

Master's Thesis:

**Matrix Analytic Analysis of Delays in Communication
Systems.**

The topic of the Thesis has been confirmed
by the Departmental Council of the
Department of Information Technology
on 15 October 2003.

Supervisor: Valery Naumov, Professor
naumov@lut.fi

Examiner: Oleg Chistokhvalov
chistokh@lut.fi

Author: Sergey Boldyrev

Address: Ruskonlahdenkatu 13-15 C2,
Lappeenranta, Finland, 53850.

Phone number: +358 50 545-34-08
+7 911 242-08-33

E-Mail: boldyrev@lut.fi

Lappeenranta
2003

TIIVISTELMÄ

Lappeenrannan Teknillinen Yliopisto
Tietotekniikan Osasto

Sergey Boldyrev

Matrix Analytic Analysis of Delays in Communication Systems.

Diplomityö

2003

71 sivua, 32 kuvaa

Ohjaaja ja 1 tarkastaja: Professori Valery Naumov

2 tarkastaja: Oleg Chistokhvalov

Avainsanat: Quasi Birth Death prosessi solver, matriisi-geometrinen analyysi, Markov prosessi, QBD prosessi.

Tässä päättötyössä annetaan kuvaus kehityksestä sovelluksesta Quasi Birth Death processien ratkaisuun. Tämä ohjelma on tähän mennessä ainutlaatuinen ja sen avulla voi ratkaistasarjan tehtäviä jaita tervitaan kommunikaatio systemien analyysiin. Maunittuun sovellukseen on annettu kuvaus ja määritelmä. Lyhyt kuvaus teisesta sovelluksesta Quasi Birth Death prosessien tehtävien ratkaisuun on myös annettu.

ABSTRACT

Lappeenranta University of Technology
Department of Information Technology

Sergey Boldyrev

Matrix Analytic Analysis of Delays in Communication Systems.

Thesis for the Degree of Master of Science in Technology

2003

71 pages, 32 figures, 1 table and 3 appendices.

Supervisor: Professor Valery Naumov

Examiner: Oleg Chistokhvalov

Keywords: Quasi Birth Death processes solver, matrix-geometric analysis, Markov process, QBD process.

In this paper a description of developed application for solving a Quasi Birth Death processes system was given. This program is unique so far and allows to solve a set of tasks needed for communication systems analysis. The description and specification of mentioned application is given. A brief description of another applications for Quasi Birth Death processes tasks solution is provided in this paper also. Some notes about implemented algorithms improvement ways are discussed.

TABLE OF CONTENTS

TIIVISTELMÄ.....	2
ABSTRACT	3
TABLE OF CONTENTS.....	4
ACKNOWLEDGEMENTS.....	6
1. INTRODUCTION	7
2. PROJECT DEFINITION AND MATHEMATICAL BACKGROUND.	8
2.1 Project definition and goals	8
2.2 Used algorithms	9
3. EXISTING TOOLS OVERVIEW	14
3.1 MAMSolver tool.....	14
3.2 Telpack (XTelpack) tool.....	16
3.3 BoDyTool application.....	19
4. PROGRAM OVERVIEW AND USER INTERFACE DESCRIPTION.....	20
4.1 Main menu bar and main toolbar	21
4.1.1 File menu item	22
4.1.2 Settings menu item.....	25
4.1.3 Calculations menu item.....	26
4.1.4 View menu item	29
4.1.5 Help menu item	30
4.2 Main work area	30
4.2.1 Drawing a new QBD process.....	30
4.2.2 QBD process representation and it's operations	30
4.2.3 Inner state matrixes addition to the QBD process.....	30
4.2.4 Boundary condition matrixes addition to the QBD process.....	30
4.2.5 Connection description.....	30
4.2.6 QBD correctness checking procedure.....	30
4.3 Information window	30
5. PROGRAM SPECIFICATION, FUNCTIONALITY AND IMPLEMENTED ALGORITHMS.	30
5.1 Program architecture.....	30
5.2 BoDyTool.exe module file description.....	30
5.3 MatrixDLL.dll module file description	30
5.4 QuadEqSolver.dll module file description.....	30
5.5 QBDSolver.dll module file description	30
5.6 Dynamic Link Libraries description	30

6. PRACTICAL ASSIGMENT.	30
7. CONCLUSIONS	30
8. FUTURE WORK.....	30
REFERENCES	30
APPENDIX 1. An example of input file with data for one QBD process.....	30
APPENDIX 2. An example of input file with boundary data for one QBD process.	30
APPENDIX 3. An example of output file with solution result.....	30

ACKNOWLEDGEMENTS

I would like to acknowledge my supervisor Valery Naumov for providing an interesting topic for me and for his supervising during application development and writing Master Thesis process. Special thanks to Denis Dyakov for his great ideas about method implementation, for most part of algorithms descriptions and for very useful comments concerning the application interface. Also I would like to acknowledge the whole Laboratory of Telecommunications staff for very interesting time, spent here. Finally big gratitude to all of my friends those were worried about me and provided me with moral support after sleep-less nights.

1. INTRODUCTION

As different types of communication systems, like Internet, different types of cell phone's networks and more grows up rapidly one of the most important problem rising up is how to manage all of this systems in order to give the ability for each user to use them independently from the number of other users. The only way for future network improvement is to create some mathematical methods, which will allow to measure the performance of the given system in order to control it by using measurements results. But due of big physical size of mentioned networks sometimes it is very hard to obtain a correct result in a suitable time, even by using the modern computers. Therefore some new analytic techniques should be developed and implemented into real working applications for measurement tasks solution.

Some of the mentioned above mathematical methods already exists for different types of so-called queuing systems, but the most popular system for analysis is Quasi Birth Death process. This technique covers a huge domain of the existing tasks, have a lot of already implemented tasks solution algorithms and steel holds the scientists attention to itself.

Unfortunately all of these algorithms allow to solve a task for systems which can be represented as a single Quasi Birth Death process or as a set of QBD processes with some applied restrictions. This paper describes new technique which allows to avoid most of such restrictions as well as an implementation of this technique into application for solving an existing real-life tasks.

2. PROJECT DEFINITION AND MATHEMATICAL BACKGROUND.

In this section the project definition will be given. Note, that this chapter is based mostly on the materials, given in [1], therefore only a brief description of mathematical background will be given. Also the author assumes that the reader is already familiar with QBD processes [2] and some of the existing so far techniques, like as cyclic reduction algorithm for QBD problems [3] or Efficient Technique for the Analysis of QBD processes by Aggregation (ETAQA) [4].

The developed tool is based on the modified matrix geometric solution for finite QBD processes method, because of it has one of the lowest time complexity values with a very good precision. The given method was proposed by Valery Naumov in [5], therefore it is highly recommended to be familiar with his work too.

2.1 Project definition and goals

In this section materials and designations from [1] were used.

As it was mentioned above some of the queuing systems can not be described as a single QBD process, but they can be represented in a form of a set with different processes in it, where each process has a QBD structure. Therefore it is possible to obtain a solution (steady state probability vector) for the whole system by applying the Naumov's algorithm [5] to each QBD process in the described set.

Let's assume, that each of the QBD processes has level-independent structure, all of them have different number of levels, different matrixes A , B and C , different internal dimension (the dimension of matrix B) and the transitions can be only between boundary states of different processes as well as inside of each process.

The simplest system of such type which contains only two QBD processes is represented on the following figure (here the common square represents the transition between two QBD's):

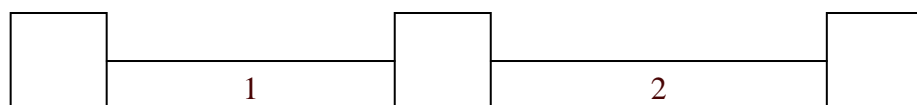


Figure 1. An example of queuing system with two QBD processes.

As it was shown in [1] the modified matrix geometric solution for finite QBD processes method can be extended to such type of tasks too and the goal of this project was to develop an application which will be able to solve a tasks for such queuing system by using this improvement.

2.2 Used algorithms

In this section materials and designations from [1] were used.

As in was shown in [1] each QBD process in the described queuing system can be defined by its length $m^{(i)}$, the number of phases $n^{(i)}$, the inner transition rate matrixes $A^{(i)}, B^{(i)}, C^{(i)}$ and the boundary matrixes $A_0^{(i)}, B_0^{(i)}, C_0^{(i)}, A_{m^{(i)}-1}^{(i)}, B_{m^{(i)}}^{(i)}, C_{m^{(i)}}^{(i)}$ with two vectors $\bar{b}_1^{(i)}, \bar{b}_2^{(i)}$ and one scalar $b_0^{(i)}$ added, which specify the additional arriving process. The i parameter describes the number of QBD process to which those parameters belongs.

In addition it is necessary to specify four transition rate matrixes as it was done in [1]:

M_{ij} - from state $(0,k)$ of the i -th process to the state $(0,h)$ of the j -th,

where $k = 0, \dots, n^{(i)}, h = 0, \dots, n^{(j)}, i, j = 1, \dots, m, i \neq j$

L_{ij} - from state $(0,k)$ of the i -th process to the state $(m^{(j)}, h)$ of the j -th,

where $k = 0, \dots, n^{(i)}, h = 0, \dots, n^{(j)}, i, j = 1, \dots, m, i \neq j$

N_{ij} - from state $(m^{(i)}, k)$ of the i -th process to the state $(0,h)$ of the j -th,

where $k = 0, \dots, n^{(i)}, h = 0, \dots, n^{(j)}, i, j = 1, \dots, m, i \neq j$

K_{ij} - from state $(m^{(i)}, k)$ of the i -th process to the state $(m^{(j)}, h)$ of the j -th,

where $k = 0, \dots, n^{(i)}, h = 0, \dots, n^{(j)}, i, j = 1, \dots, m, i \neq j$

The parameter $n^{(i)}$ represents the dimension of square matrix $B^{(i)}$ and parameter m describes the total number of QBD processes in the given system.

Note, that here the designations from [1] were used, i.e.

$$B_0^{(i)} = \begin{pmatrix} b_0^{(i)} & b_1^{(i)} \\ b_2^{(i)} & \underline{B}_0^{(i)} \end{pmatrix}, A_0^{(i)} = \begin{pmatrix} 0 \\ A^{(i)} \end{pmatrix}, C_0^{(i)} = \begin{pmatrix} C^{(i)} \\ 0 \end{pmatrix} \quad (2.1)$$

and after that the $\underline{B}_0^{(i)}$ was denoted as $B_0^{(i)}$.

According to [1] it is possible to receive a square generator matrix Q for the whole queuing system by collecting the appropriate properties (matrixes in other words) of each QBD process, therefore this task can be solved directly by using such transition matrix Q with normalizing condition:

$$\begin{cases} \bar{p}Q = \bar{0}, \\ \bar{p}\bar{e} = 1. \end{cases} \quad (2.2)$$

where \bar{e} is a column vector of all ones, $\bar{0}$ is a vector of zeros and unknown vector \bar{p} which describes the required steady states distribution for the whole system. Such solution method was also implemented in the given application and calls as “*Normal calculation*” in the program, but such type of calculations does not allow to perform fast calculation process and was developed only as addition to the improved calculations method.

Let’s return to the main goal of this section – to the description of the modified matrix geometric solution for finite QBD processes method extended to our task. As it was described in [1] the required solution can be represented as:

$$\bar{p}_k^{(i)} = \left(\bar{f}^{(i)} F_{(i)}^k + \bar{g}^{(i)} G_{(i)}^{m^{(i)}-k} \right) \Phi_{(i)}^\perp + y_{(i)} q_{(i)} h_k^{(i)} \bar{\pi}^{(i)}, \quad \begin{cases} k = 0, \dots, m^{(i)}; \\ i = 1, \dots, m. \end{cases} \quad (2.3)$$

where

$$\begin{aligned} q_{(i)} &= \left(\bar{\pi}^{(i)} C^{(i)} \bar{e}^{(i)} \right)^{-1}, \\ y_{(i)} &= \begin{cases} 0 & \text{if } \det \Phi_{(i)} \neq 0, \\ 1 & \text{if } \det \Phi_{(i)} = 0. \end{cases}, \\ \Phi_{(i)}^\perp &= \begin{cases} \Phi_{(i)}^{-1} & \text{if } \det \Phi_{(i)} \neq 0, \\ \Phi_{(i)}^\# & \text{if } \det \Phi_{(i)} = 0 \end{cases}, \end{aligned} \quad (2.4)$$

$$\bar{f}^{(i)} = \Phi_{(i)}^\# A^{(i)} \bar{e}^{(i)} - F_{(i)} \Phi_{(i)}^\# C^{(i)} \bar{e}^{(i)}, \quad \bar{g}^{(i)} = \Phi_{(i)}^\# C^{(i)} \bar{e}^{(i)} - G_{(i)} \Phi_{(i)}^\# A^{(i)} \bar{e}^{(i)}.$$

$$h_k^{(i)} = h_0^{(i)} + k \left(\bar{f}^{(i)} \bar{e}^{(i)} \right) + \sum_{j=0}^{k-1} \bar{f}^{(i)} F_{(i)}^j \bar{\phi}^{(i)} - \sum_{m^{(i)}-k}^{m^{(i)}-1} \bar{g}^{(i)} G_{(i)}^j \bar{\gamma}^{(i)}. \quad (2.5)$$

The $F_{(i)}, G_{(i)}$ matrixes are the minimal nonnegative solutions of matrix quadratic equations:

$$A^{(i)} + B^{(i)} F_{(i)} + C^{(i)} F_{(i)}^2 = \mathbf{0}, \quad A^{(i)} G_{(i)}^2 + B^{(i)} G_{(i)} + C^{(i)} = \mathbf{0} \quad (2.6)$$

and $\Phi_{(i)}$ for each of QBD processes defined as

$$\Phi = A^{(i)} G_{(i)} + B^{(i)} + C^{(i)} F_{(i)} \quad (2.7)$$

also let’s introduce a general group inverse matrix $\Phi^\#$, those elements $a^\#_{ij}$ can be obtained by following algorithm (source matrix Φ denoted as A matrix here), according to [6]:

$$a_{ij}^{\#} = \frac{1}{u^2} \sum_{\substack{1 \leq w_1 < \dots < w_r \leq n \\ 1 \leq v_1 < \dots < v_r \leq n}} A \begin{pmatrix} v_1, \dots, j, \dots, v_r \\ w_1, \dots, i, \dots, w_r \end{pmatrix} A_{ij} \begin{pmatrix} w_1, \dots, i, \dots, w_r \\ v_1, \dots, j, \dots, v_r \end{pmatrix} \quad (2.8)$$

where $u = \sum_{1 \leq w_1 < \dots < w_r \leq n} A \begin{pmatrix} w_1, \dots, w_r \\ w_1, \dots, v_r \end{pmatrix} \neq 0$

The last one undefined yet vector is $\vec{\pi}^{(i)}$ which describes the stationary probability of $A^{(i)} + B^{(i)} + C^{(i)}$

Summing up (2.3) – (2.8) to obtain steady states probabilities vectors for each of Quasi Birth Death processes it is necessary to find two vectors $f^{(i)}, g^{(i)}$ and one scalar $h_0^{(i)}$ for each of the QBD's.

According to [1] the following normalizing equation should be used:

$$p_{00} + \sum_{i=1}^m \left(f^{(i)} \vec{\alpha}_{(i)} + h_0^{(i)} \varkappa_{(i)} + g^{(i)} \vec{\beta}_{(i)} \right) = 1, \quad (2.9)$$

where

$$\begin{aligned} \vec{\alpha}_{(i)} &= \sum_{k=0}^{m^{(i)}} (F_{(i)}^k) \Phi_{(i)}^{\perp} \vec{e}^{(i)} + y_{(i)} \left(q_{(i)} \frac{m^{(i)} (m^{(i)} + 1)}{2} \vec{e}^{(i)} + q_{(i)} \sum_{k=0}^{m^{(i)}} \sum_{j=0}^{k-1} (F_{(i)}^j) \vec{\phi}_{(i)}^j \right) \\ \vec{\beta}_{(i)} &= \sum_{k=0}^{m^{(i)}} (G_{(i)}^{m^{(i)}-k}) \Phi_{(i)}^{\perp} \vec{e}^{(i)} - y_{(i)} \left(q_{(i)} \sum_{k=0}^{m^{(i)}} \sum_{j=m^{(i)}-k}^{m^{(i)}} (G_{(i)}^j) \vec{\gamma}_{(i)}^j \right), \\ \varkappa_{(i)} &= y_{(i)} q_{(i)} (m^{(i)} + 1), \end{aligned}$$

with the following set of boundary equations

$$p_{00} \sum_{i=1}^m b_0^{(i)} + \sum_{i=1}^m \left(f^{(i)} \vec{\delta}_{(i)} + h_0^{(i)} \varepsilon_{(i)} + g^{(i)} \vec{\eta}_{(i)} \right) = 0 \quad (2.10)$$

where

$$\begin{aligned} \vec{\delta}_{(i)} &= \Phi_{(i)}^{\perp} b_2^{(i)}, \\ \varepsilon_{(i)} &= y_{(i)} q_{(i)} \vec{\pi}_{(i)} b_2^{(i)}, \\ \vec{\eta}_{(i)} &= G_{(i)}^{m^{(i)}} \Phi_{(i)}^{\perp} b_2^{(i)}. \end{aligned}$$

$$\begin{aligned}
& p_{00}\vec{b}_1^{(i)} + \vec{f}^{(i)}D_{(i)}^1 + \vec{g}^{(i)}D_{(i)}^2 + h_0^{(i)}\vec{d}_{(i)}^3 + \\
& + \sum_{k=1, k \neq i}^m \left[\vec{f}^{(k)}D_{(k)}^1 + \vec{g}^{(k)}D_{(k)}^2 + h_0^{(k)}\vec{d}_{(k)}^3 \right] = \mathbf{0},
\end{aligned} \tag{2.11}$$

where

$$\begin{aligned}
D_{(i)}^1 &= \Phi_{(i)}^\perp B_0^{(i)} + \left(F_{(i)}\Phi_{(i)}^\perp + y_{(i)}q_{(i)}\vec{e}^{(i)}\vec{\pi}_{(i)} \right) C_0^{(i)} \\
D_{(i)}^2 &= G_{(i)}^{m^{(i)}}\Phi_{(i)}^\perp B_0^{(i)} + G_{(i)}^{m^{(i)}-1}\Phi_{(i)}^\perp C_0^{(i)} \\
\vec{d}_{(i)}^3 &= y_{(i)}q_{(i)}\vec{\pi}_{(i)} \left(C_0^{(i)} + B_0^{(i)} \right) \\
D_{(k)}^1 &= \Phi_{(k)}^\perp M_{k,i} + \left(F_{(k)}\Phi_{(k)}^\perp + y_{(k)}q_{(k)} \left(m^{(k)}\vec{e}^{(k)} + \sum_{j=0}^{m^{(k)}-1} \left(F_{(k)}^j \right) \vec{\phi}_{(k)} \right) \vec{\pi}_{(k)} \right) N_{k,i} \\
D_{(k)}^2 &= G_{(k)}^{m^{(k)}}\Phi_{(k)}^\perp M_{k,i} + \left(\Phi_{(k)}^\perp - y_{(k)}q_{(k)} \sum_{j=0}^{m^{(k)}-1} \left(G_{(k)}^j \right) \vec{\gamma}_{(k)}\vec{\pi}_{(k)} \right) N_{k,i} \\
\vec{d}_{(k)}^3 &= y_{(k)}q_{(k)}\vec{\pi}_{(k)} \left(M_{k,i} + N_{k,i} \right).
\end{aligned}$$

$$\vec{f}^{(i)}U_{(i)}^1 + \vec{g}^{(i)}U_{(i)}^2 + h_0^{(i)}\vec{u}_{(i)}^3 + \sum_{k=1, k \neq i}^m \left[\vec{f}^{(k)}U_{(k)}^1 + \vec{g}^{(k)}U_{(k)}^2 + h_0^{(k)}\vec{u}_{(k)}^3 \right] = \mathbf{0} \tag{2.12}$$

where

$$\begin{aligned}
U_{(i)}^1 &= \left(F_{(i)}^{m^{(i)}-1}\Phi_{(i)}^\perp + y_{(i)}q_{(i)} \left((m^{(i)}-1)\vec{e}^{(i)} + \sum_{j=0}^{m^{(i)}-2} \left(F_{(i)}^j \right) \vec{\phi}_{(i)} \right) \vec{\pi}_{(i)} \right) A_{m^{(i)}-1}^{(i)} + \\
& + \left(F_{(i)}^{m^{(i)}}\Phi_{(i)}^\perp + y_{(i)}q_{(i)} \left(m^{(i)}\vec{e}^{(i)} + \sum_{j=0}^{m^{(i)}-1} \left(F_{(i)}^j \right) \vec{\phi}_{(i)} \right) \vec{\pi}_{(i)} \right) B_{m^{(i)}}^{(i)} \\
U_{(i)}^2 &= \left(G_{(i)}\Phi_{(i)}^\perp - y_{(i)}q_{(i)} \sum_{j=1}^{m^{(i)}-1} \left(G_{(i)}^j \right) \vec{\gamma}_{(i)}\vec{\pi}_{(i)} \right) A_{m^{(i)}-1}^{(i)} + \\
& + \left(\Phi_{(i)}^\perp - y_{(i)}q_{(i)} \sum_{j=0}^{m^{(i)}-1} \left(G_{(i)}^j \right) \vec{\gamma}_{(i)}\vec{\pi}_{(i)} \right) B_{m^{(i)}}^{(i)} \\
\vec{u}_{(i)}^3 &= y_{(i)}q_{(i)}\vec{\pi}_{(i)} \left(A_{m^{(i)}-1}^{(i)} + B_{m^{(i)}}^{(i)} \right) \\
U_{(k)}^1 &= \Phi_{(k)}^\perp L_{k,i} + \left(F_{(k)}^{m^{(k)}}\Phi_{(k)}^\perp + y_{(k)}q_{(k)} \left(m^{(k)}\vec{e}^{(k)} + \sum_{j=0}^{m^{(k)}-1} \left(F_{(k)}^j \right) \vec{\phi}_{(k)} \right) \vec{\pi}_{(k)} \right) K_{k,i} \\
U_{(k)}^2 &= G_{(k)}^{m^{(k)}}\Phi_{(k)}^\perp L_{k,i} + \left(\Phi_{(k)}^\perp - y_{(k)}q_{(k)} \sum_{j=0}^{m^{(k)}-1} \left(G_{(k)}^j \right) \vec{\gamma}_{(k)}\vec{\pi}_{(k)} \right) K_{k,i} \\
\vec{u}_{(k)}^3 &= y_{(k)}q_{(k)}\vec{\pi}_{(k)} \left(L_{k,i} + K_{k,i} \right).
\end{aligned}$$

Now it is possible to write an implemented in application “*Improved calculations*” algorithm step-by-step:

1. Perform any needed precalculations like calculate $F_{(i)}, G_{(i)}$ matrixes and some more needed in future parameters by using formulas (2.4), (2.6), (2.7) and (2.8).
2. Form system of linear equations by using the (2.10), (2.11) and (2.12)

3. If some of the QBD processes have singular matrix Φ , for each of such QBD's the following equation should be added into obtained at second step system:

$$f^{(i)}\bar{e}^{(i)} + g^{(i)}\bar{e}^{(i)} = 0. \quad (2.13)$$

4. Add a normalizing condition (2.9) to the described system of linear equations.
5. Totally there will be $1+1+q+2\sum_{i=1}^m n^{(i)}$ equalities in the given system of linear equations, where q describes the total number of QBD processes which have $\det(\Phi) = 0$ or the number of equations (2.13) added to the system. Let's denote the matrix of coefficients for our system as Q' . Note that the rank of this matrix will be $1+q+2\sum_{i=1}^m n^{(i)}$ which equals to number of unknown variables (for those QBD processes which have nonsingular matrixes Φ we can set the values of $h_0^{(i)}$ to zero, because the solution procedure for such QBD's does not require them)
6. After solving a system of linear equations the following unknown vectors and scalars will be obtained: $p_{00}, h_0^{(i)}, f^{(i)}$ and $g^{(i)}$ where i describes the number of appropriate QBD process.
7. By using (2.3) and (2.5) it is possible to obtain the required steady states probabilities vectors for each of Quasi Birth Death processes.
8. Solution is done!

Note that this algorithm was adapted for using in computer based calculations, therefore it does not require any symbolical solution procedures and can be implemented without any troubles.

3. EXISTING TOOLS OVERVIEW

In this section the description of some of most well-known applications will be given as well as the comparison of them with BoDyTool program.

3.1 *MAMSolver tool*

In this section the materials from [7] were used.

MAMSOLVER is a software tool that provides efficient implementations of the state-of-the-art algorithms of the matrix-analytic methodology including the matrix-analytic and matrix-geometric solution techniques for M/G/1-type and GI/M/1-type processes, respectively. MAMSOLVER also provides an implementation of the ETAQA methodology. Although, this exposition of matrix-analytic methods and ETAQA considers only CTMCs, MAMSOLVER provides solutions for both DTMCs and CTMCs.

The matrix-analytic algorithms that were taken into consideration are defined in terms of matrices, making matrix manipulations and operations the basic elements of the tool. The input to MAMSolver, in the form of a structured text file, is the finite set of matrices that accurately describe the process to be solved. Since there are different algorithms that provide solution for the same process, the user specifies the method to be used. However, several tests are performed within the tool to insure that special cases are treated separately and therefore more efficiently. MAMSOLVER is implemented in C++, and classes define the basic components of the type of processes under consideration.

Matrix is the module that implements all the basic matrix operations such as matrix assignments, additions, subtractions, multiplications, and inversions. For computational efficiency, the developers used well known and heavily tested routines provided by the Lapack and BLAS packages. Since solving a finite system of linear equations is a core function in matrix-analytic algorithms, MAMSolver provides several numerical methods depending on the size of the problem, i.e., the size of the coefficient matrices. For small-size problems exact methods such as LU-decomposition are used, otherwise the Lapack implementation of iterative methods such as GMRES and BiCGSTAB, are chosen.

Matrix-analytic modules handle both Continuous Time Markov Chain and Discrete Time Markov Chain processes. First these modules provide storage for the input matrices. In addition, these modules provide all the routines necessary for the implementation of the algorithms needed for solution of mentioned types of tasks. Both the data structures and routines of the matrix-analytic modules are based on the data-structures and routines provided by the matrix module. The high-level structure of MAMSOLVER is illustrated in following figure.

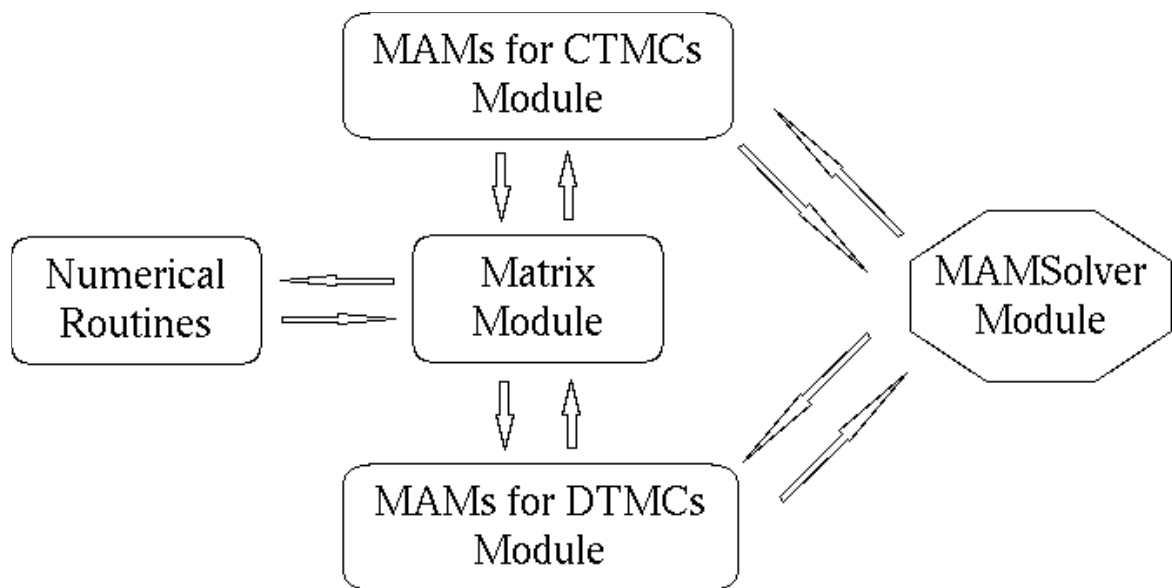


Figure 2: MAMSolver structure.

The solution of **QBD** processes, requires computation of the \mathbf{R} (and sometimes of \mathbf{G} depending on the solution algorithm). First the matrix \mathbf{R} is computed using the logarithmic reduction algorithm. For completeness, the developers provide also the classic numerical algorithm. The available solution methods for QBD processes are matrix-geometric and ETAQA-QBD.

GI/M/1 processes require the computation of the matrix \mathbf{R} . The classic matrix geometric solution is implemented to solve this type of processes. First the algorithm goes through a classic iterative algorithm to compute. Then, the tool computes the boundary part of the stationary probability vector. Since a geometric relation exist between vectors $\pi^{(i)}$ for $i \geq 1$, there is no need to compute the whole stationary probability vector.

M/G/1 processes require the computation of the matrix \mathbf{G} which is calculated using the classic iterative algorithm or the cyclic-reduction algorithm or the explicit one (if applied). The stationary probability vector is computed recursively using either the recursive

Ramaswami formula or its fast FFT version. ETAQA-M/G/1 is also implemented as an alternative for the solution of M/G/1 processes. A brief summary of the most important matrix-analytic algorithms implemented in MAMSOLVER is depicted at the following figure.

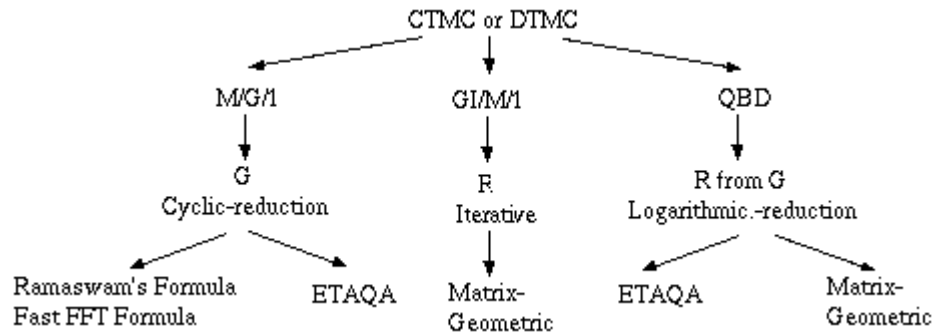


Figure 3: MAMSolver algorithms.

More information about this tool with detailed mathematical background, algorithms and methods description can be obtained at the official application homepage [8].

3.2 Telpack (XTelpack) tool

In this section the materials from [9] were used.

Telpack solves a rich set of stochastic models and queueing problems frequently encountered in teletraffic analysis. These fall into two categories as discrete- and continuous-state problems, and are listed below.

Discrete-state problems	Continuous-state problems
<ol style="list-style-type: none"> 1. G/M/1-type structured Markov chains 2. M/G/1-type structured Markov chains 3. QBD processes 4. Combined G/M/1-M/G/1 structure 5. Discrete G/G/1 queue 	<ol style="list-style-type: none"> 1. MAP/G/1 queue 2. PH/PH/1 queue 3. MMPP/G/1 queue 4. GI/G/1 queue 5. Fluid flow and Brownian motion models
Table 1: Telpack task types.	

Telpack employs state-of-the-art solution methodologies and numerical techniques to solve the problems listed above and specific forms of some of them. E.g., finite-level M/G/1 and QBD chains, level-dependent QBD chains, M/G/1 chains with multiple boundaries, and G/M/1, M/G/1 and QBD chains with non-canonical boundaries. Depending on the problem dimension, these solutions may indeed become computationally very demanding to obtain. Algorithmic aspects of the solution methods, matrix-algebraic operations, certain matrix factorizations, and invariant and deflating subspace computations constitute the crux of the overall numerical task for any given problem. Telpack makes exclusive use of the LAPACK/BLAS library routines to carry out these tasks.

Generally speaking, Telpack obtains stationary queue-length distributions for discrete-state problems, and stationary waiting time (or unfinished work) distributions for continuous-state problems. These solutions take matrix-geometric and matrix-exponential forms, respectively, for the two problem categories. In addition to the elements of such solutions, Telpack computes and outputs the moments of these distributions, as well as their tail behavior characterizations. In the case of discrete-state problems, it also computes and outputs detailed queue-length probability vectors, and aggregate and overflow probabilities as desired.

Telpack interacts with the user through ASCII text files. That is, it writes all and any output it computes to text files with distinctive extensions that identify the particular output type (for example, `.opr` for overflow probabilities), and it reads the problem description and data from an input file. The various outputs can optionally be directed to an m-file for use in further computations under Matlab if desired. Input files are necessarily formatted, and a simple, line-oriented command language is used to describe a particular problem and to enter associated numerical data. This command language offers various constructs for efficient entry of special matrices such as constant, diagonal, sparse, etc.

The Telpack application is installed on a 16-processor Sun Enterprise® 6500 server (400 Mhz UltraSPARC®, 16 GB memory, Solaris® 8) maintained at the School of Interdisciplinary Computing and Engineering, University of Missouri - Kansas City. It is accessible to anyone on a *best-effort, non-guaranteed 24/7* basis through XTelpack, the client program for Windows and Linux that you can download and use free of charge.

XTelpack is for the most part a graphical user interface to Telpack, written in Tcl/Tk. In addition to performing the client-side tasks of submitting a problem and retrieving the results, it does offer expert assistance in various forms to the user. Chief among these are the expert dialogs for preparing input files correctly. A Telpack input file (its format and what data it should contain) depends on the particular problem type at hand, and this may constitute difficulty for the user, probably the very first hurdle for the beginner. These expert dialogs walk the user through the necessary steps according to the given problem type. They are, however, practically useful only for limited problem dimensions since they expect the user to type numerical data. For large-scale problems, or for a prospective frequent user, it is possible to have the input files prepared in other ways, say, by a program.

Telpack is configurable by the user to take different run-time actions/decisions. There is, for example, a multitude of possible solution methods for most problem types. There are many option switches and configuration parameters; some of these are general, and some apply to a given problem type. XTelpack again walks the user through the necessary steps of filling in all this data as it relates to the particular problem at hand, and provides defaults or suggestions on the way.

XTelpack also offers a built-in graphing utility for plotting results comparatively. These plots can be exported as encapsulated postscript files.

More information about this tool with detailed users guide description can be obtained at the official application homepage [9], but, unfortunately, this website does not provide a detailed information about implemented algorithms and program architecture, therefore there is no any possibility to describe this program in all details.

3.3 *BoDyTool application*

Though the existing applications are able to solve a wide range of tasks, all of them (or at least those applications which are available for public use) are unable to find an answer in the described above set of tasks (i.e. tasks for QBD processes chain). Therefore it was decided to develop own application which does not have any analogue in the world so far for performing such calculations. The following constrains for future application were set:

- An application should be a stand alone program, i.e. it should not require any remote server for task calculations, like in case of XTelpack application.
- An application should have the components based architecture for future algorithms improvements without rebuilding the whole program.
- An application should work on any computer with Windows operation system installed on it.
- The default implemented algorithm for calculations should be Valery Naumov's "Modified Matrix Geometric Algorithm" [5].
- The application should have user interface with clear task representation.

The following hardware requires for running an application.

- Any Intel Architecture based CPU (the recommended CPU frequency is 1 Giga Hertz or higher)
- At least 128 MB of RAM.
- Any Windows family operating system.

Note, that the given application is able to solve only described type of tasks, therefore it will not be able to obtain an answer for GI/G/1 queue, for example. Therefore you should use any of the described above applications for solving such type of tasks.

4. PROGRAM OVERVIEW AND USER INTERFACE DESCRIPTION

In this section the description in details of application will be given as well as some features and useful tricks.

The first thing you should do for using the application is to start the program. To perform this operation the user has to select the folder where the copy of program was allocated and run the application by clicking on the file with “BoDyTool.exe” name. If it was the first program launching time you’ll see the following message:

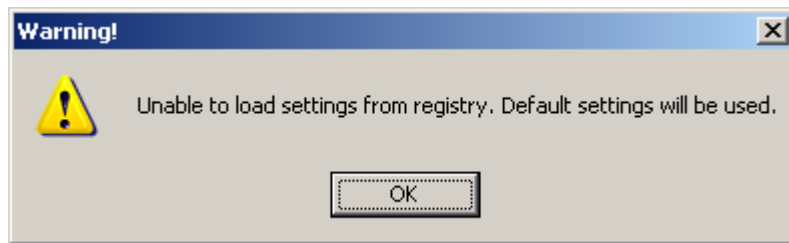


Figure 4: “Default setting will be used” notification message.

This message is default message during first startup and it means that the program was unable to load the setting from the registry, therefore the default setting will be used. In most cases you’ll not see this notification box more, but if not, you should ask the administrator of the computer to check your security rights to read and write to the Windows registry.

After a successful start of the program you’ll see the following picture on your desktop screen:

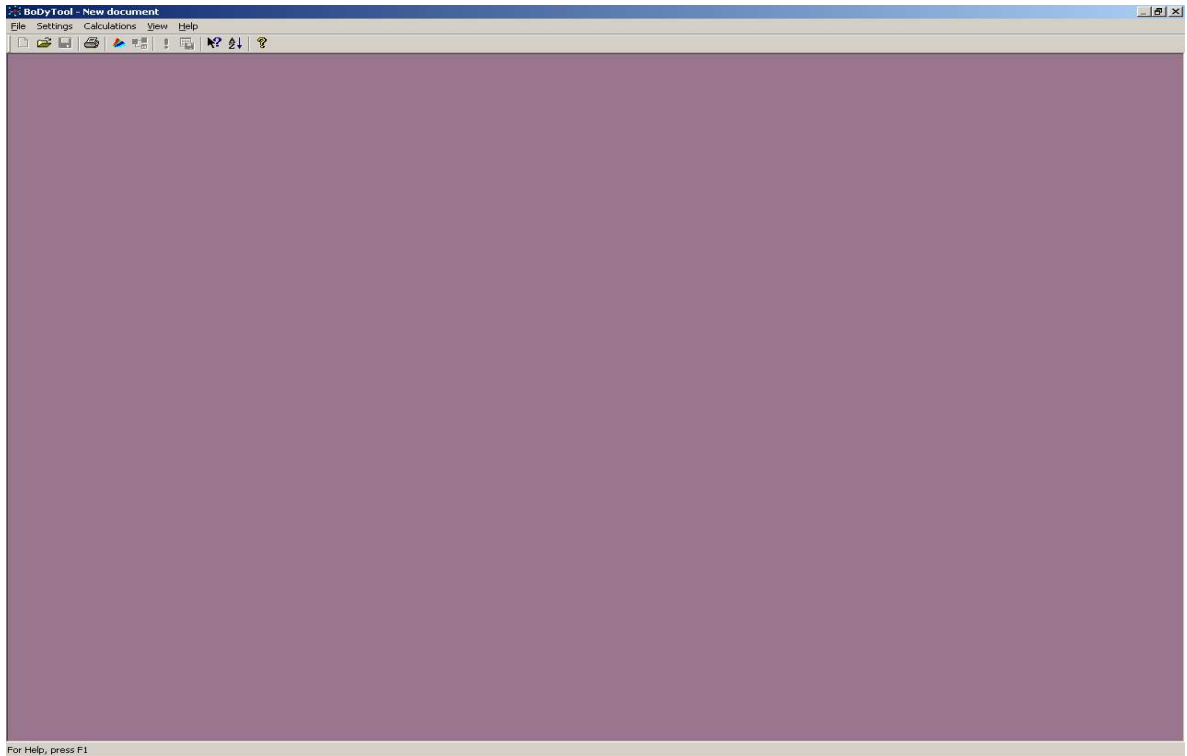


Figure 5: The main program screen.

This User Interface screen can be divided into three following part:

1. Main menu bar and main toolbar
2. Main work area
3. Information window

In this section we will describe all these interface parts in more details.

4.1 Main menu bar and main toolbar

The main menu bar let the user to perform all operations, which are available in the application. For better usability of the program some of these functions were implemented at the toolbar as well. Such implementation gives the ability not to select the needed option from the main menu, but select it directly from the toolbar. Also some hotkeys were implemented, which allows user to perform the commonly used operations by pressing the key combinations. You can see the implemented main menu bar and main toolbar of the following two pictures:

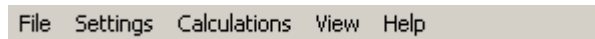


Figure 6: Main menu bar.



Figure 7: Main toolbar.

All of menu items will be described in the following sections.

4.1.1 File menu item

The first item of the menu bar is “File” section.

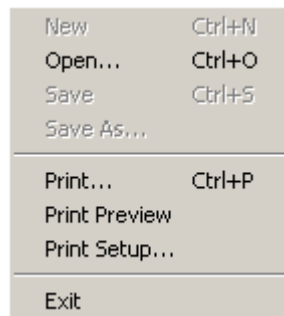




Figure 8: File menu section.

This section of menu allows user to create new task, load an existing task, save modified task to the disk or print it by printer. There are the following items in this menu:

- “*New*” – This menu function allows user to create a new document. If any task already exists in the main work area the confirmation box will be shown and new task will be created only after user confirmation. You can also perform this procedure by pressing the “*New*” button  on the toolbar. The hotkey for this menu item is “Ctrl + N” keys combination.
- “*Open...*” – This main menu item allows user to open an existing task. After pressing the “*Open...*” button you’ll see the standard built-in Windows “*Open file*” dialog from which you should select the appropriate file with “.bdp” extension which contains previously saved task. After needed file was selected press the “*Open*” button in the file dialog to open it and you’ll see your task graph at the main work area, if file contains the correct task data, or you’ll get the error box with error explanation, if not. You can also perform this procedure by pressing the

“Open” button  on the toolbar. The hotkey for this menu item is “Ctrl + O” keys combination.

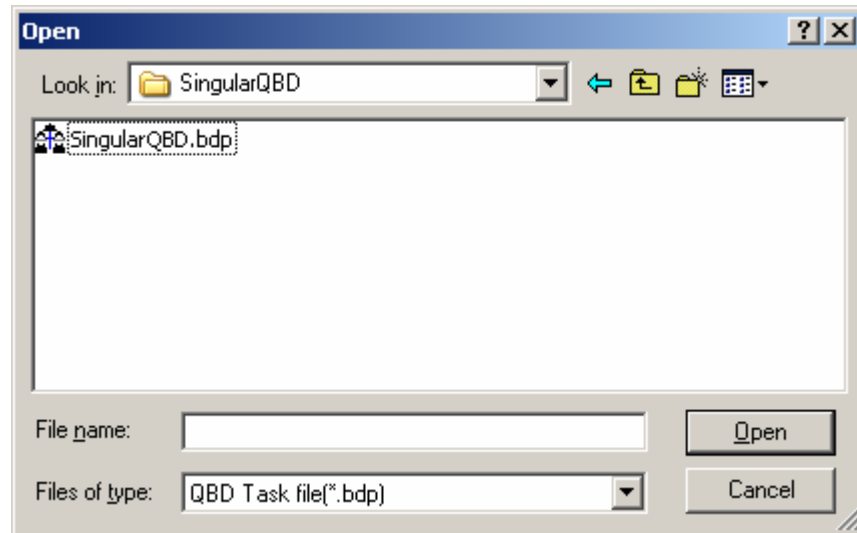



Figure 9: The open task dialog.

- “Save” – This main menu function allows user to save a modified existing task. In case the name of the file was not specified yet (for example if task was not saved yet) the function “Save as...” will be performed instead of function “Save”, otherwise the program simply stores the task changes to the specified before location. You can also perform this procedure by pressing the “Save” button  on the toolbar. The hotkey for this menu item is “Ctrl + S” keys combination.
- “Save as...” – This menu item allows user to save to the disk the existing task with another name. After pressing the “Save” button you’ll see the standard built-in Windows “Save file as...” dialog where you should enter new name for your file to the “File name” field, select proper extension for new file from the “Files of type” listbox and press the “Save” button. Note, that the default extension for the files, which contain stored tasks is “.bdp”. If this operation was performed successfully new file will be created at the specified location, otherwise you’ll see the error message window with error explanation.

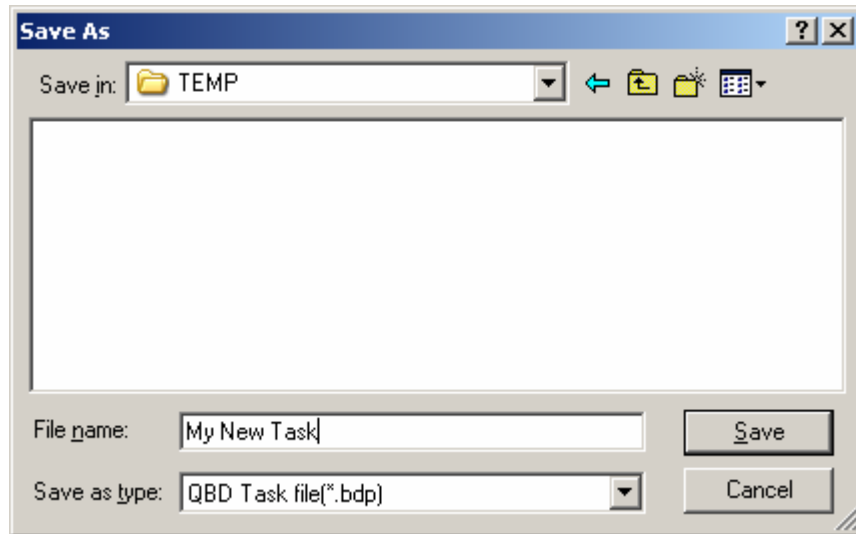



Figure 10: The “Save as...” task dialog.

- The “*Print...*” menu item allows user to print the current task on printer. After pressing this button user will see the standard built-in Windows “*Print...*” dialog, where you should specify the needed options and press “OK” button. If all printing operations were performed successfully the selected printer will begin to print, otherwise you’ll see the error window with error description. You can also perform this procedure by pressing the “*Print*” button  on the toolbar. The hotkey for this menu item is “Ctrl + P” keys combination.
- “*Print preview*” – This main menu option is for preliminary output for a printing result, i.e. after selection this item the user will see his task displayed as it would appear when printed. When you choose this command, the main window will be replaced with a print preview window in which one or two pages will be displayed in their printed format. If preview result will satisfy you, you can initiate the printing procedure by selecting the “*Print...*” item.
- After choosing the “*Print setup...*” command user will see the dialog, where it is possible to specify some printing parameters, like as paper size, paper orientation, needed printer and some more printer setting which depends of the printer type. All this setting will be permanent for all of next printing, unlike the settings in the “*Print...*” dialog which are valid only for current printing procedure.
- The last one item in the File menu section is “*Exit*” command, which closes the application. In case you’ve not saved your task yet you’ll see the information

message window, which allow you to save or not to save the changes to the file in order to prevent the lost of data. After that the application will be closed.

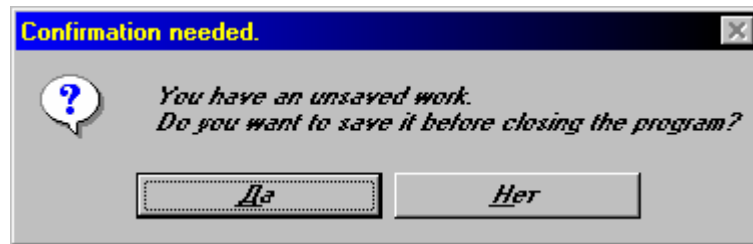


Figure 11: The message window before application closing.


4.1.2 Settings menu item

The next one item of the menu bar is “Settings” section, which contains only one command



Figure 12: Settings menu section.

This menu section allows user to change global program parameters. The current version of the program has only one function in this section:

- “Set Background” menu item allows user to change the background color of the main work area. After activating this command the user will see the standard built-in Windows “Color...” dialog where it is possible to select the appropriate color for program. After pressing the “OK” button all the elements in main work area will be redrawn with new color. Note, that all elements in the work area, for example, different QBDs, have their own colors which calculates to be in contrast with background color automatically, therefore you’ll be able to see your graph in work area whichever background flooding paint was not selected. You can also perform the color changing procedure by pressing the “Set new background color”  button on the toolbar.

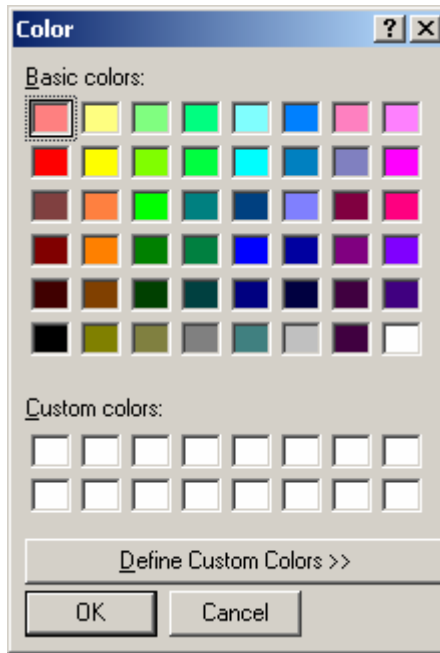


Figure 13: Default colour choosing dialog.

4.1.3 Calculations menu item

The third and the most important item in the menu bar is “Calculations” section.

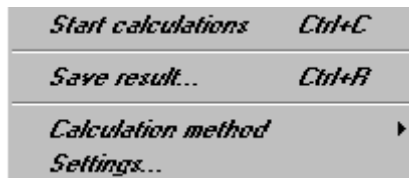


Figure 14: Calculation menu section.

This menu section allows user to set a calculation parameters, choose a type of calculations, obtain a solution for current task and save result to a file. There are the following items in this dropdown menu:

- ”*Start calculations*” command allows user to obtain an answer of the task. After all the parameters of current job were entered, the user can initiate the calculation process by choosing this menu item. If needed task was created correctly the application will begin to solve it by using selected before calculation method (see “*Calculation method*” dropdown menu description) otherwise you’ll see the error window with detailed error description in it (note, that the calculation will not be started until all errors will not be fixed). The calculation progress shows in the special progress bar situated at the center of main work area. After the progress bar value reach 100% your solution will be obtained and user can observe it at the special information window (see Information


window description for more details). Sometimes it's possible that some windows with messages of notifications will popup during calculation process. It's a normal situation which means that application needs the user response to one or another question. You can also start the calculation procedure by pressing the "Startup the calculations" button  on the toolbar or by pressing the hotkey combination "Ctrl + C".



Figure 15: A typical error message window after pressing the "Start calculations" button.

- After task result was obtained the user can store it to the file by choosing "Save result..." menu item. This command opens the standard built-in Windows "Save file as..." dialog where you can change or leave generated by program name for your new file at the "File name" field, also you can select appropriate extension for your result file from the "Files of type" list box and press the "Save" button.. If this operation was not performed successfully you'll see the error message window with error explanation, otherwise new file will be created at the specified location. The result file always has the following structure (see an example of such file in the appendixes section):
 - **"The probability, that system is in idle condition:"** - This string and value below it describes the probability that whole system is in idle condition.
 - **"----->>>-----<<<<-----"** – This is a separator string, which separates the result section for current QBD process from result sections of other QBD's. From this separator string the obtained answer segment starts.
 - **"The following results is for QBD # 1:"** – This string describes the number of QBD to which result following below belongs, the QBD process number 1 in our case.


- **”The mean length of queue:”** – This string and number following below this string describes the mean length of queue in the given QBD process. This value calculates as

$$l = \sum_{i=0}^m \bar{X}_i \cdot \bar{e} \cdot (i+1) \quad (4.1)$$

where \bar{e} is a column vector of all ones, m is a QBD length parameter for given Quasi Birth Death process and \bar{X}_i are the steady state probabilities vectors of it (note, that \bar{X}_k equals to $\bar{p}_k^{(i)}$ from section 2 of this paper).

- **”X0 :”** – This string and a column of values follows by this string describes the result vector X_0 in our case, or the steady state probability vector $\bar{p}_0^{(i)}$. The dimension of this vector equals to the dimension of B matrix of current QBD process. The last one vector in the current QBD result section will be X_m where m is a QBD length parameter for given QBD process. After that the result section which belongs to the current QBD process will be finished.

If there exists more than one QBD process in the current task the next one string will be separator string described above and another QBD process result section will begin until all of the QBD processes results will not be stored.

This function can be start by pressing the *”Save result”*  button on the taskbar or by pressing the hotkey combination *”Ctrl + R”* also.

- The *”Calculation method”* item contains dropdown sub-menu with two options in it: *”Normal calculations”* and *”Improved calculations”*. These two items describes two implemented calculations methods – normal method which simply solves the task $xQ = 0$ by using the Gauss linear system solution algorithm and improved method which solves the task by using the Naumov’s theory (note, that the implementation of the *”Improved calculations”* menu item depends of developers. By default the application contains a described in section 2 method in it, but third-side developers can easily rewrite it to another one algorithm by rewriting the appropriate Dynamic Loading Library). Due of memory requirements needed for processing a *”Normal calculations”* procedure it is not recommended to apply it to the tasks which contains QBD processes with QBD length parameter more than

300, because it will take very long time to calculate such task by using the Gauss method and will require a big amount of free memory. It's better to use the "Improved calculations" procedure for such big tasks, however for small tasks with small matrix dimensions and small QBD length parameters "Normal calculations" method works perfectly. See section 2 of this paper for more details about methods.

- The last one item in this menu section is "Settings..." command. After activating this command the "Settings" dialog box will appear where the user can set necessary accuracy of calculations. This accuracy will affect on all types of calculations, i.e. some iterative methods, number truncation during calculations and result values truncation. The accuracy text field supports two types of input values: normal input by typing decimal number with decimal point separation between whole and fractional parts, for example number "0.0001", and so-called scientific form where decimal number presented as some number multiplied by ten in some power, for example "1e-6". After inputting the appropriate precision of calculation the user should press "OK" button to save changes or press the "Cancel" button for leaving previous value unchanged.

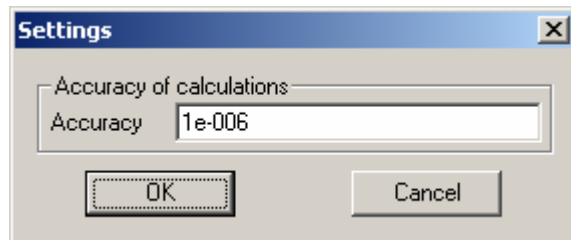


Figure 16: Settings dialog window.


4.1.4 View menu item

The next one item of the menu bar is "View" section, which contains the following submenu elements:



Figure 17: View menu section.

This menu section allows user to show or hide some elements of user interface and there are the following commands to perform such operations:

- By pressing the *”Toolbar”* menu item user can display or hide main toolbar. A check mark appears next to the menu item when the toolbar is displayed.
- Use the *”Status Bar”* command to display and hide the status bar, which describes the action to be executed by the selected menu item or depressed toolbar button. A check mark appears next to the menu item when the status bar is displayed.
- By selecting the *”Toggle info”* command the user can display or hide the Information window which contains the information about selected object in the main work area. A check mark appears next to the menu item when the Information window is displayed. You can also toggle the described window by pressing the *“Toggle info window”* button  on the toolbar. Note, that information window can be viewable only if there at least one element (QBD process actually) exists in the work area.


4.1.5 Help menu item


The last one item of the menu bar is “Help” section, which contains the following submenu elements:



Figure 18: Help menu section.

This menu section provides user with help topics about user interface, program functions and context sensitive help and there are the following commands to perform the given operations:

- The *”Help Topics”* menu command shows the index of all topics which are exists in the help system. After typing the necessary word in the search text field and pressing the *”Show”* button user will get an information about typed term, it’s description and some more useful information.
- The *”Help Index”* command consist of two parts. After selecting this menu item from main menu or pressing the *”Help Index”* button  on the toolbar the user will see a list of help topics grouped by subject. There are user interface description topics, program function description topics, calculation methods overview topics and so on. The second part of this command is so-called context-sensitive help which allows to

obtain a description of any element of user interface. To perform this operation press the "Help"  button on the toolbar – there the question sign should appear near your mouse pointer – and select any of the user interface element for which you want to get a help topic.

- "About BoDyTool..." menu item provides user with information about program, version and copyrights.

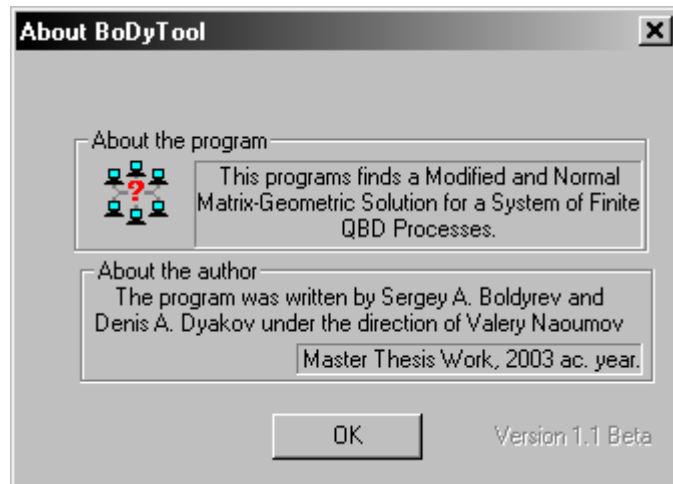


Figure 19: About BoDyTool dialog window.

4.2 Main work area

Main work area is the most important user interface area where all of application operations with current task produce. The viewable part of work area starts directly below toolbar and ends at the beginning of the status bar. There are two scrollbars implemented in this area, therefore the physical dimensions of main working area can reach the resolution up to 65535x65535 pixels totally which will be enough for all type of tasks. The functionality of main work area will be described in the following sections.

4.2.1 Drawing a new QBD process

The main work area of application contains a context-sensitive set of context menus and all of the operations are performing by using them. One of such operations is addition of new QBD process to the system. To perform it the user should click right-button of mouse once on any unused place of the work area. The following popup context menu will appear.

Add New QBD

Figure 20: Add new QBD context menu.

Select this menu item to add new QBD process to the system and you'll see new element of task shown at the following figure:



Figure 21: Adding new QBD process.

The start element with “*QBD level 0*” description has fixed location, but end element with “*QBD level ???*” has not and can be moved with mouse movement. After setting mouse pointer with QBD process end point clinging to it to the proper position press left mouse button again. If operation of addition was not performed successfully the user will see an error window message with error description, otherwise the standard built-in Windows “*Open file...*” dialog will appear where you should select the file, containing Quasi Birth Death process information, i.e. matrixes A, B, C and QBD process length information (file format description goes below) and press the “Open” button. If you don't want to add this information on current step it will be possible to add it later. Finally the user will see the following new QBD process with new number at the bottom (QBD process number 1 in our case):



Figure 22: New QBD process after successful addition.





4.2.2 QBD process representation and it's operations

As it was mentioned above, the typical QBD process implemented in this application should contain the following parameters: it's length m , the inner transition matrixes A, B, C and the boundary matrixes $A_0, B_0, C_0, A_{m-1}, B_m, C_m$ with a set of transition rate matrixes, therefore the generator matrix Q for one single QBD will be as follows:

$$Q = \begin{pmatrix} B_0 & A_0 & 0 & 0 & \dots & \dots & 0 \\ C_0 & B & A & 0 & \dots & \dots & 0 \\ 0 & C & B & A & \dots & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & C & B & A & 0 \\ 0 & 0 & 0 & 0 & C & B & A_{m-1} \\ 0 & 0 & 0 & 0 & 0 & C_m & B_m \end{pmatrix}. \text{ All of these parameters (except matrix } Q) \text{ have}$$

their representation in the program. This section of the paper will provide the information about all such representations and available functions for work with them.

The QBD process representation in the application can be divided into three parts, they are:

- 
 • Zero-level boundary conditions which are represented by  icon. The “QBD level 0” label means that in this part of Quasi Birth Death process representation the matrixes A_0, B_0, C_0 and all of the outer transitions (M_{ij} and L_{ij} matrixes namely) to another QBD processes are stored. Let call such QBD representation element as “Start node of QBD process” or simply “start node”. All of the functions which are available for zero-level boundary conditions are accessible by clicking right mouse button on the start node – the context menu will appear. If the user is not satisfactory with current icon placement he can press the left button on such interface element and, holding the left mouse button pressed, move it to the new position. After the appropriate position was reached, pressed button should be released. There is information about start node available also. It can be displayed by pressing the left button on it – start node of QBD process will become selected and there short info will appear at the information window (see Information window description for more details).
- 
 • m -level boundary conditions which are represented by  icon. The “QBD level m ” (30 in our case) label means that in this part of Quasi Birth Death process representation the matrixes A_{m-1}, B_m, C_m and all of the outer transitions (N_{ij} and K_{ij} matrixes namely) to another QBD processes are stored. Parameter m describes a QBD length parameter. Let call such QBD representation element as “End node of QBD process” or simply “end node”. All of the operations which are available for

start node are available for end node too and the ways how to perform them are the same as well, it means that, for example, to display the information about m -level boundary conditions node the user should press the left button on end node of QBD process.

- Inner states which are represented by a straight line, connecting the start node with end node of the given QBD process. In this user interface element the matrixes A, B, C and QBD length parameter are stored. As in previous cases the context menu and information about whole QBD process are available by clicking right or left mouse buttons on this straight line correspondingly. If user needs to change the coordinates of the whole QBD process, i.e. change the position of start and end nodes on the screen simultaneously, he should press the left button on this user interface element and, holding the left mouse button pressed, move it to the new position. After the appropriate position was reached, previously pressed button should be released.

4.2.3 Inner state matrixes addition to the QBD process

There are two ways to add the matrixes A, B, C and QBD length parameter to the given QBD process: during Quasi Birth Death process creation and in any time after given QBD process was successfully added to the task. These two methods are almost the same, except one thing – if user wants to add the information during creation process he does not need to call the special “Open file...” dialog – it will appear automatically, if user wants to add such information later, he needs to call for QBD context menu. Let’s discuss in details the second case.

To add information about inner states to the Quasi Birth Death process the user should call for QBD context menu, as it was mentioned above (the description how to do it is in the previous section).

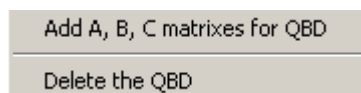


Figure 23: QBD process context menu.

After that “Add A, B, C matrixes for QBD” should be selected to open standard built-in Windows “Open file...” dialog. In this dialog the user should select an appropriate file with required data in it and press the open button. Note, that the default file extension for such

type of files is “.mqf”, but it is possible to change this extension to another one. If operation was not performed successfully the user will see an error window, otherwise the information about given QBD process in the Information window will change. There is no any way to delete once entered data for QBD process, the user can only reload it by choosing the “Add A, B, C matrixes for QBD” menu item and processing already described steps again or delete the whole Quasi Birth Death process at all by choosing the “Delete the QBD” option from the same menu.

The file which contains the information about current QBD process inner states is a plain text file and should have the following structure (see an example of such file in the appendixes section):

- “**QBD PROCESS LENGTH = 30**” string which describes the QBD length parameter ($m^{(i)}$ from section 2), which equals to 30 in our case. Note, that the minimal QBD length parameter equals to 3.
- “**START MATRIX: A**” string which says to program that there data for matrix A goes below this string.
- “**1.1, 2**” string which contains the first row for matrix A . Every element in matrix A row data string should be separated by commas from another elements, integer and decimal parts of number should be separated by decimal point. Every row should be finished by linefeed symbol. All elements in matrix A should be nonnegative.
- “**END MATRIX**” prefix means that the data description for current matrix was finished
- “**START MATRIX: B**” prefix which starts the data array for matrix B
- “**-2.1, 0**” – first row with data for matrix B . The rules of matrix description are the same as in description of matrix A except one thing: the diagonal elements in matrix B should be non-positive and each sum by row in expression $C+B+A$ should be equal to zero.
- “**END MATRIX**” prefix means that the data description for current matrix (matrix B in our case) was finished
- “**START MATRIX: C**” as in the previous case it’s a prefix which starts the data array for matrix C
- “**0, 0**” – first row with data for matrix C . The rules of matrix description are the same as for matrix A description.

- “END MATRIX” prefix means that the data description for last one matrix C was finished.

The dimensions of all these three matrixes should be the same both by columns and rows and each matrix should be square in order to set the conditions of the task correctly.

The maximal dimensions of each matrix and number of signs after decimal point are depends of implementation of MatrixDLL.dll Dynamic Link Library, by default they are 32767x32767 elements and 16 signs after decimal point accuracy correspondingly.

4.2.4 Boundary condition matrixes addition to the QBD process

Due of similarity of adding zero-level boundary conditions matrixes and m -level boundary conditions matrixes there is no need to separate this two processes description into two section. In this section the zero-level boundary conditions matrixes addition procedure will be described and all insignias for m -level boundary matrixes addition method will be noted in the square brackets.

Unlike in the inner state matrixes addition procedure there is only one way to add boundary conditions for start [end] node – by selecting the “Add Boundary matrixes (A_0 , B_0 , C_0 , ...) for QBD” [“Add Boundary matrixes (A_m , B_m , C_m , ...) for QBD”] context menu item.

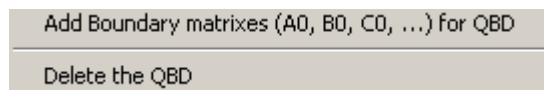


Figure 24: QBD start node context menu.

After that the standard built-in Windows “Open file...” dialog will be opened. In this dialog the user should select an appropriate file with required boundary data in it and default file extension “.mcf” (this file extension can be easily changed to another one, however), and press the open button. As in previous section if operation was not performed successfully the user will see an error window with error description, otherwise the information about given QBD process boundary matrixes (A_0, B_0, C_0 [A_{m-1}, B_m, C_m] with transition rate matrixes in our case) in the Information window will change as well as new connections will be drawn. Also there is no any way to delete once entered data for QBD process boundary, the user can only reload it by choosing the described above menu item

and processing already described steps again or delete the whole QBD process at all by choosing the “Delete the QBD” option from the same menu.

Important note: do not add any of the boundary condition matrixes until all of the QBD process will not be created; otherwise you can loose some connection matrixes for not yet existing QBD processes.

Important note 2: unlike in section 2 here matrixes B_0 , A_0 and C_0 are the same matrixes as in (2.1), i.e. matrix A_0 should have a top row of zeros. The same for B_0 - if it's components $\bar{b}_1^{(i)}, \bar{b}_2^{(i)}$ have only zero values they should be entered to the task as zero row and columns correspondingly.


The file which contains the information about current QBD process boundary condition matrixes is a plain text file as well and should have the following structure (see an example of such file in the appendixes section):

- “**START MATRIX: A0**” [“**START MATRIX: AM**”] string which says to program that there data for matrix $A_0[A_{m-1}]$ (note, that instead of AM-1 header there is AM matrix header)] goes below this string.
- “**0, 0, 0**” string which contains the first row for matrix $A_0[A_{m-1}]$. Every element in matrix $A_0[A_{m-1}]$ row data string should be separated by commas from another elements, integer and decimal parts of number should be separated by decimal point. Every row should be finished by linefeed symbol. All elements in matrix $A_0[A_{m-1}]$ should be nonnegative. The number of rows in matrix $A_0[A_{m-1}]$ should be equal to the number of rows in matrix $B_0[B]$ and the number of columns in matrix $A_0[A_{m-1}]$ should be equal to the number of columns in $B[B_m]$.
- “**END MATRIX**” prefix means that the data description for current matrix was finished.
- “**START MATRIX: B0**” [“**START MATRIX: BM**”] prefix which starts the data array for matrix $B_0[B_m]$

“**-1, 1, 0, 0**” – first row with data for matrix $B_0[B_m]$. The rules of matrix description as usual are the same as in description of matrix $A_0[A_{m-1}]$ except one thing: the diagonal elements in matrix $B_0[B_m]$ should be non-positive.

The size of matrix $B_0[B_m]$ should be $(n+1) \times (n+1)$ [$n \times n$] if matrix B has $n \times n$ dimensions.

- “**END MATRIX**” prefix means that the data description for current matrix (matrix $B_0[B_m]$ in our case) was finished
- “**START MATRIX: C0**” [“**START MATRIX: CM**”] this prefix starts an array of data for matrix $C_0[C_m]$ as in previous cases
- “**0, 1, 3, 1**” first row with data for matrix $C_0[C_m]$. The rules of matrix description are totally the same as in description of matrix $A_0[A_{m-1}]$. The number of rows in matrix $C_0[C_m]$ should be equal to the number of rows in matrix $B[B_m]$ and the number of columns in matrix $C_0[C_m]$ should be equal to the number of columns in matrix $B_0[B]$.
- “**END MATRIX**” prefix means that the data description for current matrix was finished.
- “**START CONNECTION**” – this prefix means that the following information describes the connection between current and some other QBD process (additional transition rate matrixes in other words).
- “**IS CONNECTION ENDS AT START = 0**” this string describes the connection destination node: 1 means that connection ends at the start node, 0 means that connection ends at the end node of another QBD process (M_{ij} [N_{ij}] or L_{ij} [K_{ij}] matrixes description correspondingly in terms of section 2). Note that the type of node in the current Quasi Birth Death process from which connection outgoes describes by the type of file, i.e. the current file is for start node or for end node, therefore it is possibly to define clearly the proper pair of matrixes - M_{ij} and L_{ij} or N_{ij} and K_{ij} .
- “**CONNECTED TO QBD NUMBER = 2**” - string which describes the destination QBD process number, 2 in our case. As it was mentioned above the QBD process number shows at the bottom side of each of nodes in every QBD, for example start

node for process number 1 will be shown as .

- “**START MATRIX: OUTGOING**” – this prefix starts a data array for current transitional rate matrix.
- “**1, 1, 0, 0**” - first row with data for connection matrix. The rules of matrix description are the same as in previous cases, i.e. every element in matrix row data string should be separated by commas from another elements, integer and decimal parts of number should be separated by decimal point. Every row should be finished by linefeed symbol. All elements in the given matrix should be nonnegative. The number of rows in connection matrix should be equal to the number of rows in matrix B of current QBD process and the number of columns in the given connection matrix should be equal to the number of columns in matrix B of destination QBD process (QBD process number 2 in our case).
- “**END MATRIX**” - as usually this prefix finishes the data description array for current matrix.
- “**END CONNECTION**” – this prefix closes the connection description.
- More connection description sections, if needed.

The information about number accuracy and maximum matrix dimensions provided in section 4.2.3 stays valid for this section too, i.e. the maximal dimensions of each matrix and number of signs after decimal point are depends of implementation of MatrixDLL.dll Dynamic Link Library, by default they are 32767x32767 elements and 16 signs after decimal point accuracy correspondingly.

More detailed connection description will be provided in the next section of this paper.

4.2.5 Connection description

In this section the connection description in details will provided.

As it was mentioned in this paper earlier the connection is only a virtual term for indication a non-zero transition rate matrix between two QBD processes like as M_{ij} , N_{ij} , L_{ij} or K_{ij} . The aim of creating such term is only in simplification of representation. All another ways of representation does not allow to create a clearly understandable tasks with simple representation, especially in case of tasks with cycles, for example task with two QBD processes where start node of QBD process 1 connected with end node of QBD process number 2 and end node of QBD process number 1 connected with start node of QBD process number 2 (here the word “connected” means that, for example, at least one

transition rate matrix from the start node of QBD process 1 to end node of QBD process number 2 or from end node of QBD process number 2 to start node of QBD process 1 is non-zero, in other case we consider that there is no connection, because both transition rate matrixes are equal to zero). An implemented representation of the connection is a straight line with white dots as it shown at figure 21.

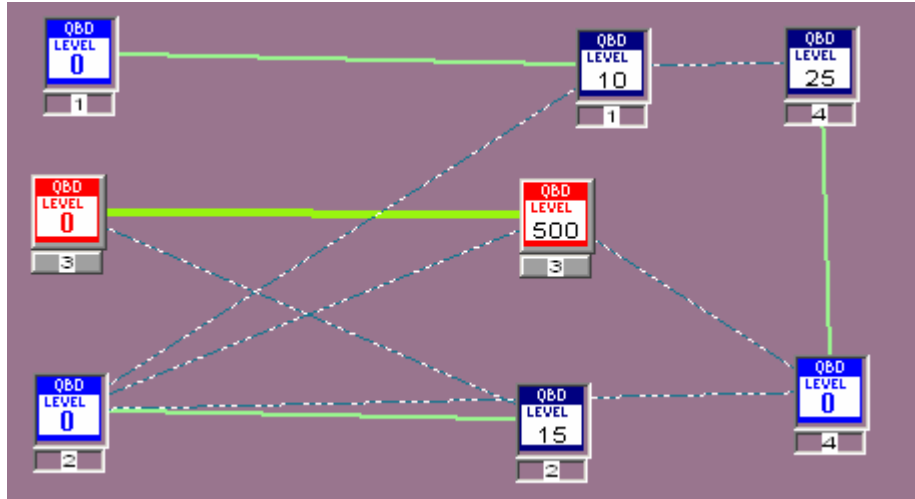


Figure 25: A typical task with 4 QBD processes and connections between them.

On figure 21 there are 7 connections which mean that at least seven transition rate matrixes in this task are non-zero. There is no need to specify the full set of connections for one QBD process to another QBD processes during addition of boundary conditions, it's possibly to specify only nonzero transition matrixes, all other connection matrixes will be set to zero matrixes with proper dimensions automatically, like in case of pseudo-connection between start node of QBD 1 and start node of QBD process 3.

As any of elements in work area each connection has own context menu with one operation in it. To perform this operation the user should click right-button of mouse once on any unused place of the work area. The following popup context menu will appear.



Figure 26: A connection context menu.

By selecting this menu item the user can zero two connection matrixes, for example if start node of QBD process number one connected to start node of QBD process number 2 there exist two transition matrixes M_{12} and M_{21} . As it was mentioned above one of them is non-zero and another one is zero or non-zero too. By choosing the “Delete the Connection Matrixes” menu item the user will zero both matrixes, therefore this pseudo-connection with zero transition rate matrixes will not be represented at work area more. Also there is

short information about given connection available. It can be displayed by pressing the left button on it – the appropriate connection will become selected and there short info will appear at the information window (see Information window description for more details).

4.2.6 QBD correctness checking procedure.

After all of the parameters for given QBD process were entered, i.e. inner state matrixes and boundary conditions the application tries to verify them for correctness by checking the dimensions of all matrixes, checking the values in whose matrixes and some more. If any errors will be found the user will see an error window with error description otherwise the start and end nodes icons for given QBD process will change their color to blue (or red if current QBD process is selected).



Figure 27: A representation of correctly entered data for QBD process.

Note, that the following warning window can appear as a result of verifying data process:

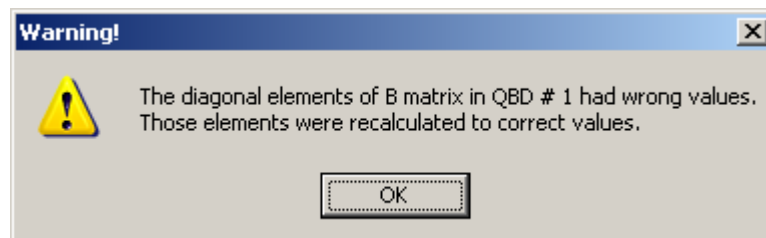


Figure 28: QBD process warning message after verifying.

This message means, that some sum by row in matrix G in one or more rows are not equal 0, where

$$G = C + B + A \quad (1.2)$$

The application fixed this incorrect row by increasing or decreasing the appropriate diagonal element of matrix B . This message is not an error message, but it is recommended to check again the matrixes A, B and C . The same situation can be for boundary conditions matrixes B_0 and B_m - there will be almost the same window with proper specification of type of corrected matrix. The recommendation in this case is the same as in previous case for matrix B – to check all of boundary matrixes.

Note that at the start of calculation stage and even during calculations there will be some more verifying procedures, like as checking for graph connectivity and so on. Therefore in case you have the properly entered correct data for each QBD process (meaning that every Quasi Birth Death Process is correct) it does not mean that all of the task conditions are correct and in some cases user should fix some more mistakes. Only after fixing all of existing mistakes the calculation process will start.

4.3 Information window

The information window is a window there short information about any element in work area will be represented. This interface element contains two parts – “Object selected” field where the type and some information about the object will be represented and “Object info” scrollable area where the properties of the given item will be displayed.

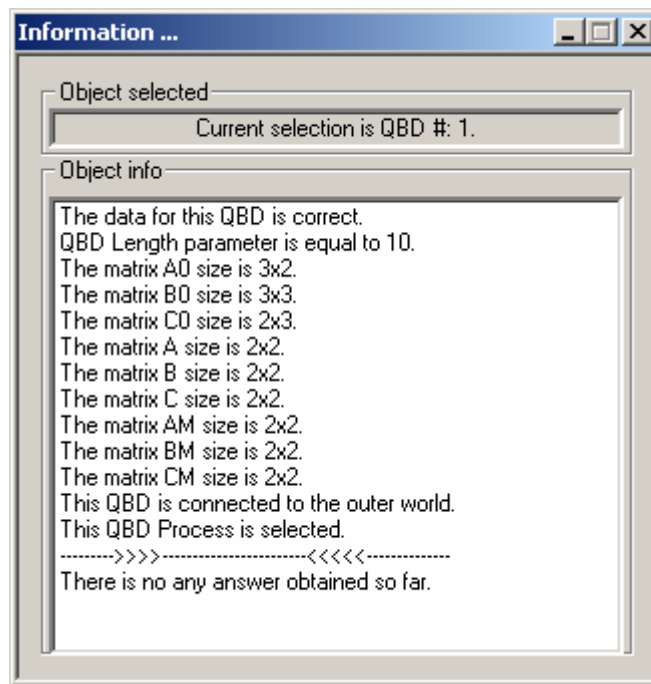


Figure 29: An information window with QBD process information in it.

The information window either can be empty or hold information about three types of main work area elements which could be selected, namely QBD process, QBD start or end node and connection between QBDs. In this section we will describe all this three types of information:

- In case of some node of QBD process selection there will be the following description of this selected item in information window:

- At the “Object selected” field will be the following string: “**Current selection is node number m** ”, where m is a number of selected node. Note that all nodes in the program have end to end numbering.
- At the “Object info” will be the following properties displayed:
 - The type of node, i.e. start or end and the number of QBD process to which the selected node belongs, for example “**This is a Start node of QBD process # 4.**”
 - Selected node X coordinate at the work area, for example “**Coordinate X of node is: 613**”
 - Selected node Y coordinate at the work area, for example “**Coordinate Y of node is: 232**”
 - The string with node selection status, for example “**This node is selected.**”
- When one of the existing connections will be selected the information window provides the following information.
 - At the “Object selected” field will be the following string: “**Current selection is QBD Connection #: m** ”, where m is a number of selected connection. Note that all connections which start from one QBD process numerate from 1, therefore in two different QBD processes can exist two different connections with the same number, for example 1.
 - At the “Object info” will be the following selected connection properties displayed:
 - The string which represents the result of connection data checking procedure, for example “**The data for this connection is correct.**”
 - The following strings which describes two QBD processes numbers between which connection was established. The first string of mentioned set will be as follows “**This QBD Connection connects two following QBDs:**”
 - “**The Start node of the QBD #: 4**” – this string means that selected connection starts at the start node of QBD process number four.
 - “**with end node of the QBD #: 3**” – this string means that selected connection destination is at the end node of QBD process number three.
 - “**The Outgoing matrix goes from Start node of QBD #: 4**” – this is the last string of described above set of string which describes the start and end points of selected connection and it means that so-called “outgoing matrix” (L_{43} in terms of section 2) was described at the boundary conditions file for start

node of QBD process number four, therefore so-called “incoming matrix” was described as “outgoing matrix” (N_{34}) in boundary conditions file for end node of QBD process number three. The terms “outgoing” and “incoming” matrixes were created for more clearly representation of them, because if boundary conditions file of one QBD process contains some transition rate matrix - this matrix will be outgoing transition matrix for given QBD process but at the same time it will be incoming transition rate matrix for destination QBD process to which the given QBD connects. Therefore in case of two connected Quasi Birth Death processes there will be two transition matrixes and each of them can be “incoming” or “outgoing” matrix at the same time. To prevent such situation and to define clearly the type of each of transition matrixes the terms “incoming” and “outgoing” were presented.

- The string which describes the dimensions of “outgoing” matrix (transition matrix, described in the boundary conditions file for start node of QBD process number four in this example), for example “**The Outgoing matrix size is $m \times n$** ”, where m is number of rows and n is number of columns of such matrix.
 - In case the “outgoing” matrix has only zero elements the following string will be displayed “**The Outgoing matrix has only ZERO elements**”.
 - The string which describes the dimensions of “incoming” matrix for QBD process number four (this transition matrix was described in the boundary conditions file for end node of QBD process number three in this example), for example “**The Incoming matrix size is $m \times n$** ”, where m is number of rows and n is number of columns of such matrix.
 - In case the “incoming” matrix has only zero elements the following string will be displayed “**The Incoming matrix has only ZERO elements**”.
 - The last one string keeps information about selection status of connection, for example “**This connection is selected**”.
- The last one type of information, provided by information window is the description of currently selected QBD process:
 - At the “Object selected” field will be the following string: “**Current selection is QBD #: m** ”, where m is a number of selected QBD process. Note that all QBD processes in the task have non-intersecting numeration.

- At the “Object info” will be the following information:
 - The string which represents the result of QBD process data checking procedure, for example **“The data for this QBD is NOT correct or full.”**
 - The string which shows the QBD length parameter of selected QBD process, for example **“QBD Length parameter is equal to m ”**, where m number represents the described value.
 - The following nine string describes the dimensions of inner transition matrixes A, B, C and the boundary matrixes $A_0, B_0, C_0, A_{m-1}, B_m, C_m$, for example **“The matrix A_0 size is $m \times n$ ”**, where m is number of rows and n is number of columns of matrix A_0 if it was already entered or **“The matrix A_0 is not specified yet”** otherwise.
 - The next string holds information about selection status of QBD process, for example **“This QBD Process is selected”**.
 - The last one set of strings contains the information about obtained or not yet obtained calculation result for given QBD process. This set starts from the separator string, looking like this: **“----->>>-----<<<<-----”**
 - In case the calculations was not performed yet the string below separator will be as follows: **“There is no any answer obtained so far”**, otherwise starting from this string the set of strings will be containing the result information for given QBD process in exactly the same format as result file has during the description of answer for one QBD, i.e. starting from **“ X_0 :”** prefix and finishing by **“ X_m :”** prefix, where m parameters equals to QBD length value (see Calculations menu item description for more details).

5. PROGRAM SPECIFICATION, FUNCTIONALITY AND IMPLEMENTED ALGORITHMS.

Due of impossibility to attach the source code to this paper (the source code is simply very large – it contains more that 500 kilobytes of plain text or more than 250 A4 paper sheets) it was decided to describe the implemented calculation support classes as briefly as it possibly, because without this information it will be impossible to improve any of the

calculation methods in future. However the source code of application is still available in electronic version.

In this chapter the description of program architecture, functions and implemented calculations algorithms will be given in details. The application was developed by using the Microsoft Visual C++ 6.0 Service Pack 5 development environment with using the Microsoft Foundation Classes (MFC), therefore the author assumes, that the reader is familiar with Visual C++ language and MFC classes. Most of the components of the program are implemented in DLL's, consequently the basic knowledge about Dynamic Link Libraries structure are also required.

5.1 Program architecture

Let's start from the application architecture description: there are four physical parts in the given program, three of them are contains in the Dynamic Link Library and implements the calculation algorithms, the fourth one is executable file which implements user interface and some internal functions. Such structure allows changing any needed algorithm without rewriting the whole program - it is necessary to rewrite the proper DLL function or functions only. Even the whole calculation method (for example, Naumov's method) can be changed to another one. The following figure describes the dependencies between implemented modules.

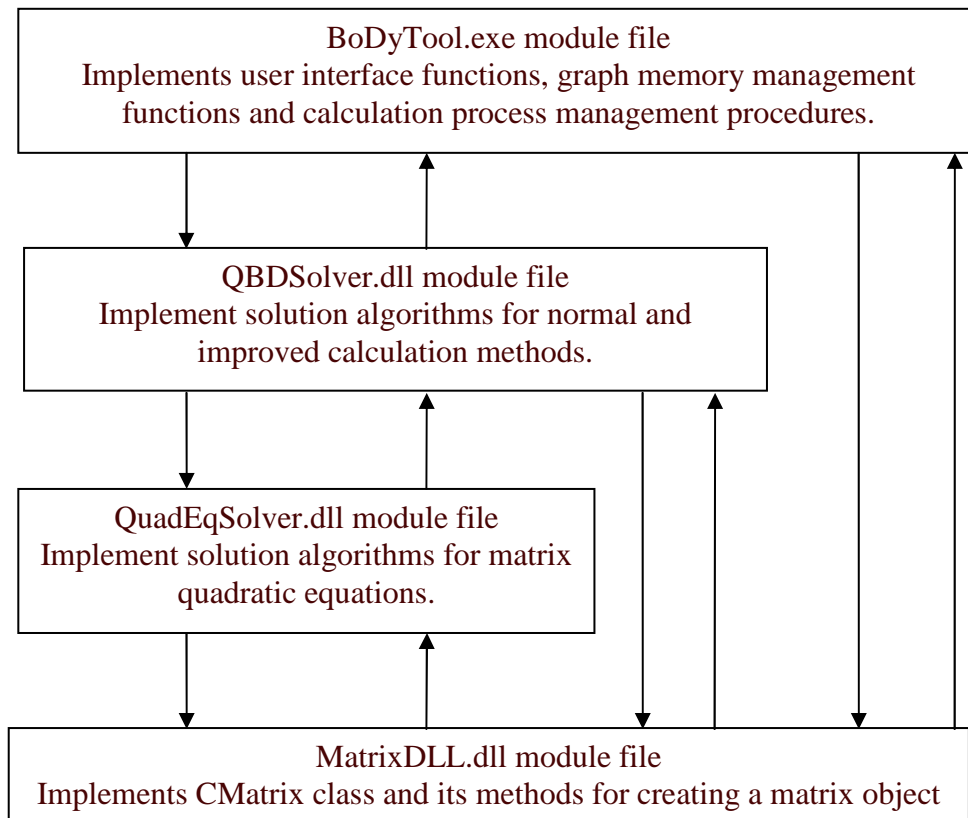


Figure 30: BoDyTool application architecture.

All of the implemented modules of application will be described in details in the following sections.

5.2 BoDyTool.exe module file description

This part of application is a user interface mostly, it provides the user with all of the objects, which can be viewed, for example any sorts of error windows, dialog boxes and so on. Also it holds all of the entered data about task and transfers it to the calculation modules if needed. No any calculations, except data checking procedures, perform inside this component! Therefore there is no any necessity to describe the functions and classes containing in this module in details, because this unit only calls the appropriate methods from other modules (DLL's in this case), but does not allow to call any of methods implemented in its own classes from outside. In other words this part of program depends from the functions implemented in other parts of application, but any of other parts is not depends from BoDyTool.exe component. One can easily develop his own interface with observance of calculation procedures restrictions, of course. There is a short list of implemented operations given below:

- User Interface functions (like as main menu support or work area data output support)
- Data storage and memory management functions (for example the functions for correct work area elements placement into the memory and others)
- Input/output to disk support (like as data saving to or data loading from file procedures)
- Calculation process management functions (for example multi-threaded calculation management procedures)
- Data checking procedures (for example functions for checking the QBD process for correctness)
- Some more internal functions.

5.3 *MatrixDLL.dll module file description*

MatrixDLL Dynamic Link Library is a primary component of application. Every other components use the implementation of CMatrix object class. This C++ class describes the standard mathematical matrix object and its available methods as addition of two matrixes and so on (the implementation of this class based on R. Allen code [13], but the most part of this code was rewritten). In this section of paper this program component will be described in all details.

Let start from definition of this class – it should be exactly like this:

class AFX_EXT_CLASS CMatrix : public CObject

Pay attention to prefix “AFX_EXT_CLASS” which means that this class can be exported from this DLL by other program components.

The implementation of matrix object, like as memory management, the number of signs in matrix element (accuracy), the maximum dimension of one matrix and so on can be easily redone by any third-part developer, but this class should support the following methods:

- **CMatrix();** - a default constructor for matrix object, which creates an empty matrix.
- **CMatrix(const CMatrix &other);** - so-called copy constructor, which creates a matrix and copies the information from **CMatrix &other** matrix object to newly created matrix.

- **CMatrix(int nCols, int nRows) ;** - this constructor should create a zero-filled matrix with **nCols** number of columns and **nRows** number of rows in it.
- **CMatrix(int n, bool set_diagonal = true);** - this constructor creates a square matrix with $n \times n$ size filled with zeros, if parameter **set_diagonal** will be equal to 1 the diagonal elements of such matrix should be equal to 1, therefore the identity matrix will be obtained.
- **virtual ~CMatrix();** - a destructor for CMatrix object which destroys earlier created matrix.
- **CMatrix& operator=(const CMatrix &other) ;** - this function should perform an $A = B$ operation, where A and B are matrixes
- **CMatrix operator+(const CMatrix &other) const ;** - this is an implementation of matrix addition operation, i.e. $C = A + B$
- **CMatrix operator-(const CMatrix &other) const ;** - - this is an implementation of matrix subtraction operation, i.e. $C = A - B$
- **CMatrix operator*(const CMatrix &other) const ;** - this function should implement a matrix by matrix multiplication operation, i.e. $C = A \cdot B$
- **void operator+=(const CMatrix &other) const ;** - this function should implement a special C++ operation += which means that $C = C + A$
- **void operator-=(const CMatrix &other) const ;** - this function should implement a special C++ operation -= which means that $C = C - A$
- **void operator*=(const CMatrix &other) const ;** - this is an implementation of special C++ operation *= which multiplies matrix by matrix, i.e. $C = C \cdot A$
- **void operator*=(double a) ;** - this is an implementation of special C++ operation *= which multiplies matrix by constant, i.e. $C = C \cdot a$ where a is a constant
- **bool operator==(const CMatrix &other) const ;** - it is an equality verifying operator which checks if $A = B$ or not. In first case this operator returns 1, in second – value 0.
- **void Serialize(CArchive& archive) ;** - this input/output function should store to disk or load from disk the matrix object by using the MFC object **archive**.
- **bool SetElement(int nCol, int nRow, double v) ;** - this function should set new value **v** for matrix element which locates at **nCol** position by columns and **nRow** position by rows.

- **double GetElement(int nCol, int nRow) ;** - this function should return a value of matrix element which locates at **nCol** position by columns and **nRow** position by rows.
- **inline int GetNumColumns() const;** - this procedure returns number of columns in the given matrix.
- **inline int GetNumRows() const;** - this procedure returns number of rows in the given matrix.
- **double SumColumn(int col) const ;** - this function should return a sum of elements in column number **col** of given matrix
- **double SumRow(int row) const ;** - this function should return a sum of elements in row number **row** of given matrix
- **double GetDeterminant(void);** - the result of proceeding this function will be a determinant value of the given matrix. The standard recursive algorithm was implemented in this procedure.
- **CMatrix GetTransposed() const ;** function and void **Transpose() ;** simply transposes the given matrix.
- **CMatrix GetInverted() const ;** and **void Invert() ;** functions looks for inverted matrix for current matrix and contains two part – the procedure which calculates a normal inverted matrix and a procedure which calculates a general inverse group in case of determinant of source matrix equals to zero (a singular case). The first type of algorithm is well known and there is no need to discuss it in this paper, the second one is not so commonly used. To implement the general inverse group calculation function the article [6] was used where the method for obtaining $\Phi^{\#}$ was given.
- **CMatrix GetConcatinatedColumns(const CMatrix& other) const;** and **void ConcatinateColumns(const CMatrix &other) ;** procedures concatenate two matrixes into one by column splicing, i.e. in case we have two matrixes A and B with dimensions mxl and mxn the result matrix C dimensions will be $mx(l+n)$. It can be drawn schematically as follows $C = A \therefore B$ where operation \therefore means columns concatenation.
- **CMatrix GetConcatinatedRows(const CMatrix& other) const ;** and **void ConcatinateRows(const CMatrix &other) ;** functions makes the same operations as functions described above but for matrix rows.

- **CMatrix SolveLinearSystem(const CMatrix &b);** the most important function in this class which solves the system of linear equations by Gauss method. This method is quite slow, but it is suitable for solving the described QBD system tasks due of the small amount of equations, however any other method, like LU, can be implemented in this function.

- **CMatrix ExtractSubMatrix(int cs, int rs, int csize, int rsize) const ;** it is a mostly used function in the program, it extract a submatrix from the given matrix. The newly extracted submatrix will be as follows:

$$A = \begin{pmatrix} a_{rs,cs} & \cdots & a_{rs,cs+csize} \\ \vdots & \ddots & \vdots \\ a_{rs+rsize,cs} & \cdots & a_{rs+rsize,cs+csize} \end{pmatrix} \text{ where } \mathbf{cs, rs, csize, rsize} \text{ are the incoming}$$

function parameters, which describes the start element position and number of rows and columns to extract from the source matrix.

- **double GetInfNorm(void);** this function returns the calculated infinity norm of the given matrix. The formula for calculating such norm is as follows:

$$l = \max_j \left| \sum_{i=1}^n a_{ij} \right| \quad (5.1)$$

- **CMatrix GetInPow(int MatrixPow);** and **void InPow(int MatrixPow);** functions calculate the power of the current matrix, where **MatrixPow** parameter is needed value of power. Due of much number of calls for this procedures, the improved algorithm of power calculations, offered by Valery Naumov [10] was used:

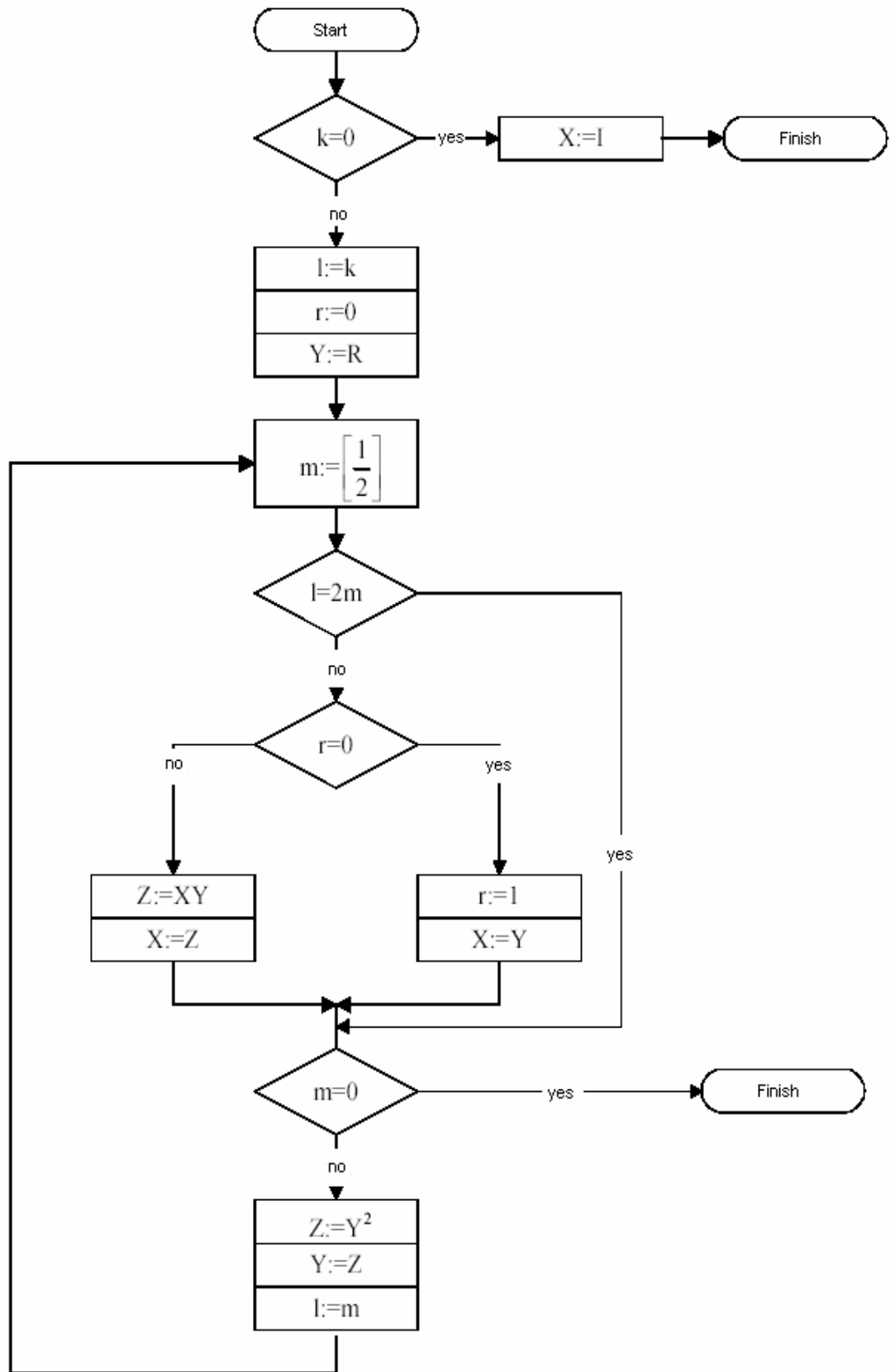


Figure 31: Improved algorithm of power calculations representation.

Here k parameter (in our case it is equal to **MatrixPow** value) describes the value of needed power, R matrix is a source matrix and X matrix is a result matrix, i.e. $X = R^k$. The described algorithm allows to decrease the time of power calculations significantly, therefore the time, required for task solution decrease significantly too.

- **int GetLastMatrixErrorCode(void)**; this function returns the last error code of mistake, occurred during performed operation, see list of all error codes below.
- **bool IsEmpty(void)**; the last one function in this class returns 1 in case the given matrix is empty (there was no any kind of operations with this matrix yet), otherwise this function returns 0.

There are some error codes implemented in this DLL, they are:

- **OPERATION_OK** – operation was performed successfully, no errors were found.
- **WRONG_MATRIX_FILE_FORMAT** – this error shows, that during loading process the serialization function found that input file has wrong format, therefore there is no any possibility to continue matrix loading procedure.
- **MATRIX_IS_NOT_SQUARE** error code appears during attempting to apply some functions to non-square matrix, for example during determinant calculation procedure.

The interface module can use this error codes to inform user about error occurred.

As it was mentioned above all of other application components uses this class, therefore any changes which will be done in this Dynamic Link Library will directly affect on another parts of program. Due of this fact the developer, who modifies this part of application, should be very careful.

5.4 QuadEqSolver.dll module file description

QuadEqSolver application part is a special library, which solves different types of matrix quadratic equations by using iteration methods. This Dynamic Link Library uses the CMatrix object implemented in MatrixDLL.dll component and used by QBDSolver.dll

module. This DLL is also so-called extensional DLL and implements a class which performs the described functions. This class can be exported by any external program components.

As in the previous section, let start from definition of the class implemented in this library– it should be exactly like this:

class AFX_EXT_CLASS CQuadraticEquationSolver

Pay attention to prefix “AFX_EXT_CLASS” which means that the class **CQuadraticEquationSolver** can be exported from this DLL by other program components.

The destinations of this class are to manage memory for input parameters and obtained results and for supporting quadratic equation solution procedures. One can redone any of implemented algorithms, but this class should support the following C++ methods:

- **CQuadraticEquationSolver();** - a default constructor which constructs the implemented into library class and initializes all variables to proper default values.
- **virtual ~CQuadraticEquationSolver();** - a default destructor, which destroys an existing object and releases memory.
- **void SetMatrixA(CMatrix NewMatrixA);** - this implemented method sets the elements in matrix *A* to be equal to input parameter matrix **NewMatrixA** for a further task solution.
- **void SetMatrixB(CMatrix NewMatrixB);** - the same as in the previous case, but for matrix *B*
- **void SetMatrixC(CMatrix NewMatrixC);** - the same as in the previous case, but for matrix *C*
- **void SetTaskType(int NewTaskType);** - this function sets the type of the task to be solved. The description of available task types goes below.
- **CMatrix SolveTask(void);** - the most important function in this class. This procedure solves a task, with type and input parameters specified earlier.

Here the description of all task types and implemented algorithms will be given:

There are three types of tasks supported by given Dynamic Link Library, they are:

$$A + R \cdot B + R^2 \cdot C = 0 \quad (5.2)$$

This is a standard matrix quadratic equation, where unknown matrix is R , lets define this task as “Left type task”.

$$A + B \cdot R + C \cdot R^2 = 0 \quad (5.3)$$

This equation is close to (5.2) the difference is only in location of unknown matrix R . Lets define this task as “Right type task”.

$$V = -(B + AVC)^{-1} \quad (5.4)$$

$$W = -(B + CWA)^{-1} \quad (5.5)$$

This is a special type of tasks, needed for improved calculations method implementation (Naumov’s algorithm for single QBD process in this case [5]). The equations (5.4) and (5.5) solves simultaneously in one iteration algorithm, because there is no need to separate these two quadratic equations into two independent tasks.

It is easy to see that the left type and right type tasks can be solved by using the same method, because (5.2) task can be obtained from (5.3) task by transposing all of summands.

The algorithm for solving (5.2) is a Logarithmic Reduction algorithm [11] which can be represented as follows:

$$S = B$$

$$V = A$$

$$T = C$$

$$W = B$$

do

$$X = -S^{-1}V$$

$$Y = -S^{-1}T$$

$$Z = VY$$

$$W = W + Z$$

$$S = S + Z + TX$$

$$V = VX$$

$$T = TY$$

while ($\|Z\|_{\infty} \geq \varepsilon$)

$$R = -AW^{-1}$$

where matrixes A, B, C is an input parameters for the described above task and matrix R is a result matrix.

For solving a $V = -(B + AVC)^{-1}$ and $W = -(B + CWA)^{-1}$ tasks a little bit modified method was used, however it is very similar to the previous one.

$$N = B$$

$$L = A$$

$$M = C$$

do

$$X = -N^{-1} \cdot L$$

$$Y = N^{-1} \cdot M$$

$$Z = L \cdot Y$$

$$T = M \cdot X$$

if ($\|Z\|_{\infty} \leq \varepsilon$) *and* ($\|T\|_{\infty} \leq \varepsilon$) *FINISH*

$$V = V + Z$$

$$W = W + T$$

$$N = N + Z + T$$

$$L = Z \cdot X$$

$$Z = M \cdot Y$$

$$M = Z$$

while not FINISH

$$V = V + B$$

$$W = W + B$$

$$V = -V^{-1}$$

$$W = -W^{-1}$$

where matrixes A, B, C is an input parameters and matrix V and W are result matrixes. As it was mentioned above the result of the solution equations (5.4) and (5.5) can be obtained in one iteration method simultaneously.

Also in each method the maximum iteration constant was implemented. This constant allows to avoid cases, when a too high accuracy value was set and programs tries to reach this high accuracy solution by increasing the used cycles number, therefore answer obtaining procedure takes a lot of time. By default this

maximum iteration count constant equals to 1000 – it is far enough for solving the most part of tasks.

After the solution was obtained it will be returned by given function in case of right or left type tasks; however obtained V and W values should be returned by calling the **GetResultVW** function.

- **void GetResultVW(CMatrix &MatrixV,CMatrix &MatrixW);** - this function returns the answers of (5.4) and (5.5) tasks after they were obtained.
- **int GetLastError(void);** - this method returns the code of last error occurred during solving a task. See a list of available error codes below.
- **void SetAccuracyConstant(double NewAccuracyConst);** - this procedure sets a new accuracy constant for all calculations methods (see ε value in **SolveTask** function).

There are a list of available error codes were implemented in this library, they are:

- **OPERATION_OK** – operation was performed successfully, no errors were found.
- **MATRIX_B_HAS_DET_0**– this error can appear during calculation process in case if on some step the determinant of matrixes S or N correspondingly to two described algorithms will be equal to zero, therefore it is impossible to calculate the inverted matrix S or N .
- **MAX_ITERATION_NUMBER_REACHED** error code appears in case of reaching the maximum number of iterations in described algorithms without obtaining any solution.

Though these errors theoretically can appear there was no any case of such kind of errors in practice during development process, however it is better to call the **GetLastError** function after calculations were finished and check the error code always.

5.5 QBDSolver.dll module file description

The last one application component which should be described is QBDSolver.dll file. This file is also a special library which performs the task solution process by using the input data and returns an obtained result. This Dynamic Link Library uses the CMatrix

object implemented in MatrixDLL.dll component and CQuadraticEquationSolver component for solution some matrix quadratic equations which allocated in QuadEqSolver.dll module. This DLL is also so-called extensional DLL and implements a class which performs the described functions, therefore this class can be exported by program interface.

As always, let start from definition of the class implemented in this library– it should be exactly like this:

class AFX_EXT_CLASS CQBDDTaskSolver – this means that the class **CQBDDTaskSolver** can be exported by external objects (BoDyTool.exe application in our case). There are no any classes for external usage in this library, but there exist two special structures for transferring the data about each QBD process and each QBD connection, which are also should be accessed outside.

The destinations of this class are to manage memory for inputted from user interface module data, to control the process of task solution and to return a result after all calculations were finished. As in previous cases one can redone any of implemented algorithms and functions, but this class should support the following C++ methods:

- **void SetMatrixQForNormalMethod(CMatrix NewMatrixQ);** - this function sets the matrix Q for solving a task by using a normal method. Note, that matrix Q generates by user interface, because this operation does not require any calculations and there is no any improvements there.
- **void SolveTaskWithNormalMethod(void);** - this procedure solves a task with using normal method (Gauss method for solving a system of linear equations) for transferred earlier matrix Q . The equation, which will be solved by this method is as follows:

$$pQ = 0 \quad (5.6)$$

where matrix Q is a generator matrix for the whole system and p is unknown vector which should satisfy to the normalizing condition

$$pe = 1 \quad (5.7)$$

After obtaining a solution it can be transferred back by using **GetResult** method in case of error absence. Note, that this method of solution is not optimized and takes a long time to receive any result, so it is not recommended to use it for solving big

tasks with high QBD length parameter – for such type of tasks it is better to use an improved solution method.

- **void SetQBDChainLength(unsigned int NewQBDChainLength);** - this function starts a new calculation process and should release all memory, used for previous solution, as well, as to allocate a memory for new QBD processes data. The number of such QBD processes transfers in **NewQBDChainLength** parameter.
- **void SetNewQBDParameters(CQBDDataTransferStruct NewQBDDData);** - this functions transfers the data for some QBD process by using special **CQBDDataTransferStruct** structure to the calculation component from user interface. The transferred data should be stored into memory by proper procedures in this method. Also it is possible to perform some precalculations in this function, for example, the default version of this procedure performs step 1 of algorithm, described in section 2, by using the **CQuadraticEquationSolver** class stored in **QuadEqSolver.dll**
- **void SetConnectionChainLength(unsigned int NewConnectionChainLength);** - this function is similar to **SetQBDChainLength** method, i.e. it also should free any previously used memory for QBD connections data and should allocate a new portion of memory for storing the properties of QBD connections which will be transferred later. The number of such QBD connections is in input **NewConnectionChainLength** parameter.
- **void SetNewConnectionParameters(CConnectionDataTransferStruct NewConnectionData);** - this function, as in case of **SetNewQBDParameters** function stores a transferred from user interface QBD connection data by using a special **CConnectionDataTransferStruct** structure, which fills by external application component, into memory. Also it is possible to perform some precalculations in this method, if needed.
- **void SolveTaskWithImprovedMethod(void);** This is a heart of described component, i.e. this function solves the given task with previously set parameters by using the improved method. The procedures, which performs here are steps 2) – 8) of the algorithm, described in section 2.
- **int GetLastError(void);** - this function returns an error code of last error occurred during solution process or during memory allocation procedures. See a list of available error codes below.

- **CMatrix GetResult(void)**; function returns a result of calculations (steady state probability vector) in form of vector. The result vectors for both implemented methods of solution, i.e. normal and improved methods, should be the same.
- **void SetAccuracyConst(double NewAccuracyConst)**; method sets new accuracy constant for solution process. This constant will be used by all stages of calculations, therefore this value should be transferred to another computation modules by calling the appropriate functions.

There are a list of available error codes were implemented in this library, they are:

- **OPERATION_OK** – operation was performed successfully, no errors were found.
- **NO_MORE_MEMORY**– this error appears in case of application was unable to receive a required amount of memory. The calculations should be stopped after this error.
- **WRONG_TASKMATRIX_DIM** – this is an error code for debugging purposes mostly, it shows that during solution process some matrix had wrong dimensions; therefore the solution can't be obtained.
- **ABORTED_BY_USER** – in some stages of calculations the user activity can be required (for example a notification about wrong QBD process data and so on), this error code is a notification to main user interface that user aborted calculations.

Note, that it is necessary to call the **GetLastError** function after each operation performed, because of memory allocation procedures existence.

As it can be easily seen this module manages and performs whole solution procedures or calls for them another components, therefore during a new method development process it is possible to create some addition libraries with some required functions implemented in them. For example the QuadEqSolver.dll module can be easily separated to two independent components with special equation type solution procedure in each.

5.6 *Dynamic Link Libraries description*

Only main things about DLLs will be presented here. This section does not give an answer to the question like “how to create a DLL?”, for more information consult the documentation of the language that you are using and MSDN website [12].

An MFC extension DLL is a DLL that typically implements reusable classes derived from existing Microsoft Foundation Class Library classes. Extension DLLs are built using the dynamic-link library version of MFC (also known as the shared version of MFC). Only MFC executables (either applications or regular DLLs) that are built with the shared version of MFC can use an extension DLL. With an extension DLL, you can derive new custom classes from MFC and then offer this “extended” version of MFC to applications that call your DLL.

Extension DLLs can also be used for passing MFC-derived objects between the application and the DLL. The member functions associated with the passed object exist in the module where the object was created. Since these functions are properly exported when using the shared DLL version of MFC, you can freely pass MFC or MFC-derived object pointers between an application and the extension DLLs it loads.

The main reason to use the extension DLLs in the BoDyTool was to implement a reusable class, which describes matrices and matrix operations. The definition of the CMatrix class looks as **class AFX_EXT_CLASS CMatrix : public COject** Therefore to create a new class for description of matrices you should add the **AFX_EXT_CLASS** prefix to the class definition.

Note that you should not change the functions names, the functions arguments, the functions return values and the class members types and names, because these statements are used throughout the program, and any changes can lead to undesirable consequences.

6. PRACTICAL ASSIGMENT.

In this section we will describe the results obtained during testing procedures and compare the accuracy and calculation speed with XTelpack tool and two implemented methods between each other.

As it was mentioned above no one of the existing applications can perform full set of the available task type's solutions, but anyway all of such applications are able to solve any task for single QBD process. Although such tasks do not allow to demonstrate a developed application availabilities in full details, they allow to compare the accuracy and solution time for the same job, at least.

Due of XTelpack is an application which works on a very powerful server the time, needed for performing calculations is very low and approximately equals to zero independently from task, on the other hand the BoDyTool application is a stand-alone program which works on a single user computer, therefore it was decided to compare the time required for solution obtaining for two implemented algorithms, i.e. normal solution method and improved solution method (Valery Naumov's "Modified Matrix Geometric Algorithm" [5] in this case).

All of the tests were performed on a typical Intel Pentium III 450 MHz computer with 256 MB of Random Access Memory and Windows XP SP1 operation system installed.

A single QBD process with variable QBD length parameter and 2x2 internal dimension (the dimension of matrix B is 2x2) was taken as a test task for measuring accuracy and speed of calculations.

The time, needed for solution obtaining is represented on the next figure:

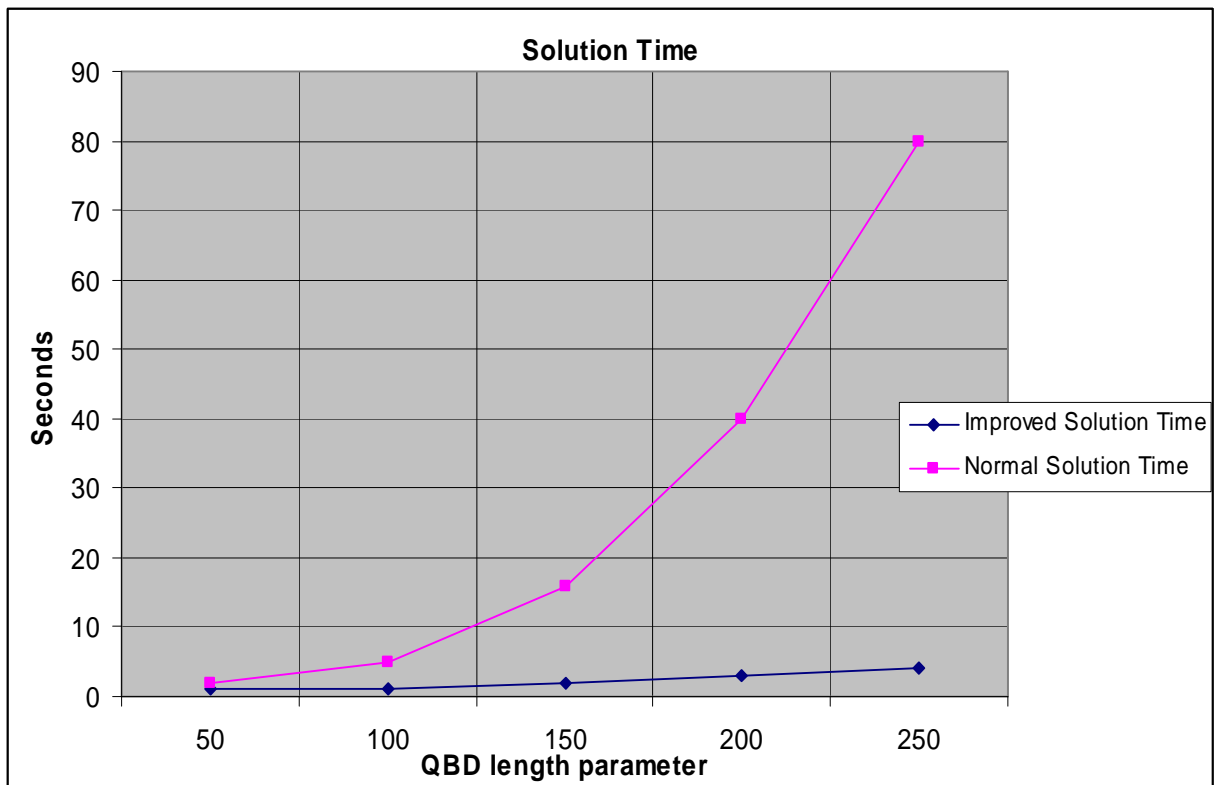


Figure 32: Time, required for solution obtaining in case of single QBD process task.

Such results are very predictable, because normal solution algorithm requires the matrix Q for the whole system generation (and in case of QBD length parameter equal to 250 the dimensions of such matrix will be 502x502 elements), as well as solution of system of linear equations where the amount of equations is equal to matrix Q dimensions (i.e. 502 in our case). Therefore such method is suitable only for tasks with small QBD length parameter; otherwise the solution process will require a huge amount of memory and a lot of CPU cycles. The improved method does not require matrix Q for solution, therefore the amount of required memory is low, but number of equations (and a solution time, therefore) depends greatly from internal dimension parameter and number of QBD processes in the task. Anyway improved algorithm presented an excellent result by calculation speed – in case of QBD length parameter equals to 250 Naumov's method obtained a solution in 16 times faster than normal algorithm.

During the accuracy of solution tests the default accuracy parameter from the XTelpack tool was used as a required precision value for both implemented methods and was equal to $1e-8$. Due of QBD length parameter affects greatly on solution process and, therefore allows to check the result precision dependency from the number of performed cycles, it was decided to perform the same set of tests, i.e. the QBD length parameter was varied from 50 to 250 for single QBD task. The results were as follows: the difference between answers in normal method, improved method and solution, obtained by XTelpack tool (Logarithmic Reduction method was used in this test) was not greater than $1e-7$. Such results allow to say that the accuracy of considered methods is almost the same, therefore two implemented algorithms are suitable for real tasks solution.

7. CONCLUSIONS

In this paper a description of developed application for solving a Quasi Birth Death processes system was given. This program does not have any known and available for public use analogues in the world, therefore it is possible so say that this application is unique. How it was mentioned above this application allows to solve a special set of tasks needed for communications system analysis, for example for analyzing a packet delays across GPRS/GSM networks. Due of the implemented Valery Naumov's "Modified Matrix Geometric Algorithm", the calculation speed was totally increased against a normal calculation method. The accuracy of the described improved method was nearly the same as in normal calculation process.

A fully tested application which performs all of the described operations was developed by using Visual C++ 6.0 language and this program demonstrated very good results comparing to another applications, which allows to solve some of the cognate tasks.

The fully detailed description of the program and some of input and output data files examples was provided.

The application architecture description in details with each component specification was given. All of the implemented algorithms with some notes about their modifications or improvements were described also. Due of this one can change any part of calculation process in order to increase speed or accuracy of computations or even to implement his own calculation method.

Due of application unique it can be applied in different domains of communication system tasks and due of component-based architecture mentioned domain of tasks for the given program can be easily expanded by implementation of new algorithms.

8. FUTURE WORK

The further work is to use this application during computations of a set of special communication systems tasks at the Laboratory of Telecommunications. Also it is planning to extend some previously implemented algorithms and methods of solution in order to get a really multifunctional tool which will be able to solve a wide range of tasks. One more thing which might be needed in future is a some time critical source code implementation by using another more fast code execution language, for example Microsoft Assembler. The next possible step of application improvement is to port the program to another platform, like as Unix/Linux systems.

REFERENCES

- [1] D. Dyakov. *Spectral Analysis of Buffers in Communication Systems*. Master Thesis, Lappeenranta University of Technology, 2003.
- [2] V.L. Wallace. *The Solution of Quasi Birth Death and Death Processes Arising from Multiple Access Computer Systems*. PhD thesis, University of Michigan, 1969.
- [3] D. Bini, B.Meini. *Improved Cyclic Reduction for Solving Queuing Problems*. Numerical Algorithms, Vol. 15, 57-74, 1997.
- [4] G. Clardo, E. Smirni. *ETAQA: an Efficient Technique for the Analysis of QBD Processes by Aggregation*. Performance Evaluation, Vol.36, 71-93, 1999.
- [5] V. Naumov. *Modified Matrix-Geometric Solution for Finite QBD Processes*, in *Advances in Algorithmic Methods for Stochastic Models*. Notable Publications Inc., 2000.
- [6] K. Manjunatha Prasad, R.B. Barat. *General Inverses Over Integral Domains II. Group Inverses and Drazin Inverses*, Linear Algebra Appl., Vol. 146, 31-47.
- [7] A. Riska. *Aggregate Matrix-analytic Techniques and their Applications*. PhD Theses, College of William and Mary, Williamsburg, VA, 2003.
- [8] *MAMSolver application official website*, <http://www.cs.wm.edu/MAMSolver/>
- [9] *Telpack Version 2 application official website*, <http://www.sice.umkc.edu/telpack/>
- [10] V. Naumov. *Numerical Methods for Markovian Systems Analysis*, The UDN Press, Moscow, 1985 (in Russian).
- [11] U.R. Krieger, V. Naumov. *Analysis of a Versatile Queuing Model with State-Dependent Service Times*, in *Measurement, Modeling and Evaluation of Computer and Communication Systems*. VDE – Verlag Berlin, 1999.

[12] *Microsoft Developer Network official website*, <http://www.msdn.microsoft.com/>

[13] R.Allen *Visual C++ CMatrix class implementation*,
<http://www.codeproject.com/cpp/matrixclass.asp?target=CMatrix>

APPENDIX 1. An example of input file with data for one QBD process.

%This is an example of QBD matrixes input file
QBD PROCESS LENGTH = 50 %This parameter describes the QBD queue length

START MATRIX: A0 %A0 matrix header
0, 0
1, 1
2, 1
END MATRIX %End of matrix A0

START MATRIX: B0 %B0 matrix header
-3, 1, 2
2, -4, 1
1, 4, -8
END MATRIX %End of matrix B0

START MATRIX: C0 %C0 matrix header
0, 2, 0
0, 1, 3
END MATRIX %End of matrix C0

START MATRIX: A %A matrix header
0, 1
2, 1
END MATRIX %End of matrix A

START MATRIX: B %B matrix header
-2, 3
4, -1
END MATRIX %End of matrix B

START MATRIX: C %C matrix header
2, 0
1, 3
END MATRIX %End of matrix C

START MATRIX: AM %AM matrix header
0, 1
2, 1
END MATRIX %End of matrix AM

START MATRIX: BM %BM matrix header
-4, 2
4, -8
END MATRIX %End of matrix BM

START MATRIX: CM %CM matrix header
2, 0
2, 3
END MATRIX %End of matrix CM

APPENDIX 2. An example of input file with boundary data for one QBD process.

%This is an example file, which contains the example how to create a boundary conditions matrix file

%This file is for AM BM CM Boundary conditions

START MATRIX: AM %AM matrix header

1, 1, 1, 1

1, 1, 1, 1

1, 1, 1, 1

1, 1, 1, 1

END MATRIX %End of matrix AM

START MATRIX: BM %BM matrix header

-20, 1, 1, 1

1, -20, 1, 1

1, 1, -20, 1

1, 1, 1, -20

END MATRIX %End of matrix BM

START MATRIX: CM %CM matrix header

1, 1, 1, 1

1, 1, 1, 1

1, 1, 1, 1

1, 1, 1, 1

END MATRIX %End of matrix CM

START CONNECTION %Starting to describe the outgoing transaction matrixes

IS CONNECTION ENDS AT START = 1 %This transaction ends at B0 of another QBD Process

CONNECTED TO QBD NUMBER = 2 %This transaction ends at QBD #2

START MATRIX: OUTGOING %Transaction Matrix starts here

1, 1, 1

1, 1, 1

1, 1, 1

1, 1, 1

END MATRIX %Transaction Matrix ends here

END CONNECTION %Transaction (Connection) description ends here

START CONNECTION %Starting to describe the outgoing transaction matrixes

IS CONNECTION ENDS AT START = 0 %This transaction ends at BM of another QBD Process

CONNECTED TO QBD NUMBER = 4 %This transaction ends at QBD #4

START MATRIX: OUTGOING %Transaction Matrix starts here

1

1

1

1

END MATRIX %Transaction Matrix ends here

END CONNECTION %Transaction (Connection) description ends here

%End of all file

APPENDIX 3. An example of output file with solution result.

%This is a file, which contains the results of task calculations.
%The QBD number described the QBD the answer belongs to
%The Xi describes the probabilities to be in the given condition.

The probability, that system is in idle condition:
0.01082

----->>>-----<<<<-----

The following results is for QBD # 1:

The mean length of queue:

0.260205

X0 :

0.006392999999999999

0.009675999999999999

X1 :

0.008033

0.009160999999999999

X2 :

0.009556

0.011764

X3 :

0.016424

0.020023

----->>>-----<<<<-----

The following results is for QBD # 2:

The mean length of queue:

1.41047

X0 :

0.031641

0.023795

0.024315

X1 :

0.021575

0.035023

0.028508

X2 :

0.022935

0.037289

0.030237

X3 :

0.024556

0.039963

0.031297

X4 :

0.029564

0.037339

0.034269

----->>>-----<<<<-----

The following results is for QBD # 3:

The mean length of queue:

1.08032

X0 :

0.009058

0.007623

0.006544

0.005905

X1 :

0.008956

0.008956

0.008956

0.008956

X2 :

0.01063

0.01063

0.01063

0.01063

X3 :

0.012303

0.012303

0.012303

0.012303

X4 :

0.013977

0.013977

0.013977

0.013977

X5 :

0.023059

0.01979

0.009875

0.009875

----->>>-----<<<<-----

The following results is for QBD # 4:

The mean length of queue:

0.513022

X0 :

0.046637

X1 :

0.036062

X2 :

0.027837

X3 :

0.02144

X4 :

0.016465

X5 :

0.012595

X6 :

0.009585