-1}^{-1} v_k (h_{ii} - v_k^T H_{k-1}^{-1} v_k)^{-1} v_k^T H_{k-1}^{-1} \end{array} \right\|$$

$$= \frac{1}{\alpha} \left\| \begin{array}{cc} 1 & -v_k^T H_{k-1}^{-1} \\ -H_{k-1}^{-1} v_k & H_{k-1}^{-1}(h_{ii} - v_k^T H_{k-1}^{-1} v_k) + H_{k-1}^{-1} v_k v_k^T H_{k-1}^{-1} \end{array} \right\|,$$

$$\alpha = h_{ii} - v_k^T H_{k-1}^{-1} v_k.$$

Elements h_{ii}, h_{ii+1},. . . ,h_{in}, and inverse matrix H_{k-1}^{-1} are destroyed, H_k^{-1} is saved.

Thus consequentially for n steps we will get inverse matrix to the matrix of inertia forces $H(\theta, \xi)$. Due to the fact that matrices are symmetric, com-

putational complexity of the algorithm of finding the inverse matrix will be negligible.

Let's move to finding the effective moments of inertia matrix $I_{e,i},\ i = \overline{1,n}$. It is enough to make *-1* steps of described above algorithm of construction of the inverse matrix to the matrix $H(\theta,\xi)$. On the n-th step it should be found $H_{n-1}^{-1}v_n = (b_1, b_2, ..., b_{n-1})^T = b,\ a_1 = h_{11} - v_n^T b$.

Then effective moments can be calculated using formulas:

$$I_{e,1} = a_1,$$

$$I_{e,i} = (h_{n-1_{(i-1,i-1)}} + a_1^{-1}b_{i-1}^2)^{-1},\ i = \overline{2,n}, \tag{8.46}$$

where $h_{n-1_{(i,i)}},\ i = \overline{1,n-1}$, – diagonal elements of matrix H_{n-1}^{-1}.

8.6 Creation of a method and a system for controlling coordinated movements of manipulative robots

Feature of the systems for controlling of manipulative robots is work in real time, as well as the possibility of using information about the real state of the manipulator by measuring the true position, velocity and acceleration, forces and torques. Methods of motions building with the possibilities of movement correction according to the real situation are can be developed. Researches of Bernstein [3], and others on the organization of movements [6,12,14] of beings can be essentially used for analysis and synthesis of manipulative robots control systems. Hierarchic of structures and use of reverse links are the basis for making such systems.

At each level of hierarchy command information of the upper levels is processed and using the information available for this level of hierarchy, decisions is taken aimed at achieving the objectives of this level and command information is generated on the lower level. Herewith control is the result of many components, some of which are determined by signals coming from the upper levels of the hierarchy, and some part is invariant to them. Thus, on changing information on the upper levels the correction of action should be made on subordinates' levels in accordance with these changes. In organization of motion control system for manipulative robots three levels of hierarchy are

emitted: strategic, tactical and executive. The role of the first two levels is to organize and plan movements to solve the task; the third role is to implement these movements.

At the strategic level tasks of implementation of the goal in general and construction of possible trajectories of motion in specific environments with obstacles are being solving.

At the tactical level the motion trajectory of model in the external world for manipulative robot converts into a trajectory of change of generalized coordinates by solving the inverse problem of planning. At the same level the law of motion which is being implemented at operational level is determined. In fact, it represents the level at which different forms of movements and their implementation mechanisms are stored. Output signals of this level are submitted to the executive level drives, which cause physical movement of parts of the manipulator in space. Information about the actual state of manipulative robot during movement comes at the executive level from the position transducers, speed, etc.. Under such control system organization complexity of each level depends on the upper levels solutions of problems. Therefore, new methods were proposed which would simplify solving of tasks of lower levels of hierarchy.

Since at the end the problem is reduced to the generalized coordinates displacement which are received in the process of solving the inverse problem of the state planning, then the complexity of executive level depends on what values of the generalized coordinates are chosen for the solution and how they agree with previous. Coordination in terms of physiology of movements is overcoming the excessive levels of mobility to achieve the goals. It is clear that the fewer quantity of degrees of mobility (number of links) take part in movement, the easier the realization of such a movement on the operational level. Motion manipulator construction includes calculating a trajectory to reach the objective state, the law of motion on this path and calculating forces and torque that are needed to ensure the implementation of motion on a given trajectory.

To overcome the excessive number of generalized coordinates (geometric coordination) that transform the manipulator to the objective state means to identify those of them that make the largest contribution to the movement, thus making the objective state to be achievable. Kinematic coordination of movement is determined by finding such changes of generalized coordinates as function of time in which between them will be reduced interdependence.

Coordination at the level of dynamics (dynamic coordination) system consists in calculating of those elements of motion equations that make the largest contribution to its implementation.

Thus, the method of movement coordination of manipulative robot is reduced to the consistent solution and the correction of necessary tasks:

1. set the initial state of manipulator (the vector of generalized coordinates),
2. set objective values of position, approach and orientation capture vector of manipulator,
3. make geometric coordination,
4. determine time interval,
5. make a kinematic coordination,
6. check restrictions on the location, speed and acceleration on each of the generalized coordinates. If there are violations, then increase the length of time and go to paragraph 8.5,
7. form the equations of motion,
8. check limits for generalized control forces. If there are violations, then increase the length of time and go to paragraph 8.5,
9. make a dynamic coordination.

This way, a time span for making the specified move will be found as the result of the algorithm, without breaking any restrictions; a set of structure of dynamic equations to implement the move will be found as well. The proposed motion coordination algorithm agrees well with experimental data that confirm the setting (joints and muscle groups choice) before making movements by man. Let's propose the organization of manipulative robot motion control system based on coordination approach. Coordination algorithm can be implemented in advance, in off-line mode, and then a found set of simplified structures put into control system thus creating a system 'memory' (base of motion). Clearly, the control found by the simplified models in the process of coordination won't let accurately track the program trajectory but the deviation between the real and program trajectories will be insignificant

that is guaranteed to fall in the of a trajectory vicinity. Also many factors (friction, wear, noise, inaccuracies of parts manufacture) affect the movement of the manipulator and it's difficult to take them into account.

Since there exists a possibility to measure the real state of the manipulator, to stabilize motion by program trajectory it's necessary to find additional control actions during the motion process that will compensate the deviation between real and program trajectory. Such control can be calculated by local linear adaptive controllers, which depend only on one generalized coordinate. Parameters of regulator are selected so, that it allows manipulation of a system in real time and to meet a certain level of quality. Thus, the original approach to the organization of the system of motion control of manipulative robot in parallel working units has been developed. Control actions of the algorithm are computed based on the results, in the first coordination unit in real time, additional control actions, based on methods of adaptation are computed in the second block in real time too.

8.7 Optimization and pseudoinversity in problems concerning equilibrium states

Significant attention is paid to the task of study equilibrium states of manipulative systems especially while building manipulative robot's control [2,5,6,16,22]. This is due primarily to the possible of use statistical models for synthesis controls when performing assembly operations, handling large volume and heavy loads, performing welding operations, etc., that is, in general class of practical problems that do not require fast actions. Therefore, when performing operations of such class, influence of inertia, centrifugal and coriolis forces can be neglected. In this case the main task of control system is the calculation control acts that would compensate gravitational forces of manipulative system. Such control actions can be obtained from building static models. To construct the equations of statics different approaches of classical mechanics are used; such as D'Alembert's principle [19], which allows us to construct recurrent relations for forces and torques which act on each link of manipulation system or the Lagrange's equation of the n-th type, in which the equation of statistic is the partial case. In this paragraph we will formulate the three problems of statics. Based on D'Alembert's principle, static model of manipulation system is represented as a system of linear equations.

Depending on the capabilities of solving this system, the equilibrium properties of manipulation systems are investigated and methods for solving problems of statics based on the pseudoinverse linear operations and optimization [1,10,15] are presented. For problem statement static structured approach to the mathematical description of manipulation systems will be used. Write the balance equation using the principle of d'Alembert (exemption from connections) for i-th link of the manipulator.

$$\begin{cases} R(i) = R(i+1) - G(i)i_3, \\ Q(i) = Q(i+1) + R(i+1) \times (K(i)\tilde{b}(i)) - G(i)i_3 \times (K(i)\tilde{p}(i)), \end{cases} \tag{8.47}$$

where $R(i), Q(i)$ – force and torque at the point of joining i-th link to the $(i-1)$-th, respectively; $R(i+1), Q(i+1)$ – force and torque at the point of joining i-th link to the $(i+1)$-th, respectively; $K(i) = (k_1(i), k_2(i), k_3(i))$ – orientation matrix of i-th link in an absolute coordinate system; $\tilde{b}(i)$ – vector specified in the i-th coordinate system (which is linked to the i-th link manipulation system) which connects $(i-1)-$th point with the $(i+1)$-th; $\tilde{p}(i)$ – vector directed from $(i-1)$-th point to the center of mass of i-th link specified in the i-coordinate system; $G(i)$ – weight of link; i_3 – unit vector of absolute coordinate system. Orientation matrix can be determined by following recurrent formula:

$$K(i) = K(i-1)\tilde{C}(i-1)A_1^T(\theta_i)\,, \tag{8.48}$$

where $\tilde{C}(i-1)$ – unit vectors matrix of joining i-th level; $A_1(\theta_i)$ – orthogonal rotation matrix; θ_i – generalized coordinate.

The generalized control force in the i-join will equal the projection of force or torque on the axis of kinematic pair, respectively in progressive or rotational connection:

$$u_i = k_1^T(i)(\Delta_i Q(i) + (1-\Delta_i)R(i))\,, \tag{8.49}$$

where $\Delta_i = 0$ in progressive and $\Delta_i = 1$ in a rotational connection.

Let n-th link of manipulation system is influenced by external forces:

$$R = (R_1, R_2, R_3)^T,$$
$$Q = (Q_1, Q_2, Q_3)^T,$$

force and torque respectively.

Let's introduce vector $f = (R,\, Q)^T = (R_1,\, R_2,\, R_3,\, Q_1,\, Q_2,\, Q_3)^T$.

Static models associate with each other the state of manipulative system, which is determined by the vector of generalized coordinates, and generalized forces that are necessary for finding the system in this state, as well as acting external forces.

Depending on what parameters are known and which are needed to be identified, we highlight three problems of statics:

Problem 1. Given:

1. position of manipulator $\theta = (\theta_1, \ldots, \theta_n)^T$,
2. force and torque that are applied to the n-th link of manipulative system $f = (R, Q)$.

Define: generalized control forces $u = (u_1, ..., u_n)$.

Problem 2. Given:

1. generalized control forces $u = (u_1, ..., u_n)$,
2. generalized coordinates $\theta = (\theta_1, ..., \theta_n)^T$.

Define: force and torque that are applied to the n-th link of system.

Problem 3. Given:

1. generalized control forces $u = (u_1, ..., u_n)$,
2. force and torque that are applied to the n-th link of system.

Define: vector of generalized coordinates $\theta = (\theta_1, ..., \theta_n)^T$.

Since the balance equation (8.47) are given in an absolute coordinate system, then the product $K(i)\tilde{b}(i)$ is a vector $\tilde{b}(i)$, given in coordinate system of i-th link and listed in the absolute coordinate system. Denote this vector in an absolute coordinate system by $b(i)$. Similar the product $K(i)\tilde{p}(i)$ is a vector $\tilde{p}(i)$, given in coordinate system of i-th link and listed in the absolute coordinate system. In the absolute coordinate system denote it by $p(i)$. Introduce also the notation $g(i) = G(i)i_3$.

Then equation (8.47) can be rewritten as follows:

$$\begin{array}{l} R(i) = R(i+1) - g(i)\,, \\ Q(i) = Q(i+1) + R(i+1) \times b(i) - g(i) \times p(i), \quad i = \overline{n,\ 1}\,, \end{array} \tag{8.50}$$

and $R = R(n+1), Q = Q(n+1)$ are force and torque respectively are applied to the n-th link of system.

Theorem. Let manipulation system of n-dimension is given. Then the system of equations

$$u = A(\theta,\, p)f + h(\theta,\, p), \tag{8.51}$$

where $A(\theta, p)$ – matrix $n \times 6$ with elements:

$$a_{i,\,1} = (1 - \Delta_i)k_{11}(i) + \Delta_i \left[k_{13}(i) \sum_{j=i}^{n} b_2(j) - k_{12}(i) \sum_{j=i}^{n} b_3(j) \right],$$

$$a_{i,\,2} = (1 - \Delta_i)k_{12}(i) + \Delta_i \left[k_{11}(i) \sum_{j=i}^{n} b_3(j) - k_{13}(i) \sum_{j=i}^{n} b_1(j) \right],$$

$$a_{i,\,3} = (1 - \Delta_i)k_{13}(i) + \Delta_i \left[k_{12}(i) \sum_{j=i}^{n} b_1(j) - k_{11}(i) \sum_{j=i}^{n} b_2(j) \right],$$

$$a_{i,\,4} = \Delta_i k_{11}(i),\ a_{i,\,5} = \Delta_i k_{12}(i),\ a_{i,\,6} = \Delta_i k_{13}(i),$$

and $h(\theta, p)$- vector $n \times 1$ with elements:

$$h_i = k_1^T(i)d(i), \tag{8.52}$$

where $d(i) = -(1-\Delta_i) \sum_{j=i}^{n} g(j) - \Delta_i \sum_{j=i+1}^{n} g(j) \times \sum_{k=i}^{j-1} b(k) - \Delta_i \sum_{j=i}^{n} g(j) \times p(j)$, describes static model of manipulation system.

Proof. From the equilibrium equations (8.50) for i-th link of manipulation systems follows that:

$$R(i) = R(i+1) - g(i) = R(i+2) - g(i+1) - g(i) = ... = R - \sum_{j=i}^{n} g(j), \tag{8.53}$$

$$\begin{aligned} Q(i) &= Q(i+1)+R(i+1)\times b(i)-g(i)\times p(i)=Q(i+2)+R(i+2)\times b(i+1)+ \\ &- g(i+1)\times p(i+1)+R(i+1)\times b(i)-g(i)\times p(i)= \\ &= ... = Q + R \times \sum_{j=i}^{n} b(j) - \sum_{j=i+1}^{n} g(j) \times \sum_{k=i}^{j-1} b(k) - \sum_{j=i}^{n} g(j) \times p(j). \end{aligned} \tag{8.54}$$

From (8.49), considering (8.53), (8.54), find generalized force u_i:

$$\begin{aligned} u_i &= k_1^T(i)\left(\Delta_i Q(i) + (1-\Delta_i)R(i)\right) = \\ &= k_1^T(i)\Bigg(\Delta_i\left[Q + R \times \sum_{j=i}^{n} b(j) - \sum_{j=i+1}^{n} g(j) \times \sum_{k=i}^{j-1} b(k) - \sum_{j=i}^{n} g(j) \times p(j)\right] + \\ &+ (1-\Delta_i)\left[R - \sum_{j=i}^{n} g(j)\right]\Bigg) = \\ &= k_1^T(i)\left(\Delta_i Q + \Delta_i R \times \sum_{j=i}^{n} b(j) - \Delta_i \sum_{j=i+1}^{n} g(j) \times \sum_{k=i}^{j-1} b(k) + \right. \\ &- \left. \Delta_i \sum_{j=i}^{n} g(j) \times p(j) + (1-\Delta_i)R - (1-\Delta_i)\sum_{j=i}^{n} g(j)\right). \end{aligned} \tag{8.55}$$

Write this equation as follows:

$$u_i = (a_{i1},\, a_{i2}, ..., a_{i6}) \begin{pmatrix} R_1 \\ R_2 \\ R_3 \\ Q_1 \\ Q_2 \\ Q_3 \end{pmatrix} + h_i.$$

Using the vector representation of the product as a product of the matrix and vector:

$$R \times d = \begin{pmatrix} 0 & d_3 & -d_2 \\ -d_3 & 0 & d_1 \\ d_2 & -d_1 & 0 \end{pmatrix} \begin{pmatrix} R_1 \\ R_2 \\ R_3 \end{pmatrix}, \qquad d = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix},$$

get:

$$\begin{aligned}
a_{i,1} &= (1-\Delta_i)k_{11}(i) + \Delta_i \left[k_{13}(i) \sum_{j=i}^{n} b_2(j) - k_{12}(i) \sum_{j=i}^{n} b_3(j) \right], \\
a_{i,2} &= (1-\Delta_i)k_{12}(i) + \Delta_i \left[k_{11}(i) \sum_{j=i}^{n} b_3(j) - k_{13}(i) \sum_{j=i}^{n} b_1(j) \right], \\
a_{i,3} &= (1-\Delta_i)k_{13}(i) + \Delta_i \left[k_{12}(i) \sum_{j=i}^{n} b_1(j) - k_{11}(i) \sum_{j=i}^{n} b_2(j) \right], \\
a_{i,4} &= \Delta_i k_{11}(i), \\
a_{i,5} &= \Delta_i k_{12}(i), \\
a_{i,6} &= \Delta_i k_{13}(i).
\end{aligned} \tag{8.56}$$

To find h_i:

$$h_i = k_1^T(i) \left[-\Delta_i \sum_{j=i+1}^{n} g(j) \times \sum_{k=i}^{j-1} b(j) - \Delta_i \sum_{j=i}^{n} g(j) \times p(j) - (1-\Delta_i) \sum_{j=i}^{n} g(j) \right].$$

Theorem is proved.

Let's move to the solving problems 1-3 based on the representation (8.51) of statics equations of manipulation system.

Problem 1 – direct static problem is most simple. It solution – the vector of generalized forces – is uniquely determined by formulas (8.51) – (8.52) at a fixed position and attached efforts to clamp.

Problem 2. Matrix $A(\theta, p)$, in general, is a rectangular matrix of dimension $n \times 6$. Let $n = 6$, that manipulation system has six degrees of mobility. To find the force and moment forces solve system (8.51) concerning f. Get:

$$f = A^{-1}(\theta,\, p)(u - h(\theta,\, p)) = A^{-1}b. \tag{8.57}$$

Perform a more detailed investigation of the system (8.57).

A. If $\det A(\theta, p) \neq 0$, then for a known vector of generalized coordinates θ and generalized forces vector u vector f is uniquely determined.

B. If $\det A(\theta, p) = 0$, then two possible cases exist:

B1. For given u and θ vector f could be found ambiguously, there exist a whole set of vectors f, which satisfy (8.51).
So set problem to find f from the condition:

$$\Omega f = \arg\min_{f} \|Af - b\|^2 . \tag{8.58}$$

Solution of (8.58) will be:

$$\arg\min_{f \in \Omega_f} \|f\|^2 = \hat{f} = A^+ b ,$$

where A^+ – pseudo inverse matrix.

B2. For given u and θ solutions do not exist. This means that for them there is no statically balanced situations. Let $n \neq 6$. In this case matrix $A(\theta, p)$ will be rectangular, and for finding the forces and torques that are applied to the n-th link of manipulation system criterion (8.58) can be used.

Thus, to solve problem 2 pseudo inverse matrix should be used.

Problem 3 – static inverse problem – one of the most difficult problems. Let u^* – given vector of generalized control forces, f^* – given force and torque applied to the n-th link. Vector of generalized coordinates is found from the conditions of minimum of the functional:

$$F(\theta) = \|A(\theta,\, p) f^* + h(\theta,\, p) - u^*\|^2 \Rightarrow \min_{\theta \in \Theta}, \tag{8.59}$$

where Θ – a set of internal restrictions on the generalized coordinates:

$$\Theta = \left\{ \theta_i :\, \theta_{i\,\min} \leq \theta_i \leq \theta_{i\,\max},\, i = \overline{1,\, n} \right\}, \tag{8.60}$$

$\theta_{i\,\max}$, $\theta_{i\,\min}$ – given numbers.

To find the vector of generalized coordinates we apply gradient procedure:

$$\theta(i+1) = \pi_\Theta \{ \theta(i) - \rho_i \operatorname{grad}_{\theta(i)} F(\theta(i)) \} , \tag{8.61}$$

where π_Θ – operation of projection onto the set Θ (8.60), ρ_i – iteration pitch, $\theta(i) = (\theta_1(i), ..., \theta_n(i))^T$.

To find the partial derivatives we have:

$$\frac{\partial F(\theta)}{\partial \theta_i} = 2 \left(A(\theta, p) f^* + h(\theta, p) - u^*\right)^T \left(\frac{\partial A(\theta, p)}{\partial \theta_i} f^* + \frac{\partial h(\theta, p)}{\partial \theta_i} \right). \tag{8.62}$$

To determine the derivatives $\frac{\partial A(\theta,p)}{\partial \theta_i}$, $\frac{\partial h(\theta,p)}{\partial \theta_i}$ it is needed to take the derivative by θ_i using orientation matrix (8.48)

$$\frac{\partial K(i)}{\partial \theta_i} = K(i-1) \tilde{C}(i) \frac{\partial A_1^T(\theta_i)}{\partial \theta_i}. \tag{8.63}$$

As a result of computing (8.61)-(8.63) we get the vector of generalized coordinates, which will be the solution of problem 3.

Thus, by presenting the equations of statics in the form (8.51) and using pseudo inverse techniques and optimization methods we have received constructive solution of problems 1-3.

8.8 Method of motion planning manipulation systems in arbitrary configuration with obstacles space

To solve the problem of trajectory planning, a new approach is proposed. In it's essence it means creating the working space of admissible states of manipulation systems (MS) as a discrete set of points obtained by solving the direct kinematic problem of discrete vector of generalized coordinates.

Such calculations are performed for a particular MS once and saved into a database. A subset of vectors of coordinates in which MS is positioned in the same point in space, convert into an Artificial Neural Networks. For

planning trajectories the following is offered: find by association a vector in subset of vectors of generalized coordinates using neural network which is most similar to the vectors of objective state. Thus, computing costs are minimized because the neural network 'learns' all the possible ways of achieving the objective state and the required vector is selected from the set of values that once were calculated and stored in the database. So trajectory planning of MS can be organized as a similar procedures of training and decision making to execute manipulations and locomotion of higher order organisms.

Locomotive activity belongs to the highly automated movements. This fact was pointed by Berstain [3]. Locomotive systems, especially anthropomorphous, are extremely complex dynamical systems, both in terms of mechanical structure and in terms of its control system. It should be mentioned that person can use 800 muscles to make different movements. So the biomechanical difference between human locomotion, developed animals and artificial self-acting devices goes from a large number of available degrees of its mobility. Coordination of movements means overcoming excessive levels of mobility of a body unit which moves, in other words that is transformation it into a control system. So, coordination is controlling mean of a locomotion. It should be treated not as stopping the additional degrees of freedom, but overcoming them.

Lack of information on control mechanism particularly affects issues such as communications control actions with the appropriate motions, individual pairs participation in muscles control efforts, determination of the criteria according to which the locomotive processes are carried out. In regard to such complexity of control structure and a large number of drivers which make a human move, we will use certain methods of movement control of live systems at the proposed approach to the synthesis of an artificial movement. The fact that modern state of the computer technology development, development of algorithms for database control allow to apply the methods which implement the search for the needed information (even imaginative) from the large databases influenced on a new approach to control movements of MS. Of course, it isn't made by physiological methods, but a real possibility to use algorithms which are based on simulating the behavior of live creatures exists. These possibilities are:

1. motor experience gained in relation to musculoskeletal driving apparatus, which can be represented as a database of neural networks that contain encrypted discrete space of possible movements of MS. With

the help of these movements, the object can be put to the desirable objective position (that is in a vicinity of a space point),

2. an analogue of approach of movement control for moving MS from the base point to the given,

3. at each movement act which is associated with overcoming external, uncontrollable and variable forces organism continuously faces with such irregular and, more often, unforeseen complications, which reject the movement of the target program trajectory. It is impossible or highly impractical to overcome them by using correction pulses which are targeted at restoring the previous motion. In these cases, receptor information acts as exciter for adaptive rebuilding of program 'on the go', starting from the small, in a technical meaning, movement of arrows from one to another, which lies aside, project trajectory and ending with the quality reorganization of the programs that change the nomenclature itself of the sequenced elements and stages of the movement act. From the mentioned, a new approach is proposed which allows to dynamically (on the go) change planning trajectories using motor experience, by prohibiting motion that can not be realized in the current moment.

Describe the trajectory planning algorithm with information-search approach for planning trajectories of MS of a custom configuration in the space of restrictions. This algorithm allows to plan the trajectory selecting them from the space of admissible states. So there implements an opportunity of hitting from the state to the nearest objective, so called reference state. For positioning MS to the given objective state (which is not a point that belongs to the discrete space) an approach that uses the same algorithm, but iterative, for gradual (exact) approximation to the objective point is proposed. So until reaching the objective point, discrete workspace of acceptable states for a discrete set of vectors of generalized coordinates in the vicinity of the objective points are being generated.

This way, the destination point is reached from the nearest base point with the given precision. For computer modeling of MS object-oriented approach is used, that is, using a mathematical model, a library of base classes is created, which will have classes that implement a computer model of MS with an arbitrary number of links and arbitrary types of kinematics in kinematic pairs. Neural networks (NN) algorithms for the organization of associative memory are used within the implemented computer model. When there is

no explicit need for NN to do it, that is, having the number of a sample is enough, associative memory is successfully implemented by the network of Hemming. In the method developed, NN with reverse link were used. This NN implements the Hemminh's associative memory algorithm. Let's consider a question about how, using a computer model described above, realize the information retrieval approach to solve the problem of planning states of MS in the environment with restrictions. That is, the following algorithm for solving inverse kinematics problem is proposed:

1. Disorganization of space, base points. Database is created. It consists of the following tables: table containing discrete Cartesian space of working environment of MS; table that contains the discrete space of generalized coordinates of the work environment of MS; table containing the Cartesian space point which belongs to MS and reside in the state defined by the vector of generalized coordinates.

In order to obtain and store knowledge in the specified database the following algorithm is implemented:

1.1. all possible (for this MS model) combination of generalized coordinate are enumerated (with fixed step) and direct problem of kinematic is solved, and as a result vectors of generalized coordinates and points of Cartesian space are obtained,

1.2. these vectors of generalized coordinates and the corresponding point in Cartesian space above are fixed in a specified database,

1.3. vector coordinates which lead NS to the same state and transform into NN are automatically grouped using the structure of database.

Here follows the algorithm of trajectory planning for MS in the discrete Cartesian space using the obtained knowledge. Input data: a) current state of MS which defined by point in discrete Cartesian space (x, y, z) and the vector of generalized coordinates, b) the objective state, which defined by point of discrete Cartesian space (x', y', z').

2. Algorithm of transition MS from its current state to the objective state.

2.1. Using the coordinates of the objective state (x', y', z'), a record from the table of states is looked up and then matched with the corresponding

set of records from the table of generic coordinates, that is, a NN is constructed which takes a vector of current state's coordinates as an input,

2.2. NN's output is the vector of generalized coordinates of a system that meets the objective point θ'. Moreover, vector obtained is most similar to input vector from all possible sets of vectors. Thus, the motion is received, for which least cost of transition are needed,

2.3. In the cycle, with a particular step, smooth move from current state θ to the objective state θ' is executed.

Based on the given approach there were developed the algorithms for solving following tasks:

1. approximation from the reference point $(x, y, z) \in XYZ$ to an arbitrary point $(x', y', z') \notin XYZ$ from the discretization described above but iteratively and only in the vicinity of reference point,

2. planning of trajectories of a MS in an environment with restrictions that has a proposed set of generic coordinates vectors with corresponding states of MS in case of matching the restrictions are updated in the database as forbidden and not to be used in further planning.

Thus, if present MS as described above computer model, the information retrieval approach to solve the problem of planning of states in an environment with restrictions can be realized.

Set of three-dimensional modeling programs of planning procedures of trajectories of MS (see Fig. 8.2) were developed. The following features are implemented in this set of programs: indication of arbitrarily configuration of MS; computation-learning (space discretization); finding the optimal trajectory of movement using Neural Networks; set the vicinity of restrictions; building trajectories.

The program contains a graphical representation of a discrete Cartesian space of possible states of MS. The space of possible states of MS in the following configurations is represented in Fig. 8.3: links – 7; length of each one – 30 sm. The step for discretization: link from the 1-st to 4-th – 60 degrees, from the 5-th – 30 degrees. Types of kinematic links (all revolving around

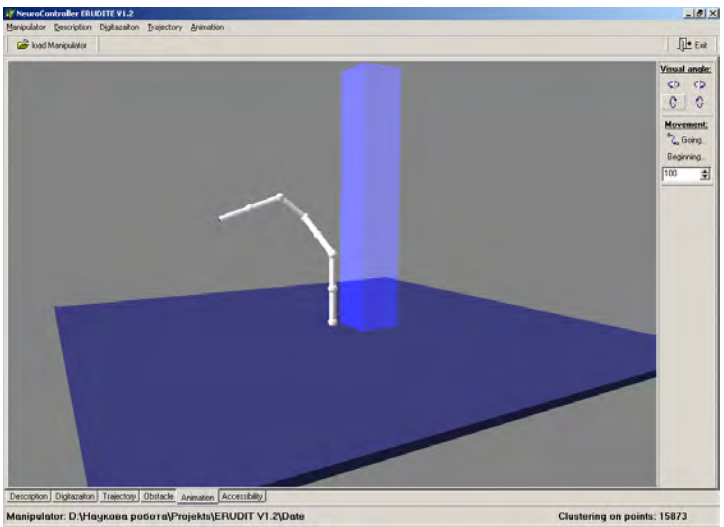


Figure 8.2: Example of three-dimensional modeling

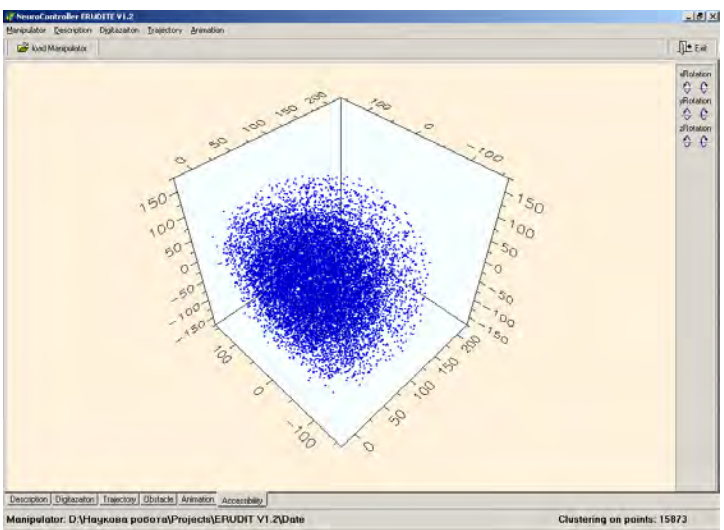


Figure 8.3: Discrete space of possible states of manipulation system

the axes): 1-st – X, 2-nd – Y, 3-rd – Z, 4-th – X, 5-th – Z, 6-th – Y and 7-th – X. Restrictions on changes of angles by links: 1-st – $(-180^o \div 180^o)$, 2-nd – $(-120^o \div 180^o)$, 3-rd – $(-120^o \div 180^o)$, 4-th – $(-180^o \div 180^o)$, 5-th – $(-150^o \div 180^o)$, 6-th – $(-150^o \div 180^o)$. The results of modeling confirm the effectiveness of the proposed approach.

Bibliography

[1] A. Albert: Regression, pseudoinversion and recurrent estimation. – M.: Nauka, 1977, – 180p. (in Russian).

[2] S. Arimoto: Learning control theory for robotics motion Int. J. Adaptive Control and Signal Processing, 1990, Vol. 4, pp. 543-564.

[3] B. Armstrong, O. Khatib, J. Burdick: The explicit dynamic model and inertial parameters of the PUMA 560 Arm IEEE Int. Conf. Rob. and Autom., San-Francisco, Apr.7-10, 1986, Vol. 1, pp. 510-518.

[4] V.E. Berbiyuk: Dynamics and optimization of robotics systems. – Kyiv.: Naukova dumka. 1989, – 187p. (in Ukrainian).

[5] N.A. Bernstain: Shot story about phisiology of movement and phisiology of activity. – M.: Medicine, 1966, – 360p. (in Russian)

[6] B.N. Booblik, M.F. Kirichenko: Basis of control theory. – Kyiv.: Vyshcha sthcola, 1975, – 320p. (in Ukrainian)

[7] J. Denavit, R.S. Hartenberg: A kinematic notation for lower-pair mechanism based on matrices J. Appl. Mech., 1955, Vol. 22, pp. 215-221.

[8] S. Dubowsky, D.I. Desforges: The application of model-reference adaptive control to robot-manipulators Trans. ASME. J. Dyn. Syst., Meas., Contr., 1979, Vol. 101, No. 9, pp. 193-200.

[9] S. Dubowsky, J.L. Grant: Application of symbolic manipulation to time domain analysis of non-linear dynamic systems Trans. ASME. J. Dyn. Syst., Meas. and Contr., 1975, Vol. 97, No. 3, pp. 60-68.

[10] K. Fu, R. Gonsales, K. Li: Robotics. – M.: Mir, 1989, – 624p. (in Russian)

[11] J.M. Hollerbach: A recursive Lagrangian formulation of manipulator dynamics and comporative study of dynamics formulation complexity IEEE Trans. Syst., Man, Cybern., 1980, Vol. 2, SMC-10, pp. 730-736.

[12] M.F. Kirichenko, Yu.V. Krak, R.O. Soroka: Manipulation robot optimisation. – Kyiv.: Lubid, 1990, – 145p. (in Ukrainian)

[13] A.A. Kobrinskii, A.E. Kobrinskii: Manipulation systems of robots: basis hardware, theory items. – M.: Nauka, 1985, – 344p. (in Russian)

[14] G.V. Korenev: Purposeful mechanics of controlling manipulators. – M.: Nauka, 1979, – 447p. (in Russian)

[15] V.V. Kozlov, V.P. Makarichev, A.V. Timofeev, E.I. Yurevich: Dynamics of robot control. – M.: Nauka, 1984, – 336p. (in Russian)

[16] Yu.V. Krak: Method of manipulation robots dynamics formation in numerical-analitical form/Technicheskaya kibernetica, 1993, No. 1. pp. 137-141. (in Russian)

[17] Yu.V. Krak: Koordination approach to manipulation robots movement organisation/Problemy upravleniya I informatyky, 1995, No. 4, pp. .120-128. (in Russian)

[18] Yu. Kryvonos, Yu. Krak, M. Kirichenko: Modelling, analysis and synthesis of manipulation systems. – Kyiv.: Naukova dumka, 2006, – 202p.(in Ukrainian)

[19] R. Kuc, B. Siegel: Physically based simulation model for acoustic sensor robot navigation IEEE Trans. Pattern Anal. and Mach. Intel., 1987, Vol. 9, No. 6, pp. 766-778.

[20] F.N. Kulakov: Supervision control of manipulation robots. – M.: Nauka, 1980, – 448p. (in Russian)

[21] H.-B. Kuntze: Regelungsalgorithmen fur rechnergesteuerte Regelungstechnik, 1984, Vol. 32, No. 7, pp. 215-226.

[22] C.S.G. Lee, B.H. Lee: Resolved motion adaptive control for mechanical manipulators Trans. ASME: Meas., and Contr., 1984, Vol. 106, No. 9, pp. 134-142.

[23] M.C. Leu, N. Homati: Automated symbolic derivation of dynamic equations of motion for robot-manipulators Trans. ASME: J. Dyn. Syst., Meas., and Contr. 1986, Vol. 108, pp. 172-179.

[24] P.J. McKerrow: Introduction to robotics. Singapore: Addison-Wesley Publ. Ltd., 1991. – 811p.

[25] V.S. Medvedev, A.G. Leskov, A.S. Yuchenko: Systems of control of manipulation robots. – M.: Nauka, 1978, – 416p. (in Russian)

[26] R. Paul: Modelling, trajectories planning and movement control of robot-manipulator. – M.: Nauka, 1976, – 103p. (in Russian)

[27] E.P. Popov, A.F. Verechagin, S.L. Zenkevich: Manipulation robots: dynamics and algorithms. – M.: Nauka, 1978, – 400p. (in Russian)

[28] M. Shahinpoor: Course of robotics. – M.: Mir, 1990, – 527p. (in Russian)

[29] F.L. Thernousko, N.N. Bolotnik, V.G. Gradetckii: Manipulation robots: dynamics, control, optimisation. – M.: Nauka, 1989, – 368p. (in Russian)

[30] M.L. Tsetlin: Investigation on automaton theory and biological systems modelling. – M.: Nauka, 1969, – 316p. (in Russian)

[31] J.J. Uicker: Dynamic force analysis of spatiallinkages ASME J. of Applied Mechanics. 1967, pp. 418-424.

[32] I. Vittenbutg: Dynamics of rigit body systems. – M.: Mir, 1980, – 292p. (in Russian)

[33] M. Vukobratovich: Valking robots ans anthropomorphic mechanisms. – M.: Mir, 1976, – 543p. (in Russian)

[34] M. Vukobratovich, D. Stokich: Control of manipulation robots: theory and application. – M.: Nauka, 1985, – 384p. (in Russian)

[35] M. Vukobratovich, D. Stokich, N. Kirchanskii: Nonadaptive and adaptive control of manipulation robots. – M.: Mir, 1989, – 376p. (in Russian)

[36] M.W. Walker, D.E. Orin: Efficient dynamic computer simulation of robot mechanisms ASME J. Dyn. Syst., Meas.,and Contr. 1982, Vol. 104, No. 4. pp. 205-211.

[37] C.H. Wu, R.P. Paul: Resolved motion force control of robot manipulator IEEE Trans. Syst., Man. and Cybern. 1982, Vol. 12, No. 3, pp. 266-276.

[38] G. Zeng, A. Hemami: An overview of robot force control Robotica. 1997, Vol. 15. pp. 473-482.

[39] Artificial intelligence in engineering: robotics and processes. Ed. Gero J.S., Elsevier, 1988, – 403p.

[40] MACSYMA Reference Manual. The Mathlab Group, Laboratory for Computer Science, MIT, Jan. 1983, – 380p.

[41] Microprocessors in robotics and manufacturing systems. Ed. by S.G. Tzafestas. London, Kluwer Academic Pr., 1991, – 409p.