Journal of Theoretical and Applied Computer Science
ISSN 2299-2634

Vol. 6, No. 1, 2012, pp. 47-66
http://www.jtacs.org

Controllability and observability gramians parallel computation using GPU

Damian Raczyński, Włodzimierz Stanisławski

Opole University of Technology, Faculty of Electrical Engineering, Automatic Control and Computer Science, Poland

d.raczynski@doktorant.po.edu.pl

Abstract: *Algorithms and parallel programs for the controllability and observability gramians computation of the Linear Time Invariant (LTI) systems using Lyapunov equation with an application of the NVIDIA general purpose Graphics Processing Unit (GPU), are presented in the paper. Parallel computing of the gramians on the basis of Lyapunov equation is justified for the large scale systems ($n>10^4$) due to the computational cost $O(n^3)$. The parallel performance of controllability gramians computation using NVIDIA graphics hardware GTX-465 have been compared with the performance obtained for MATLAB environment employing analogous algorithms. They have also been compared with the performance obtained for lyap function provided by MATLAB environment. The values of maximum computing acceleration were up to 20. The computations have been made on the basis of linearized models of the one-phase zone of a once-through boiler obtained with the finite elements method. The orders of the models were being adapted within the range between 30 and 4200.*

Key words: *Controllability and observability gramians, Lyapunov equation, GPU, parallel computing*

1. Controllability and observability gramians concept

For LTI systems, described by state equations:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{1}$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

a controllability gramian **P** and an observability gramian **Q** are square symmetric matrices determined in accordance with [20]:

$$\mathbf{P} = \int_0^\infty e^{\mathbf{A}t}\mathbf{B}\mathbf{B}^* e^{\mathbf{A}^* t} dt \qquad \mathbf{Q} = \int_0^\infty e^{\mathbf{A}t}\mathbf{C}^*\mathbf{C} e^{\mathbf{A}^* t} dt \tag{2}$$

In practice, the matrices **P** and **Q** are determined on the basis of the Lyapunov equations [9]:

$$\mathbf{A}\mathbf{P} + \mathbf{P}\mathbf{A}^* + \mathbf{B}\mathbf{B}^* = 0 \qquad \mathbf{A}^*\mathbf{Q} + \mathbf{Q}\mathbf{A} + \mathbf{C}^*\mathbf{C} = 0 \tag{3}$$

One of the main methods for linear models reduction by means of Singular Value Decomposition (SVD) is based on the gramians matrices **P** and **Q** [1].

Since both of the Lyapunov equations (3) have similar forms, therefore all algorithms and programs in the further part of the paper have been based on the procedure of determining the controllability gramian **P.**

2. Methods of solving the Lyapunov equation

The basic methods for solving the Lyapunov equation $\mathbf{AP}+\mathbf{PA}^{*}+\mathbf{BB}^{*}=0$ are as follows [20, 22]:

- Bartels-Stewart method,
- Smith method,
- Alternating Direction Implicite (ADI) method,
- Sign Function method.

2.1. Bartels-Stewart method

The algorithm includes the following steps. Firstly, Schur decomposition is applied to transform matrices **A** and $\mathbf{A}^{*}$ into upper triangular forms [22]:

$$\mathbf{R}_1=\mathbf{UAU}^{*},\ \mathbf{R}_2=\mathbf{VA}^{*}\mathbf{V}^{*} \tag{4}$$

and then matrix $\tilde{\mathbf{D}}$ is formed according to the following relation:

$$\tilde{\mathbf{D}}=\mathbf{U}^{*}(\mathbf{BB}^{*})\mathbf{V} \tag{5}$$

After transformations, the Lyapunov equation receives the following form:

$$\mathbf{R}_1\tilde{\mathbf{X}}+\tilde{\mathbf{X}}\mathbf{R}_2+\tilde{\mathbf{D}}=0\,. \tag{6}$$

Due to a triangular form of the matrices $\mathbf{R}_1$ and $\mathbf{R}_2$, equation (4) may be effectively solved by application of the following relation:

$$(\mathbf{R}_1+r_{kk}^{(2)}\mathbf{I})\tilde{\mathbf{x}}_k=-\tilde{\mathbf{d}}_k-\sum_{i=1}^{k-1}\tilde{\mathbf{x}}_i r_{ik}^{(2)} \tag{7}$$

where $r^{(2)}$ denotes an element of matrix $\mathbf{R}_2$ in a row and column defined by the subscripts, $\tilde{\mathbf{d}}_k$ denotes *k-th* column of matrix $\tilde{\mathbf{D}}$, $\tilde{\mathbf{x}}_i$ denotes *i-th* column of matrix $\tilde{\mathbf{X}}$. In the final step, the matrix, which is the solution of the Lyapunov equation, is determined with the following relation:

$$\mathbf{gramian}=\mathbf{U}\tilde{\mathbf{X}}\mathbf{V}^{*} \tag{8}$$

Figure 1 presents a program for MATLAB environment determining the controllability gramian by means of Bartles-Stewart method.

2.2. Smith method

Smith method is based on a conversion of the Lyapunov equation by means of a bilinear transformation into a discrete form presented below [9]:

$$\mathbf{VPV}^{*}-\mathbf{P}+\mathbf{W}=0\,, \tag{9}$$

where:

$$\mathbf{V} = (q\mathbf{I} - \mathbf{A}^*)^{-1}(q\mathbf{I} + \mathbf{A}^*), \tag{10}$$

$$\mathbf{W} = 2q(q\mathbf{I} - \mathbf{A}^*)^{-1}\mathbf{B}\mathbf{B}^*(q\mathbf{I} - \mathbf{A})^{-1}. \tag{11}$$

Assuming that matrix **A** is asymptotically stable, the consecutive approximations of the gramian values for parameter q>0 are obtained from the following relation:

$$\mathbf{P}_{K+1} = \mathbf{P}_K + \mathbf{V}^{2^K}\mathbf{P}_K(\mathbf{V}^{2^K})^*, \tag{12}$$

for

$$\mathbf{P}_0 = 2q(q\mathbf{I} - \mathbf{A}^*)^{-1}\mathbf{B}\mathbf{B}^*(q\mathbf{I} - \mathbf{A})^{-1}. \tag{13}$$

The number of iterations required for desired accuracy depends on an appropriate selection of parameter *q* (as proven experimentally, the most favourable *q* value equals 0.1). Figure 2 presents a program for MATLAB environment determining the controllability gramian by means of Smith method.

```
1.      [rows, cols]=size(A);
2.      [Q1,R1]=m_schur(A);
3.      [Q2,R2]=m_schur(A');
4.      D=Q1'*B*Q2;
5.      X=zeros(rows,cols);
6.      for counter=1:rows;
7.        result=zeros(rows,1);
8.        if counter>1
9.          R(1:counter-1, 1)=R2(1:counter-1, counter);
10.         result=(X(:, 1:counter-1))*R;
11.       end
12.       X(:,counter)=(R1+R2(counter,counter)*eye(rows, cols))\(-
D(:,counter)-result);
13.     end
14.     P=Q1*X*Q2';
```

Figure 1. The controllability gramian computation by means of Bartels-Stewart method

```
1.      A=A';
2.      q=0.1;
3.      [rows,cols]=size(A);
4.      V=(q*eye(rows,cols)-A')^-1*(q*eye(rows,cols)+A');
5.      W=2*q*(q*eye(rows,cols)-A')^-1*Q*(q*eye(rows,cols)-A)^-1;
6.      P=W;
7.      futher=true;
8.      while futher
9.        N=V*P*V';
10.       P=P+N;
11.       if(max(max(abs(N)))<=epsilon)
12.         futher=false;
13.       end
14.       V=V*V;
15.     end
```

Figure 2. The controllability gramian computation by means of Smith method

2.3. ADI method

Alternating Direction Implicit (ADI) method is a method for determining the gramian values in accordance with the following relations [17]:

$$\begin{aligned} &\mathbf{X}_0 = 0 \\ &(\mathbf{A} + p_j\mathbf{I})\mathbf{X}_{j-1/2} = -\mathbf{B}\mathbf{B}^* - \mathbf{X}_{j-1}(\mathbf{A}^* - p_j\mathbf{I}) \\ &(\mathbf{A} + p_j\mathbf{I})\mathbf{X}_j = -\mathbf{B}\mathbf{B}^* - \mathbf{X}_{j-1/2}^*(\mathbf{A}^* - p_j\mathbf{I}) \end{aligned} \tag{14}$$

where coefficients p_j determine the algorithm convergence degree. For a negative definite Hermitian matrix **A**, coefficients p_j are determined according to the relation[6]:

$$\{p_1, p_2, ..., p_n\} = \min_{\{p_1,p_2,...,p_n\}\in C_-} \max_{\lambda\in eig(A)} \frac{|(\lambda - p_1^*)...(\lambda - p_n^*)|}{|(\lambda + p_1)...(\lambda + p_n)|}, \tag{15}$$

where eig(**A**) denotes the eigenvalues of the matrix **A**.

In case of a non-symmetric and negative definite matrix **A**, T. Penzl proposed a heuristic algorithm for selecting optimal coefficients p_j [6, 17].

$$\{p_1, p_2, ..., p_n\} = \min_{\{p_1,p_2,...,p_n\}\in eig(A)} \max_{\lambda\in eig(A)} \frac{|(\lambda - p_1^*)...(\lambda - p_n^*)|}{|(\lambda + p_1)...(\lambda + p_n)|} \tag{16}$$

Figure 3 presents a program for MATLAB environment determining the controllability gramian by means of ADI method.

```
1.     [rows, cols]=size(A);
2.     P=zeros(rows, rows);
3.     parameters=[];
4.     lambdas=[];
5.     [parameters,    lambdas]=adi_parameters3(A,    parameters,    lambdas,
iterations);
6.     futher=true;
7.     i=0;
8.     while(futher)
9.       i=i+1;
10.      if (i>length(parameters))
11.        [parameters,   lambdas]=adi_parameters3(A,   parameters,   lambdas,
iterations);
12.      end
13.      if (mod(i, iterations)==0)
14.        P_old=P;
15.      end
16.      P1_2=(A+parameters(i)*eye(rows))\(-B-P*(A'-
parameters(i)*eye(rows)));
17.      P=(A+parameters(i)*eye(rows))\(-B-P1_2'*(A'-
parameters(i)*eye(rows)));
18.      if(mod(i,iterations)==0);
19.        if (max(max(abs(P_old-P)))<epsilon)
20.          futher=false;
21.        end
22.        P_old=[];
23.      end
24. end
```

Figure 3. The controllability gramian computation by means of ADI method.

2.4. Sign Function method

The solution of the Lyapunov equation comes down to solving the following equation [5]:

$$sign(\mathbf{Z}) = \begin{bmatrix} -\mathbf{I} & 2\mathbf{P} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \tag{17}$$

$$\text{where} \qquad \mathbf{Z} = \begin{bmatrix} \mathbf{A} & \mathbf{BB}^* \\ 0 & -\mathbf{A}^* \end{bmatrix}, \tag{18}$$

$$sign(\mathbf{Z}) = \mathbf{S}^{-1} \begin{bmatrix} -\mathbf{I}_{l*l} & \mathbf{0} \\ \mathbf{0} & +\mathbf{I}_{(n-l)*(n-l)} \end{bmatrix} \mathbf{S} \tag{19}$$

S – Jordan decomposition matrix.

The value of the Sign Function is determined by applying the Newton's iteration method $\mathbf{Z}_{K+1} = \frac{1}{2}(\mathbf{Z}_K + \mathbf{Z}_K^{-1})$. However, it is more preferred to perform two iterations on the matrices of the size *n*n* instead of one iteration on matrices of the size *2n*2n* [18]:

$$\begin{aligned} A_0 &= A \\ P_0 &= BB^* \\ A_{K+1} &= \frac{1}{2}\left(A_K + A_K^{-1}\right) \\ P_{K+1} &= \frac{1}{2}\left(P_K + \left(A_K^*\right)^{-1} P_K A_K^{-1}\right) \end{aligned} \tag{20}$$

Figure 4 presents a program for MATLAB environment determining the controllability gramian by means of the Sign Function.

```
1.    A=A';
2.    futher=true;
3.    N=zeros(size(A));
4.    iterations=10;
5.    while(futher)
6.      inverse=A^-1;
7.      N=inverse'*C*inverse;
8.      C= 1/2*(C+N);
9.      A=1/2*(A+inverse);
10.     if( (max(max(A-inverse))<epsilon) )
11.       futher=false;
12.     end
13.   end
14.   gramian=1/2*N;
```

Figure 4. The controllability gramian computation by means of the Sign Function

3. NVIDIA graphics hardware and programming libraries

3.1. Fermi architecture

Graphics Processor Unit (GPU) based on Fermi architecture may be equipped with up to 512 CUDA cores (Compute Unified Device Architecture). Every core performs one 32-bit floating-point operation (SINGLE format) during one cycle of a processor. Performing the 64-bit floating-point operation (DOUBLE format) requires two CUDA cores to be involved and is completed during one cycle of the GPU [11]. CUDA cores are grouped into 16 streaming multiprocessors (SM) and each contains 32 CUDA cores. Currently, Tesla M2090 offered by NVIDIA has the highest computational capability and it delivers 1.331 TFLOPs for single precision performance and 665 GFLOPs for double precision performance. The card offers 6144 MB of GDDR5 memory [12].

3.2. Programming model

NVIDIA CUDA programming platform provides a multi–level programming model, which enables a programmer to concentrate mainly on writing an algorithm instead of wasting time on details related to parallel operations executed in many cores [11, 12]. The basic elements of the program are the kernels. An application or a function contains at least one kernel. The kernel's code may be written in C language with extra instructions determining a parallel execution instead of traditional program loops.

After compilation, the kernels contain many threads which execute simultaneously the same program (Single Instruction Multi Threads architecture). The threads are grouped into threads block containing up to 1536 threads. Threads of a given block are executed on a single SM, which enables the individual threads to be synchronized and share the common data. Thread blocks are divided into warps with 32 threads each. The warp is a basic unit of a SM allocation and in Fermi architecture two warps may be simultaneously active on a SM.

Thread blocks are grouped into grids, which execute individual kernels. At a given moment of time the whole processor is dedicated to only one application, which may contain many kernels. Fermi architecture supports parallel execution of many kernels of the given application. Each kernel may be assigned to one or many SMs.

The GPU is equipped with a scheduler, which manages 1536 simultaneously active threads on a single SM, at kernels number amounting 16.

3.3. Programmer libraries

NVIDIA provides CUDA package for C++ language, which contains the following libraries [10, 19]:

- CUBLAS – basic linear algebra routines. It is an equivalent to a popular BLAS library (Basic Linear Algebra Subroutines) [13],
- CUFFT – Fast Fourier Transform [14],
- CUSPARSE – basic linear algebra routines on sparse matrices [15],
- CURAND – generation of pseudorandom numbers [16].

CULA library (Linear Algebra PACKage interface for CUDA-enabled NVIDIA graphics processing units) provides much more capabilities. It contains a significant number of methods provided by the LAPACK library[7, 8]. CULA library methods have been prepared in four variants:

- real SINGLE data type,
- complex SINGLE data type,
- real DOUBLE data type,
- complex DOUBLE data type.

4. Parallel solution of the Lyapunov equation by means of GPU

The class presented in the paper enables solving the Lyapunov equation, with an application of NVIDIA GPU, by applying four standard methods: Bartels-Stewart, Smith, Alternating Direction Implicite (ADI), Sign Function.

The routines have been developed in C++ language using CUDA and CULA libraries. The class enables to perform operations on matrices with real or complex elements of SINGLE or DOUBLE type. The methods of presented class have been divided into four groups as follows:

- methods related to forming and copying matrices,

- methods for the basic matrix operations, implemented as overloaded operators (+, -, ++, ^, *, /),
- methods for performing selected linear algebra operations,
- methods for computing the solution of the Lyapunov equation with use of the Bartels-Stewart, Smith, ADI and the Sign Function algorithms.

4.1. Methods related to forming and copying matrices

The methods related to formation and copying the matrices have been presented in Table 1.

Table 1. Methods related to formation and copying the matrices

Method	Description
ZEROS_CARD	Formation of a matrix consisting of all zeros of a given type and size
DIAG_CARD	Formation of a diagonal matrix with a determined value on a diagonal
ONES	Formation of an identity matrix
DUP	Copying matrix content into other object
COPY	Copying a definite number of matrix rows
PART_COPY	Copying a part of a matrix
CREATE_MATRIX	Formation of a matrix of a definite type and size
TO_CARD	Copying a matrix from the host memory into a GPU memory
FROM_CARD	Copying a matrix from a GPU memory into a host memory
DEL_DEV	Deallocation of memory on the GPU containing the matrices
DEL_HOST	Deallocation of memory on the host containing the matrices

4.2. Methods for the basic matrix operations

Methods for the basic matrix operations implemented as overloaded operators include the following operations: addition (operator +), substraction (operator -), multiplication (operator *), transposition (operator ++), determination of the power of the matrix (operator ^), matrix multiplication by scalar (operator *), solving the system of equations (operator /), solving the system of equations with triangular matrix **A** (operator /=).

4.3. Methods for performing selected linear algebra operations

Methods related to selected linear algebra operations have been presented in Table 2.

Table 2. Methods for the basic linear algebra operations

Method	Description
MUL1	Performing the operation $\mathbf{X} = \alpha\mathbf{AB}$ by applying the function gemm of CULA library
MUL2	Performing the operation $\mathbf{X} = \alpha\mathbf{AB}^*$ by applying the function gemm of CULA library
INV	Determining the inverse of matrix by applying the functions getrf and getri of CULA library
X_PLUS_A_MUL_Y	Performing the operation $\mathbf{X} = \mathbf{X} + a_{i,j}\mathbf{Y}$ where $a_{i,j}$ denotes an element of matrix A in i-th row and j-th column
DIAG_ADD	Performing the operation $\mathbf{X} = \mathbf{A} + \alpha\mathbf{I}$ where α is a scalar coefficient and **I** denotes an identity matrix of the size of matrix **X**
DIAG_SUB	Performing the operation $\mathbf{X} = \mathbf{X} - \alpha\mathbf{I}$ where α is a scalar coefficient and **I** denotes an identity matrix of the size of matrix **X**

ODD	Performing the operation $\mathbf{X} = -\mathbf{X}$
QR	QR decomposition of a matrix. The method employs the function `geqrf` of CULA library
Q_QR	Determining matrix **Q** of QR decomposition. The method employs the function `ungqr` and `orgqr` of CULA library
EIG_FULL	Calculating all of eigenvalues and optionally the right or left eigenvectors of the matrix. The method employs the function `geev` of CULA library
MY_SCHUR	Determining Schur's form of a matrix by applying an orthonormalization of eigenvectors
MY_SCHUR_FULL	Determining Schur decomposition of a matrix by applying an orthonormalization of eigenvectors

4.4. Methods for the solution of the Lyapunov equation

The class contains 4 functions determining the controllability and observability gramians by means of the GPU based on the solution of the Lyapunov equation obtained by applying the four methods: Bartels-Stewart, Smith, ADI and Sign Function.

In appendix A.1 a routine for the solution of Lyapunov equation using the Bartels-Stewart method is presented. The function performs the following oparations:

Line no. 6 - Schur decomposition of matrix **A,**
8 - Schur decomposition of matrix $\mathbf{A}^*$,
11-15 - determining the matrix $\tilde{\mathbf{D}} = \mathbf{U}^*(\mathbf{B}\mathbf{B}^*)\mathbf{V}$,
20-49 - determining the consecutive columns of converted Lyapunov equation,
22-33 - determining the expression $\sum_{i=1}^{k-1} \tilde{\mathbf{x}}_i r_{ik}^{(2)}$,
36 - determining the expression $(\mathbf{R}_1 + r_{kk}^{(2)}\mathbf{I})$,
38 - determining vector $\tilde{\mathbf{d}}_k$,
41 - determining the expression $-\tilde{\mathbf{d}}_k - \sum_{i=1}^{k-1} \tilde{\mathbf{x}}_i r_{ik}^{(2)}$,
45 - determining *k-th* column of the equation solution according to the following relation: $(\mathbf{R}_1 + r_{kk}^{(2)}\mathbf{I})\tilde{\mathbf{x}}_k = (-\tilde{\mathbf{d}}_k - \sum_{i=1}^{k-1} \tilde{\mathbf{x}}_i r_{ik}^{(2)})$,
50-59 - determining Lyapunov equation solution.

In appendix A.2 a routine for the solution of the Lyapunov equation using the Smith method is presented. The function performs the following operations:

line no. 3 - 25 - conversion of the Lyapunov equation to Stein equation,
6 - computation of the expression $(q\mathbf{I} - \mathbf{A}^*)$,
7 - computation of the expression $(q\mathbf{I} + \mathbf{A}^*)$,
9 - 12 - the expression $(q\mathbf{I} - \mathbf{A}^*)^{-1}(q\mathbf{I} + \mathbf{A}^*)$ is being computed,
15 - the expression $2q\mathbf{B}\mathbf{B}^*$ is being computed,
17 - 22 - computation of the expression $(q\mathbf{I} - \mathbf{A})^{-1}$,
25 - computation of the expression $2q(q\mathbf{I} - \mathbf{A}^*)^{-1}\mathbf{B}\mathbf{B}^*(q\mathbf{I} - \mathbf{A})^{-1}$,

31-76 - loop comprising the computations results for consecutive approximations of Stein equation solution,

36, 54 - computation of the value of the expression $\mathbf{V}^{2^K}$,

38, 56 - computation of the expression $\mathbf{V}^{2^K}\mathbf{P}_K$,

39, 57 - the expression $\mathbf{V}^{2^K}\mathbf{P}_K(\mathbf{V}^{2^K})^*$ is being computed,

41, 59 - computation of the consecutive approximation $\mathbf{P}_{K+1}$,

45, 63 - condition for an algorithm interruption.

In appendix A.3 a routine for the solution of the Lyapunov equation using the ADI method is presented. The function performs the following operations:

line no. 12 - determining the initial shift parameters for the first iterations,

15-70 - computation of consecutive matrix approximations of the Lyapunov equation solution in a loop,

27 - addition of a diagonal matrix $p_j\mathbf{I}$ to matrix $\mathbf{A}$,

30 - computation of the expression $(\mathbf{A}^* - p_j\mathbf{I})$,

32 - computation of the product of matrices $\mathbf{X}_{j-1}(\mathbf{A}^* - p_j\mathbf{I})$,

36 - determination of matrix $-\mathbf{B}\mathbf{B}^*$,

37 - computation of the difference $-\mathbf{B}\mathbf{B}^* - \mathbf{X}_{j-1}(\mathbf{A}^* - p_j\mathbf{I})$,

40 - determination of matrix $\mathbf{X}_{j-1/2}$,

43-45 - computation of the expression $\mathbf{X}^*_{j-1/2}(\mathbf{A}^* - p_j\mathbf{I})$,

49 - determination of matrix $-\mathbf{B}\mathbf{B}^*$,

50 - computation of the expression $-\mathbf{B}\mathbf{B}^* - \mathbf{X}^*_{j-1/2}(\mathbf{A}^* - p_j\mathbf{I})$,

53 - determination of the consecutive approximation X of the Lyapunov equation,

56-65 - condition for an algorithm interruption.

In appendix A.4 a routine for the solution of the Lyapunov equation using the Sign Function is presented. The function performs the following operations:

line no. 8-51 - determination of the consecutive approximations of the Lyapunov equation solution in a loop,

11 - computation of matrix $\mathbf{A}_K^{-1}$,

13 - evaluation of the expression $\mathbf{P}_K\mathbf{A}_K{}^{-1}$,

15-17 - evaluation of the expression $\left(\mathbf{A}_K{}^*\right)^{-1}\mathbf{P}_K\mathbf{A}_K{}^{-1}$,

20 - computation of the expression $\mathbf{P}_K + \left(\mathbf{A}_K{}^*\right)^{-1}\mathbf{P}_K\mathbf{A}_K{}^{-1}$,

26 - condition for an algorithm interruption,

41 - evaluation of $\mathbf{P}_K$ for the consecutive iteration,

43 - computation of the expression $\mathbf{A}_K + \mathbf{A}_K^{-1}$,

46 - evaluation of $\mathbf{A}_K$ for the consecutive iteration.

5. Experimental results

The programs developed for both MATLAB environment and NVIDIA GPU have been applied to determine the controllability gramians of linearized mathematical models of the one-phase zone of the once-through steam boiler BP-1150, which forms a part of the screen pipes of evaporator [21]. The model is described by a system of partial differential equations for the three spatial variables: the length, as well as the radius and circumference of the tubes. The lumped parameters model is formed through the application of the finite elements method and obtained system of ordinary differential equations is linearized at an operating point, providing the model (1). The computations have been done for the models of the orders between 30 and 4200, which results from an application of a different density discretisation (different number of finite elements). The structure of matrices **A** of obtained linear models is presented in Figure 5. The model is described with a sparse matrix with nonzero elements concentrated along the main diagonal.

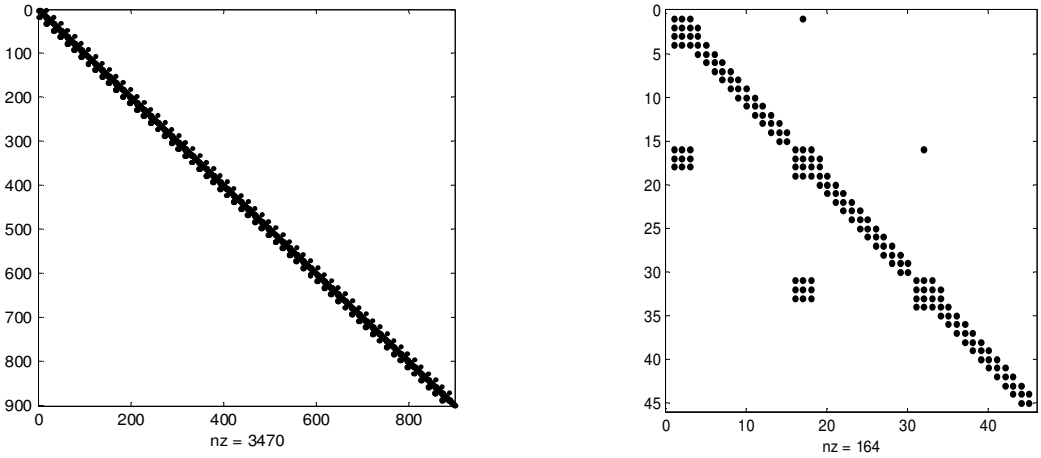

Figure 5. Matrix A structure of the 900-th order model

5.1. Computations in MATLAB environment

The computations of the controllability gramians for different model orders, have been done using four methods: Smith, Sign Function, Bartels-Stewart and ADI. Additionally, for the purpose of comparison, computations using `lyap` function of MATLAB environment (for R2007b and 2011a versions) have been done. Figure 6 shows the execution times vs. the model order for MATLAB environment. Computations have been done using MATLAB R2007b and computer with processor Core2 Duo E6750 and DRAM 4 GB.

All gramian determination methods based on the Lyapunov equation have a computational cost of $O(n^3)$ [1, 5]. Execution times for Smith and Sign Function methods are very similar, whereas Bartels-Stewart and ADI methods require a few times longer execution times. Moreover, Bartels-Stewart and ADI methods require much more storage than other methods, which limits the ability to compute the controllability grammians for the model orders not higher than 2400 and 2100 respectively.

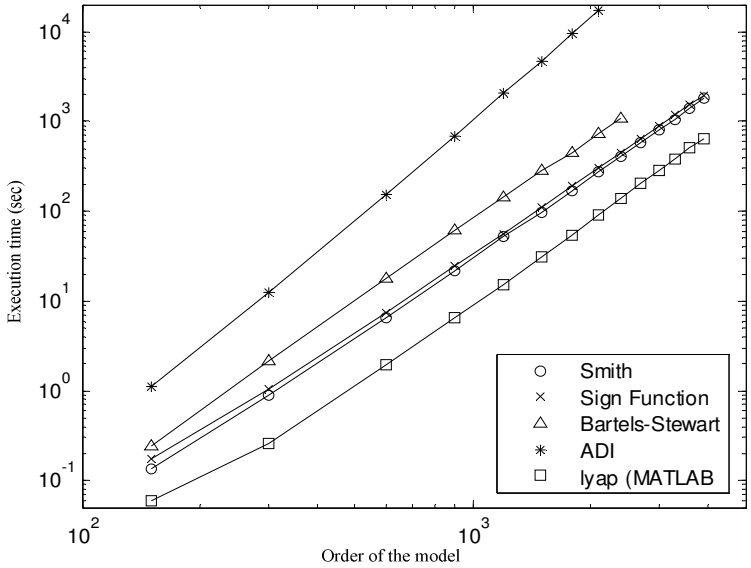

Figure 6. Execution time vs. model order for controllability gramians computation for MATLAB R2007b

The methods were also examined with use of parallel environment of MATLAB R2011a. The best algorithms – Smith and Sign Function are compared with MATLAB's `lyap` function in Figure 7.

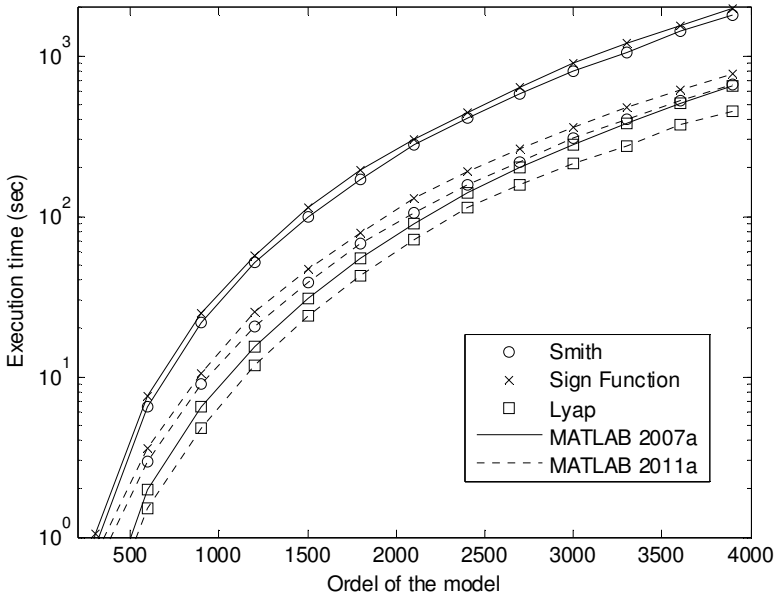

Figure 7. Comparison of execution times for chosen methods for MATLAB R2007b and R2011a

5.2. NVIDIA GPU computations

Described gramian determination methods have been implemented in C++ language with an application of CUDA and CULA libraries and the graphics hardware NVIDIA GTX-465, equipped with 1GB memory. The specifications of applied graphics hardware are presented in Table 3.

Table 3. Specifications of the graphics hardware NVIDIA GTX-465

NVIDIA GTX-465 specifications	
Technological process (nm)	40
Number of transistors in millions	3200
Motherboard interconnect	PCIe 2.0 x16
Memory size	1024 MB GDDR5
Memory bandwidth	102.6 GB/s
Core number	352
Core lock	607 MHz

Figure 8 presents execution time relating to the controllability gramians determination by means of the GPU for four analysed methods. For the highest order models, the time may be approximated by a function $\tau = a*n^3$. For the lower order models the execution time is significantly longer (marked with dotted line in Figure 8), because much of the execution time is consumed by the preparations to perform computations by the GPU.

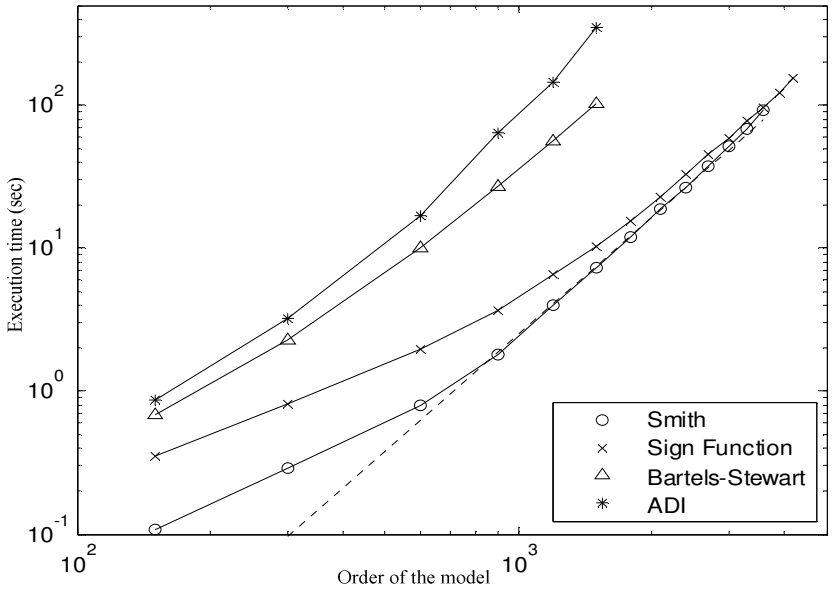

Figure 8. Execution time vs. model order for controllability gramians computation for NVIDIA GPU

Computations show, that the Smith method is the most advantageous in case of the model order lower than 3600. However, it can not be applied for greater orders due to memory limitations. Then, the Sign Function method can be applied instead (if the order of the models equals or is lower than 4200). Bartels-Stewart and ADI methods require much longer execution time and have significant memory limitations (the highest order of the model is 1500). Figure 9 presents comparison of execution times obtained for MATLAB R2007b environment and GPU for Smith and Sign Function methods. Maximum acceleration of computations reaches value 20 for models of high orders.

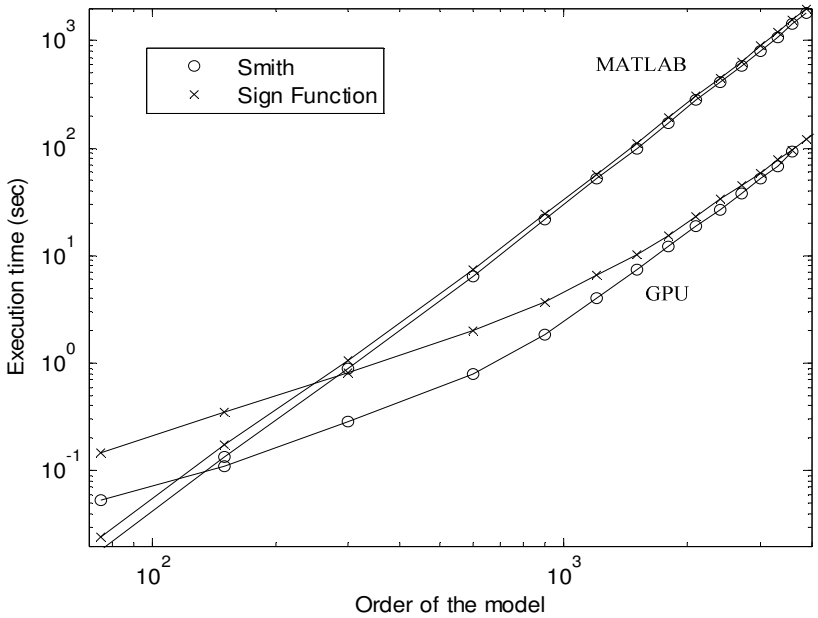

Figure 9. Comparison of execution times for MATLAB R2007b environment and GPU

The comparison of Smith method using GPU with the MATLAB's `lyap` function is shown in Figure 10.

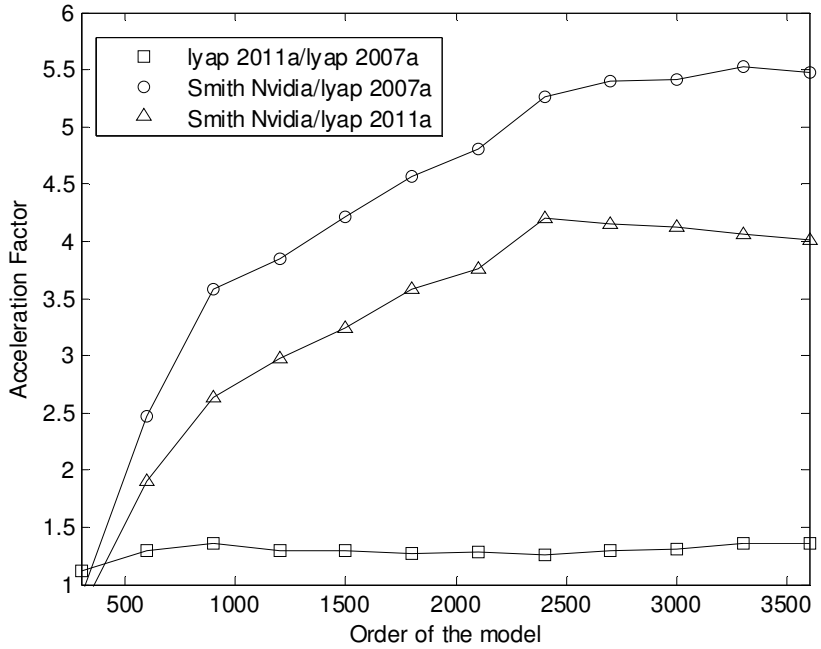

Figure 10. Comparison of acceleration factors for Smith method obtained for Nvidia and MATLAB environments

5.3. The other methods and techniques

The methods presented in the paper were chosen for purpose of computing controllability and observability gramians of linearized mathematical models of the one-phase zone of

the once-through steam boiler BP-1150. The Smith and the Sign Function methods gave the best performance in comparison with the others algorithms.

Besides the presented algorithms, there is a group of low rank methods, which take advantage from low rank of matrices **B** in case of controllability gramian and **C** in case of observability gramian [3]. These methods weren't efficient in case of orders of the examined models. The comparison of Smith, Sign Function methods, and their low rank equivalents is presented in Figure 11.

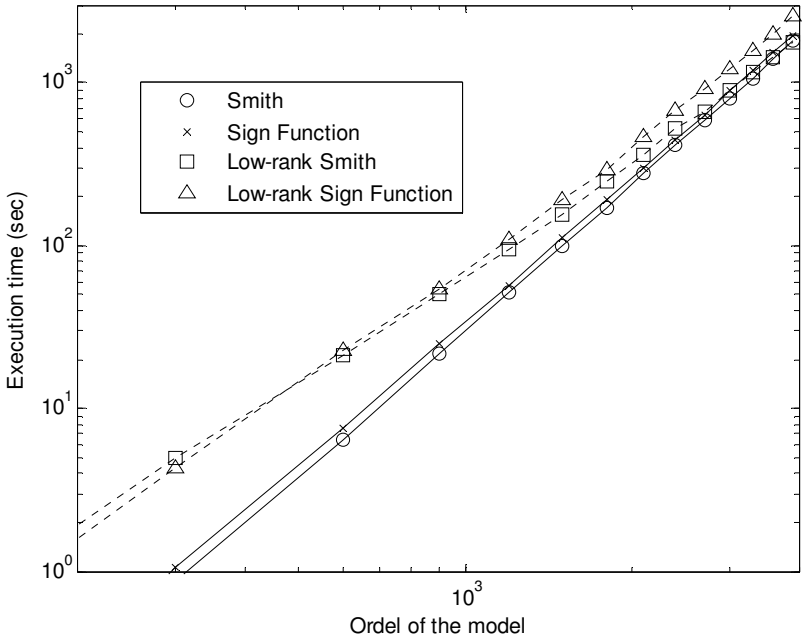

Figure 11. Comparison of execution times for Sign Function, Smith methods and their low rank equivalents in MATLAB R2007b environment

In the literature, there are examples of using hybrid GPU-CPU Lyapunov solvers [3,4]. The algorithms presented in the paper were implemented to be executed entirely on GPU to avoid any unnecessary data transfer between the main and the graphical card memory. This approach eliminates the bottleneck of systems using computing cards. Additionally this approach let us to compare efficiency of the presented methods executed with only use of GPU with the same algorithms implemented in MATLAB environment.

The methods presented in the paper use only double-precision arithmetic to obtain the high quality results. In the literature, there are examples of using mixed-precision algorithms in which a few first iterations are executed with use of a single-precision arithmetic [2].

6. Conclusions

An issue of applicability of parallel computations with the multicore GPU for the purpose of determining the controllability gramians of LTI high order models has been analyzed in the paper. The process of determining the gramians by means of the Lyapunov equation is characterised by a high computational cost ($O(n^3)$), which results in long execution time in case of high order models. It is necessary to determine the controllability and observability gramians for the linear models described by matrices **A**, **B**, **C**, **D**, during the process of a model order reduction based on the Singular Value Decomposition (SVD) [1].

The model order reduction is required particularly to analyze distributed parameter systems (described by partial differential equations) by means of the finite elements method [21]. Obtained lumped parameter models (described by ordinary differential equations) are characterized by very high orders (often higher than 10^5). To make it possible to analyse the properties of dynamic complex plants and with the purpose of designing the control systems, it has become crucial to reduce the model order and at the same time guarantee a specified scope of adequacy of reduced model.

Presented algorithms and programs enable application of the GPU to reduce LTI models and significantly minimize the computational cost. Obtained acceleration factor using GPU, in case of the same algorithms, reach the value 20 and 13 in comparison to MATLAB R2007b and R2011a respectively. The methods executed on graphical card were also 5.5 and 3.6 times faster than native `lyap` function of MATLAB R2007b and R2011a respectively. It allows for solving much bigger tasks of model reduction within a short time.

Storage capacity of graphics hardware equal 1GB enables gramians computation for models with order lower or equal to 4000. Since it has been necessary to reduce models of the order much higher than 4000, it is required to apply a multilevel substructuring technique, which will make it possible to reduce the models of any order [21].

References

[1] Antoulas A.: *Approximation of large-scale dynamical systems.* Society for Industrial and Applied Mathematics, Philadelphia, 2005.

[2] Benner P., Ezzatti P., Kressner D., Quintana-Ortí E. S., Remón A., *A mixed-precision algorithm for the solution of Lyapunov equations on hybrid CPU-GPU platforms.* Parallel Computing, Vol. 37(8), pp. 439-450, 2011

[3] Benner P., Ezzatti P., Kressner D., Quintana-Ortí E. S., Remón A., *Accelerating model reduction of large linear systems with graphics processors*. Lecture Notes in Computer Science 7134, State of the Art in Scientific and Parallel Computing – PARA 200, pp. 88-97, 2012

[4] Benner P., Ezzatti P., Quintana-Ortí E. S., Remón A., *Using hybrid CPU-GPU platforms to accelerate the computation of the matrix sign function.* Lecture Notes in Computer Science 6043, 7th Int. Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks – HeteroPar'09, pp. 132-139, 2009

[5] Benner P., Quintana-Ortí E. S., Quintana-Ortí G.: *State-Space Truncation Methods for parallel model reduction of large-scale systems*, Parallel Computing, Vol. 29, pp. 1701-1722, 2003

[6] Benner P., Li R.-C., Truhar N.: *On the ADI method for Sylvester equations*. J. Computational Applied Mathematics, Vol. 233, No. 4, pp. 1035–1045, 2009.

[7] EM Photonics, Inc.: *CULA programmer's guide,* EM Photonics, Inc., 2011

[8] EM Photonics, Inc.: *CULA reference manual,* EM Photonics, Inc., 2011

[9] Gajić Z.: *Lyapunov matrix equation in system stability and control.* Academic Press, San Diego, 1995.

[10] Hwu Wen-Mei W., Krik D..: *Programming massively parallel processors,* Elsevir Ltd. Oxford, 2010

[11] Nvidia Corporation: *Whitepaper NVIDIA's next generation CUDA compute architecture: Fermi*, Nvidia Corporation, 2009

[12] Nvidia Corporation: *Tesla M-CLASS GPU computing modules. Fastest parallel processors for accelerating science,* Nvidia Corporation, 2011

[13] Nvidia Corporation: *CUBLAS Library,* Nvidia Corporation, 2010

[14] Nvidia Corporation: *CUFFT Library,* Nvidia Corporation, 2010
[15] Nvidia Corporation: *CUSPARSE Library,* Nvidia Corporation, 2010
[16] Nvidia Corporation: *CURAND Library,* Nvidia Corporation, 2010
[17] Penzl T.: *A cyclic low rank Smith method for large sparse Lyapunov equations with applications in model reduction and optimal control.* Technische Universitat Chemnitz, 1998.
[18] Quintana-Ortí E., Geijn R.: *Specialized parallel algorithms for solving linear matrix equatios in control theory*, Journal of Parallel and Distributed Computing, Vol. 61, pp. 1489-1504, 2001
[19] Sanders J., Kandrot E.: *CUDA by example. An introduction to general-purpose GPU programming,* Addison-Wesley, 2010
[20] Schilders W.: *Model order reduction theory, research aspects and applications.* Springer, Berlin, 2008.
[21] Stanisławski W., Rydel M. Hierarchical mathematical models of complex plants on the basis of power boiler example. *Archives of Control Science Volume* 20(LVI), 2010 *No.* 4*, pp. 381–416, 2010*
[22] Zhou. Y.: *Numerical methods for large scale matrix equations with applications in LTI system model reduction.* Rice University, Houston, 2002.

Appendix

A.1. The controllability and observability gramians computation with Bartels-Stewart method.

```
1.   void CulyaMatrix::Bartels5(CulyaMatrix B)
2.   {
3.     TSchur SA, SA_transpose;
4.     CulyaMatrix A_transpose;
5.     A_transpose=(*this)++;
6.     SA=(*this).my_schur_full();
7.     (*this).del_dev();
8.     SA_transpose=A_transpose.my_schur_full();
9.     A_transpose.del_dev();
10.    CulyaMatrix D1, D2;
11.    D1=SA.U++;
12.    D2=D1*B;
13.    D1.del_dev();
14.    B.del_dev();
15.    B=D2*SA_transpose.U;
16.    D2.del_dev();
17.    CulyaMatrix X;
18.    X= zeros_card(B.rows, B.cols, B.type1, B.type2);
19.    CulyaMatrix result;
20.    for (long int counter=0; counter<B.rows; counter++)
21.    {
22.    if (counter==0) result=zeros_card(B.rows, 1, B.type1, B.type2);
23.    CulyaMatrix vect, vect2;
24.    if (counter>0)
25.    {
26.      vect=zeros_card(counter, 1, B.type1, B.type2);
27.      vect.part_copy(vect,  SA_transpose.S,  0,    0,  0,  counter,  counter-1,
counter);
28.      vect2=zeros_card(rows, counter, B.type1, B.type2);
29.      vect2.part_copy(vect2, X, 0, 0, 0,0, X.rows-1, counter-1);
30.      result=vect2*vect;
31.      vect.del_dev();
32.      vect2.del_dev();
33.    }
```

```
34.     CulyaMatrix R;
35.     R=SA.S.dup();
36.       R.diag_add(SA_transpose.S, counter, counter);
37.     CulyaMatrix D;
38.       D=B.create_vec_from_col(counter);
39.     D.odd();
40.     CulyaMatrix D3;
41.     D3=D-result;
42.     result.del_dev();
43.     D.del_dev();
44.     CulyaMatrix D4;
45.     R/=D3;
46.     R.del_dev();
47.     X.part_copy(X, D3, 0,  counter, 0, 0, rows-1, 0);
48.     D3.del_dev();
49.     }
50.     SA.S.del_dev();
51.     SA_transpose.S.del_dev();
52.     B.del_dev();
53.     CulyaMatrix Q2;
54.     Q2=SA_transpose.U++;
55.     SA_transpose.U.del_dev();
56.     CulyaMatrix Q3;
57.     Q3=SA.U*X;
58.     SA.U.del_dev();
59.     (*this)=Q3*Q2;
60.     Q2.del_dev();
61.     Q3.del_dev();
}
```

A.2. The controllability and observability gramians computation using Smith method

```
1.    CulyaMatrix CulyaMatrix::smith2(CulyaMatrix B, double epsilon)
2.    {
3.      CulyaMatrix diag;
4.      diag=(*this).diag(0.1,0);
5.      CulyaMatrix L, P;
6.      L=diag-(*this);
7.      P=diag+(*this);
8.      CulyaMatrix L1;
9.      L1=L^-1;
10.     L.del_dev();
11.     CulyaMatrix V;
12.     V=L1*P;
13.     P.del_dev();
14.     CulyaMatrix L2;
15.     L2=L1.mul1(B, 2*0.1);
16.     L1.del_dev();
17.     L1=(*this)++;
18.     P=diag-L1;
19.     diag.del_dev();
20.     diag.del_host();
21.     L1.del_dev();
22.     L1=P^-1;
23.     P.del_dev();
24.     CulyaMatrix W;
25.     W=L2*L1;
26.     L1.del_dev();
27.     L2.del_dev();
28.     L1=W;
29.     CulyaMatrix POWER, L3;
30.     bool value;
31.     for (int i=0; i<1000; i++)
32.     {
33.       if (i%2==0)
34.      {
```

```
35.       value=true;
36.       if(i>0) POWER=V*V; else POWER=V.dup();
37.       V.del_dev();
38.       L2=POWER*L1;
39.       L3=L2.mul2(POWER);
40.       L2.del_dev();
41.       L2=L1+L3;
42.       L1.del_dev();
43.       double max;
44.       max=L3.max();
45.         if(max<epsilon)
46.       {
47.         L3.del_dev();
48.         break;
49.       }
50.       L3.del_dev();
51.     } else
52.    {
53.      value=false;
54.      V=POWER*POWER;
55.      POWER.del_dev();
56.      L3=V*L2;
57.      L1=L3.mul2(V);
58.      L3.del_dev();
59.      L3=L1+L2;
60.      L1.del_dev();
61.      double max;
62.      max=L2.max();
63.      if(max<epsilon)
64.      {
65.        L2.del_dev();
66.        L1=L3;
67.        L3.culaMacierz=NULL;
68.        L3.CulaMacierz=NULL;
69.        break;
70.      }
71.      L2.del_dev();
72.      L1=L3;
73.      L3.culaMacierz=NULL;
74.      L3.CulaMacierz=NULL;
75.    }
76.    }
77.    POWER.del_dev();
78.    V.del_dev();
79.    return value?L2:L1;
80.   }
```

A.3. The controllability and observability gramians computation using ADI method.

```
1.    void CulyaMatrix::ADI_STD3(CulyaMatrix B, double epsilon)
2.    {
3.      CulyaMatrix P, P1_2;
4.      CulyaMatrix parameters;
5.      CulyaMatrix A;
6.      CulyaMatrix Previous;
7.      A=(*this).dup();
8.      TSpectrum spectrum;
9.      long int* index=NULL;
10.     long int cycles=20;
11.     long int parameters_number;
12.     parameters_number=A.shifts2(spectrum, parameters, index, cycles);
13.     A.del_dev();
14.     P=P.zeros_card((*this).rows, (*this).cols, (*this).type1, (*this).type2);
15.     for (long int i=0; i<10000; i++)
16.     {
```

```
17.       if( (i+cycles+1)>rows)
18.     {
19.       break;
20.       }
21.     if(i>=parameters_number)
22.     {
23.       parameters_number=A.shifts2(spectrum, parameters, index, cycles);
24.      }
25.      CulyaMatrix L, L2;
26.      L2=(*this).dup();
27.      L2.diag_add(parameters, i, 0);
28.      CulyaMatrix A_transpose;
29.      A_transpose=(*this)++;
30.      A_transpose.diag_sub(parameters, i, 0);
31.      CulyaMatrix product;
32.      product=P*A_transpose;
33.      P.del_dev();
34.      CulyaMatrix Pr;
35.      Pr=B.dup();
36.      Pr.odd();
37.      P=Pr-product;
38.      Pr.del_dev();
39.      product.del_dev();
40.      product=L2/P;
41.      P.del_dev();
42.      CulyaMatrix product2;
43.      product2=product++;
44.      product.del_dev();
45.      P=product2*A_transpose;
46.      A_transpose.del_dev();
47.      product2.del_dev();
48.      Pr=B.dup();
49.      Pr.odd();
50.      product=Pr-P;
51.      Pr.del_dev();
52.      P.del_dev();
53.      P=L2/product;
54.      L2.del_dev();
55.      product.del_dev();
56.      if( i==parameters_number-1 )
57.      {
58.        CulyaMatrix difference;
59.        difference=P-Previous;
60.        Previous.del_dev();
61.        double max;
62.        max=difference.max();
63.        difference.del_dev();
64.        if(max<epsilon) break;
65.      }
66.      if(i==parameters_number-2)
67.      {
68.        Previous=P.dup();
69.      }
70.    }
71.    spectrum.eig_C.del_dev();
72.    spectrum.eig_I.del_dev();
73.    spectrum.eig_R.del_dev();
74.    parameters.del_dev();
75.    cudaFree(index);
76.    (*this).del_dev();
77.    (*this)=P.dup();
78.  }
```

A.4. The controllability and observability gramians computation by using Sign Function.

```
1.   void CulyaMatrix::std_sign_f2(CulyaMatrix B, double epsilon)
2.   {
3.     CulyaMatrix transpose;
4.     transpose=(*this)++;
5.     (*this).del_dev();
6.     (*this)=transpose.dup();
7.     transpose.del_dev();
8.     for (int counter=0; counter<1000; counter++)
9.     {
10.      CulyaMatrix A_inv;
11.    A_inv=(*this)^-1;
12.    CulyaMatrix P;
13.    P=B*A_inv;
14.    CulyaMatrix A_inv_trans;
15.    A_inv_trans=A_inv++;
16.    CulyaMatrix L;
17.    L=A_inv_trans*P;
18.    P.del_dev();
19.    A_inv_trans.del_dev();
20.    P=B+L;
21.    CulyaMatrix difference;
22.    difference=B-L;
23.    double max;
24.    max=difference.max();
25.    difference.del_dev();
26.    if(max<epsilon)
27.    {
28.     L.del_dev();
29.     B.del_dev();
30.     B=P*0.5;
31.     P.del_dev();
32.     P=(*this)+A_inv;
33.     A_inv.del_dev();
34.     (*this).del_dev();
35.     (*this)=P*0.5;
36.     P.del_dev();
37.     break;
38.   }
39.   L.del_dev();
40.   B.del_dev();
41.   B=P*0.5;
42.   P.del_dev();
43.   P=(*this)+A_inv;
44.   A_inv.del_dev();
45.   (*this).del_dev();
46.   (*this)=P*0.5;
47.   P.del_dev();
48.    }
49.    CulyaMatrix A_inv;
50.    A_inv=(*this)^-1;
51.    (*this).del_dev();
52.    CulyaMatrix P;
53.    P=B*A_inv;
54.    CulyaMatrix A_inv_trans;
55.    A_inv_trans=A_inv++;
56.    A_inv.del_dev();
57.    CulyaMatrix L;
58.    L=A_inv_trans*P;
59.    A_inv_trans.del_dev();
60.    P.del_dev();
61.    (*this)=L*0.5;
62.    L.del_dev();
63.  }
```