Archiwum Instytutu Techniki Cieplnej Vol. **1** nr 1(2016) www.itc.polsl.pl/ArchiwumITC

Numerical analysis of the orthotropic sample for conductivity tests

Mieszko Tokarski[1]

e – mail: tomiesio@gmail.com

Key words: MATLAB, Ansys, FEM, Neumann's Boundary Condition, Factorization

Abstract

This work describes the development of a program based on the Finite Element Method for the calculation of a temperature field in orthotropic sample with use of the Neumann's Boundary Condition. Such a program has been created for the purposes of the project carrying out in the Intitute of Thermal Technology in Gliwice, Poland. It is an important part of the fully – automated algorithm for determining the sample's thermal conductivity by fitting numerically obtained temperature field with its counterpart provided by the measurements. Because of the specific nature of the measurement process as well as the main algorithm itself, the developed program is characterized by high efficiency (comparable to Ansys), sufficient accuracy and preparation for cooperation with the mentioned before fully – automated algorithm. Most important features of the program are: module for geometry data import (data is provided by the Ansys), module for the results export, the two control text files for easy management by external procedures, logging and error reporting module.

[1]The a*uthor have created the following chapter during the work on the master thesis carrying out in the Institute of Thermal Technology in the Department of Energy and Environmental Engineering of the Silesian University of Technology under the supervision of dr ing. Arkadiusz Ryfa. The work has been realised within the project carrying out by the Intitute of Thermal Technology in Gliwice, Poland.*

Nomenclature

Greek symbols

ξ, η, μ	- natural coordinates
P	- density, kg/m^3
Δ	- increase
∂	- derivative
A	- thermal diffusivity, m^2/s

Latinsymbols

N	- shape function
C	- specific heat, J/kg K
k	- thermal conductivity, W/(m °C)
$\dot{q}$	- heat flux, W/m^2

Superscripts

E	- element number
N	- Gaussian Point number
T	- transposition
1D, 2D, 3D	- 1, 2 ,3 dimensions
S	- step number

Subscripts

i, j	- nodal number

Abbreviations

BC(s)	- Boundary Condition(s)
BDM	- Backward Difference Method
CDM	- Central Difference Method
CMM	- Consistent Mass Matrix
DBC	- Dirichlet's Boundary Condition
DLMM	- Diagonally Lumped Mass Matrix
FDM	- Forward Difference Method
FEM	- Finite Element Method
GP(s)	- Gaussian Point(s)
NBC	- Neumann's Boundary Condition
PCG	- Preconditioned Conjugate Gradient
TC	- Thermal Conductivity

Notation

vectors and matrices are set in **boldface**

1 Introduction.

1.1 Background and motivation.

Most of widely known and used contemporary methods of determination of thermal conductivities (TC) are, in most cases, very inaccurate, time – consuming, complicated or have destructive character. A precisely determined TC is crucial in many engineering issues, for instance: assessment of heat losses and gains, definition of allowable thermal working conditions of a component or the entire machine and evaluation of thermal strains and stresses. In the case of insulating materials, the TC has decisive impact on the material's quality. In general, in the literature are described many TC measuring methods with their numerous variants, but certainly one can distinguish several major ones:

- Guarded hot plate – a solid sample is placed between two plates of which one is heated while the other is cooled (or heated in lesser extent). After reaching steady – state, necessary measurements and calculations are performed [1].
- Hot wire – a heated wire is inserted into the sample and then temperature change is recorded. Density and heat capacity have to be known. At the end, plot of the wire's temperature change versus logarithm of the time is used to calculate the TC [1].
- Modified hot wire – the wire is supported on backing so it does not require sample's penetration [1].
- Laser flash diffusivity – sample's surface is heated with laser's pulse and infrared camera records the temperature field [1].
- Magnetic resonance imaging techniques – relation between temperature and magnetic field has been used in order to determine the thermal diffusivity [2].

Guarded hot plate method requires a lot of time until steady – state is reached. Hot wire method, in turn, because of sample's penetration, can be applied to fluids, foams and melted plastics [1] but not for solids. While the Laser flash method (see also Parker flash method [3]) is fast, reliable but requires preparation of the samples that are damaged during the measurements due to high temperature. It is important to notice that most of them allow for determining TC in isotropic materials only. Hence, need of developing more reliable as well as more accurate methods arises, particularly in orthotropic or anisotropic materials case. An example of one of the promising and innovative methods is described in [2]. In brief, it constitutes a development of the Parker flash method [3] and also consists of heating a sample with use of a laser flash (Figure 2) and recording (with Infrared Camera) the temperature distribution on a heated surface. In the next stage numerical analysis is applied comprising modeling this phenomena with use of dedicated algorithm and comparing temperature fields (obtained both by experiment and by computations). Figure 1 shows schematic block diagram of the described procedure. It should be stressed that, in order to properly fit experimental temperature field with its numerically obtained counterpart, TC must be controlled during the analysis. Briefly, TC has to be guessed and experimental results are used as a benchmark.

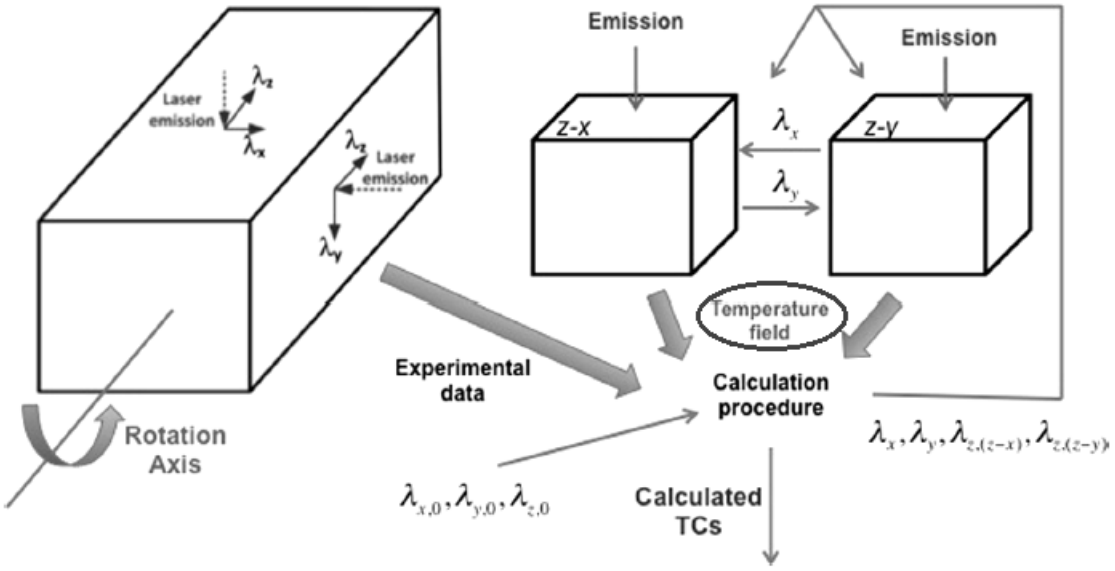


Figure 1: Schematic block diagram of the algorithm for determining the thermal conductivity in orthotropic material. The program's place in the procedure is marked with ellipse. The image has been taken from [2].

Usage of a commercial software like Ansys Mechanical, Ansys Fluent or MSC.Patran is, at least, troublesome. Mainly due to the specific requirements provided by the nature of the analysis. For instance, some difficulties occurred during modeling of heat transfer based on the Neumann's Boundary Condition (NBC) in Fluent. Most probably the Finite Volume Method (FVM) Fluent is using, is unsuitable for such phenomenon or there is a necessity for some specific settings of the analysis. For now, Fluent provides far higher temperatures than Ansys Transient Thermal (using Finite Element Method FEM). Such a situation confirms the fact that using the commercial software can be problematic due to its own specific features and requirements regarding the, for example, mesh or analysis setup. Furthermore, cooperation of this software with external procedures can provide a lot of trouble as well. It should be stressed that, for the purposes of the project it is necessary to allow for efficient cooperation between different programs belonging to the procedure described in the next paragraph. In this regard, developing of a self – made FEM code may be much better solution than use of the complicated commercial software. Besides that, the licenses of such software are expensive so in a situation when one has choose between commercial and free program with comparable performance and accuracy, the latter is better choice.

First of all, whole numerical procedure has to be fully automated. Experimental results are loaded into analysis setup file, then main program calls for a subprogram responsible for the computations which are carried out in order to get numerical temperature field. This subprogram prepares specific output files formatted in a desired way that next subprogram can compare specified temperatures and, if it is necessary, change the TCs and repeat whole procedure until convergence is reached.

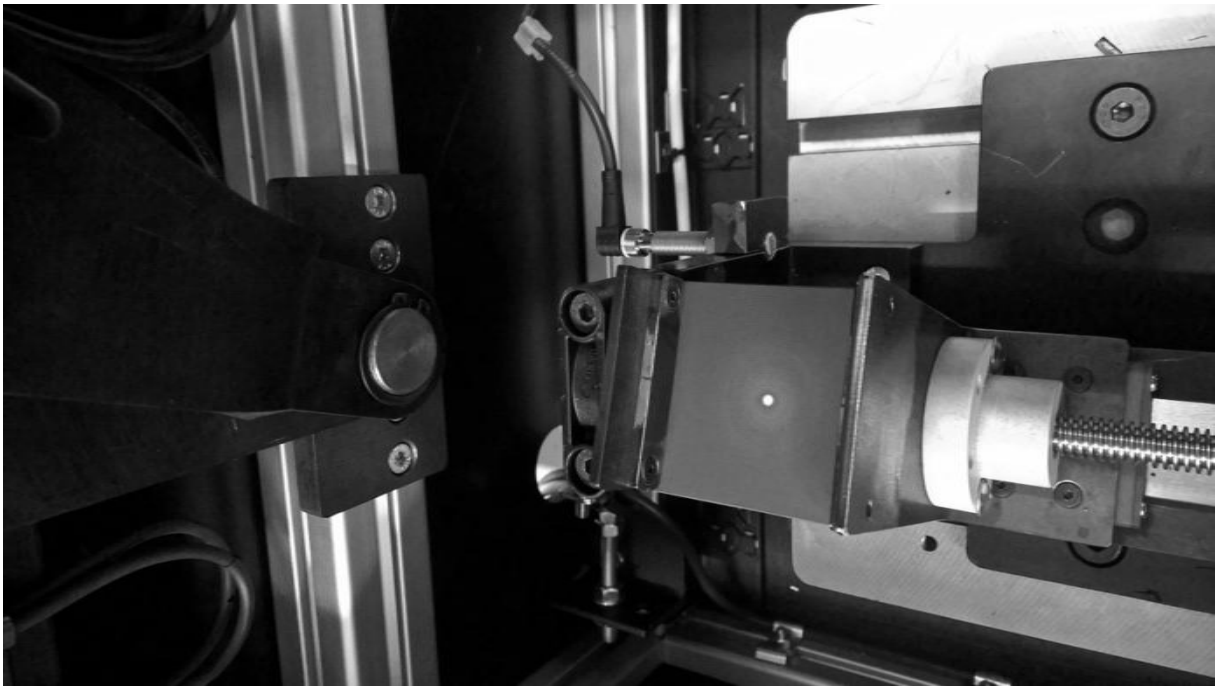


Figure 2: A sample holder and dot of the laser beam.

The subject of this work is to create a fully automated program using the Finite Element Method to solving heat transfer problems with use of the NBC. Such program will be used in a procedure of determination of the TCs within the project described in [2]. Besides that, it will have several important advantages over the Ansys. At first, in contrast to the complex Ansys and its highly developed interface, managing this program will be much simpler due to ordinary text files which are easy to create and edit with use of user defined procedures written, for instance, in MATLAB or in FORTRAN. Secondly, results provided by the program can be easily formatted in a required manner so their further processing will not cause any additional problems – it is easy to achieve with use of user defined procedures designed directly for this task and for this particular kind of the program. Additionally it is always better when there is possibility of customization both of cooperating programs than only one of them like it would have been if the Ansys was used instead of program developed as the subject of this work.

First task that had to be done, was to choose the programming environment which would allow relatively quick, easy and efficient developing of desired program. The choice fell on the MATLAB [4] because of several important factors:

- a wide range of efficient and already implemented functions for managing data,
- easy developing own procedures on the basis of the MATLAB's functions,
- efficient computing of large matrices,
- easy and user – friendly debugging module,
- 'Profiler' – tool facilitating the optimization of the code,
- File Exchange – free service for sharing files between MATLAB's users [5].

The next stage was proper application of the Finite Element Method (briefly described in Chapter 2.1). To be sure, the FEM written in MATLAB works properly, several test cases have been solved. Each of them had its own contribution into final result being finished program for the purposes of the project. It was a long journey through MATLAB programming environment, FEM's features and linear algebra needed to find a way of efficient and accurate solving systems of thousands equations.

As it was mentioned before, the program is for the purposes of the project realized at the Institute of Thermal Technology in Gliwice, Poland. A core problem is not a development of such program, but achievement of desired accuracy and efficiency. It is problematic due to the number of phenomena accompanying the process of laser's radiation. These phenomena are:

- radiation of the warmed sample' surface due to the laser beam,
- conduction inside the sample's material (TC is unknown),
- carbonization of the sample due to high temperatures.

Convection's impact is negligible. Radiation heat flow depends on a temperature to the fourth power. That temperature rises to about several hundred Celsius degrees in less than 0.2 s. Thermal conductivities are unknowns whereby they are additional complicating factor. Moreover, due to high temperatures local carbonization of the sample occurs thus chemical composition changes so material properties do as well. Besides that, sample's density, TC and specific heat, all depend on the temperature in real case.

Another factor complicating the task is short time interval. Mentioned above phenomena are very dynamic. Whole measurement process consists of three main stages:

- heating with use of the laser flash – lasts for about 0.2 second,
- removing the laser and replacing it with the Infrared (IR) camera in order to keep precisely the same angle as during previous stage – lasts for about 0.3 s,
- temperature field recording – lasts for about two second, time steps depend on recording times of IR camera (frames per second to be exact) and on the speed of saving those temperature fields on a hard drive.

At this stage, the program for numerical modeling of the temperature field is under the validation. In addition to that, importing required data, file for managing whole program, monitoring of the executed procedures (logs) as well as extracting results in desired form are already implemented. However, loss of the heat due to radiation and temperature – dependent material properties still require further investigation on their impact on the solution.

2 Governing equations.

In the FEM transient heat transfer equation [6] is solved numerically. For orthotropic bodies and constant TC, specific heat and density such equation takes the form

$$\nabla \mathbf{k} \nabla T + \dot{q}_v = \rho c \frac{DT}{Dt} \tag{1}$$

T is the temperature, **k** is thermal conductivity represented by the following vector

$$\mathbf{k} = [k_x \quad k_y \quad k_z] \tag{2}$$

$\dot{q}_v$is internal heat source (in this work $\dot{q}_v = 0$), ρ and cstand for the specific heat and density respectively.

2.1 The Finite Element Method.

Today solving complex mathematical problems of physics and engineering is possible using numerical methods only. Amongst these methods certainly worthy of distinction is the Finite Element Method – a powerful and widely used tool in the field of mechanics and heat transfer. To get more information about FEM, please refer to [7], [8], [9] and[11]. Figure 3 presents main idea of the FEM used in this work. First, main domain – a cylinder as an example of a solid that can be easily presented and simultaneously has irregular shape – is divided into sub domains (finite elements – that is the first approximation of the domain). Then it is assumed that all finite elements are hexahedrons (parent elements – this is second approximation of the domain). For each parent element Jacobian together with its determinants is computed as well as stiffness matrix of the element $\boldsymbol{K_e}$. Simultaneously mass matrix of the element $\boldsymbol{M_e}$ is determined. Next, these matrices are assembled into global $\boldsymbol{K}$ and $\boldsymbol{M}$ respectively. Right Hand Side (RHS) vector $\boldsymbol{F}$ stores coefficients of the NBC determined with GPs, nodal coordinates, shape functions for a 2D case and the same mesh. In the next stage, time step and initial temperature are applied and equation BDM [(Backward Difference Method) (16)] is solved. Each subsequent iteration has its own initial temperature from the previous time step.

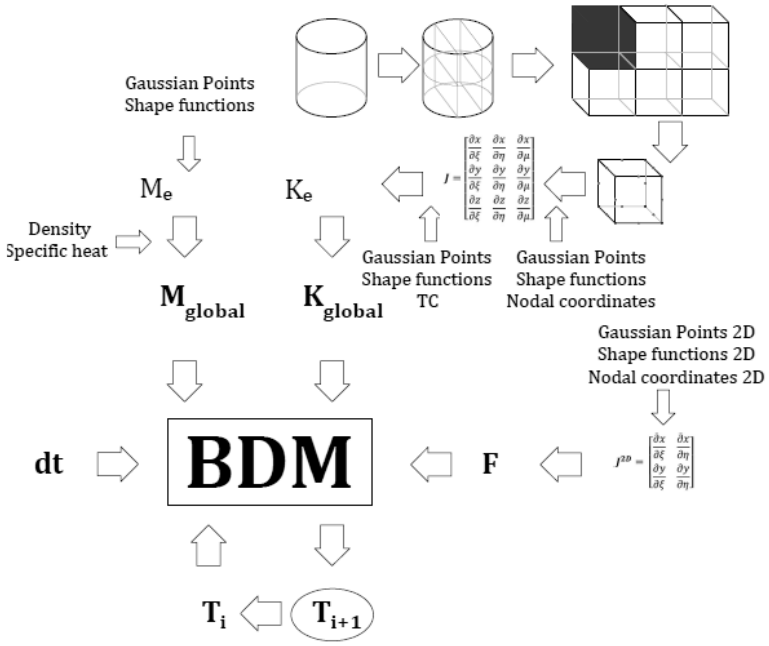


Figure 3: Schematic block diagram showing main idea of the FEM used in this work.

2.2 Shape functions.

Shape functions (SF) are polynomials and are used for interpolating continuous filed quantity as well as to define shape of the parent element. In this work hexahedral 20-node elements have been used, hence it follows the order of such functions together with their form. Such functions belong to, so called, serendipity shape functions family [9]. Shape functions are more precisely described in [7,9,10]. Figure 4 shows adopted manner of numbering of the parent element's nodes.

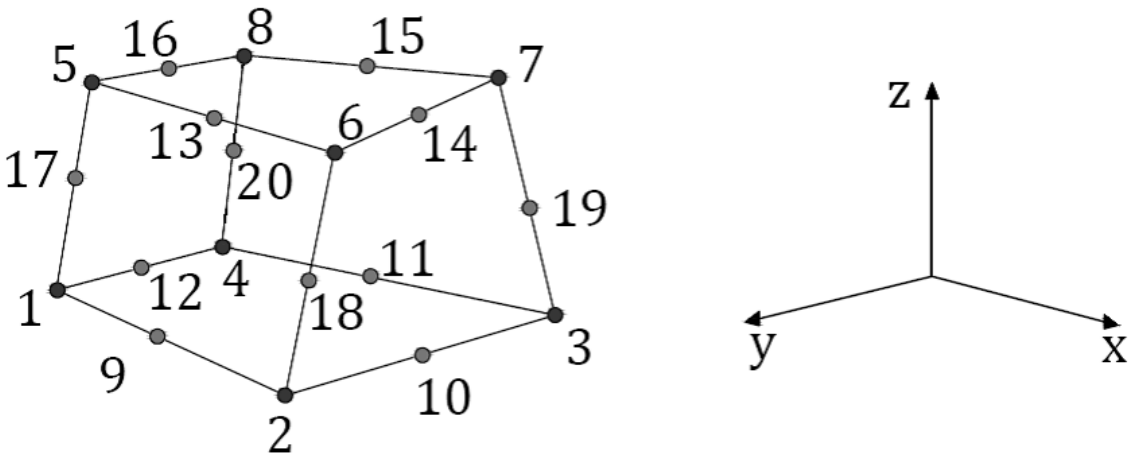


Figure 4: Hexahedral element in global coordinate system with its node numbering manner.

Nodal numbers correspond to the proper shape functions. The choice of 20 – node element (and those functions) is not accidental because of two important factors. Namely, shape functions of the second order are sufficient enough to model temperature distribution precisely, moreover use of serendipity elements allows to remove interior nodes of all hexahedron's walls as well as one right in the middle. As a result significant reduction of degrees of freedom occurs what leads to smaller computational effort – instead of 27 nodes there are only 20 in a single element. In addition to that, such reduction has no serious impact on accuracy [12].

2.3 Gaussian quadrature – integral approximation.

In numerical methods integrals are approximated instead of directly solved. There are many methods to do such approximation, some of them are more efficient than others, for instance, Gaussian quadrature. The idea is to pick such set of points (let say Gaussian Points - GP) that after the insertion of them into approximating integral, an accurate solution is obtained.

Each GP has its own wage. The wages define shares of approximated domain. In 3 dimensions (3D) it is a volume, in 2 dimensions (2D) it is a share of surface whereas in 1D case it is a section. In 3D case GP's coordinates has three components (x, y, z or ξ, η, μ), hence volume around the GP consists of a combination of three wages γ_x, γ_y, γ_z.

Figure 5 shows 1D example of the Gaussian quadratures. Despite the fact that this work concerns 3D domain, one dimensional case is far more readable. Each GP (dots) approximates

one piece of the bar. Each corresponding wage describes length of a section that is approximated by a given GP.

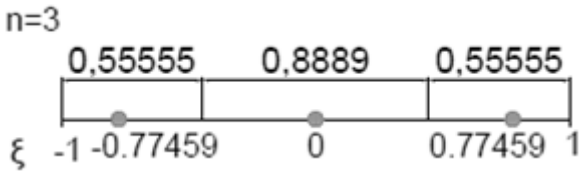


Figure 5: Gaussian quadrature of the third order in 1D case.

For more specific information about Gaussian quadrature, please, refer to [8,9,13].

2.4 The Jacobian – a link between global and natural coordinate systems.

The Jacobian [**J**] is a measure of the distortion of the given parent element defined in local coordinates in comparison to its counterpart in global coordinates [14]. Whereas determinant of the Jacobian, det[**J**], is numerically equal to the length of a section of its counterpart in 1D, surface in 2D case and equal to its volume in 3D case. The Jacobian is necessary to keep a link between sub - domains and their global counterparts.

Figure 6 shows simple 2D case while the exemplary element (gray) is transformed into quadrilateral parent element. Using the Jacobian allows to determine surface A of the given element

$$A(x,y) = \det[\boldsymbol{J}]\,\partial\xi\vartheta\eta \tag{3}$$

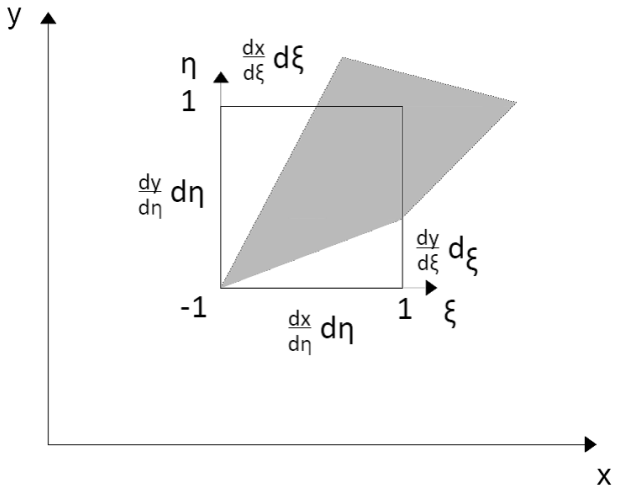


Figure 6: Finite element (gray) in global coordinate system and its counterpart in local.

If the element had been, for example, rectangle with the same length in x direction as its parent element, the two main (diagonal) Jacobian coefficients $\frac{\partial x}{\partial \xi}, \frac{\partial y}{\partial \eta}$would be non – zeros.

$$[\boldsymbol{J}] = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \mu} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} & \frac{\partial y}{\partial \mu} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} & \frac{\partial z}{\partial \mu} \end{bmatrix} \tag{4}$$

where[**J**] is the Jacobian matrix and $\frac{\partial x}{\partial \xi}, \frac{\partial y}{\partial \xi}, \frac{\partial z}{\partial \xi}, \frac{\partial x}{\partial \eta}, \frac{\partial y}{\partial \eta}, \frac{\partial z}{\partial \eta}, \frac{\partial x}{\partial \mu}, \frac{\partial y}{\partial \mu}, \frac{\partial z}{\partial \mu}$ are the distortions in a specific directions. Calculating such derivatives carries out in the following manner

$$\begin{bmatrix} \frac{\partial x}{\partial \xi} = \sum_{i=1}^{NoEN} \frac{\partial N_i^e}{\partial \xi} x_i & \frac{\partial x}{\partial \eta} = \sum_{i=1}^{NoEN} \frac{\partial N_i^e}{\partial \eta} x_i & \frac{\partial x}{\partial \mu} = \sum_{i=1}^{NoEN} \frac{\partial N_i^e}{\partial \mu} x_i \\ \frac{\partial y}{\partial \xi} = \sum_{i=1}^{NoEN} \frac{\partial N_i^e}{\partial \xi} y_i & \frac{\partial y}{\partial \eta} = \sum_{i=1}^{NoEN} \frac{\partial N_i^e}{\partial \eta} y_i & \frac{\partial y}{\partial \mu} = \sum_{i=1}^{NoEN} \frac{\partial N_i^e}{\partial \mu} y_i \\ \frac{\partial z}{\partial \xi} = \sum_{i=1}^{NoEN} \frac{\partial N_i^e}{\partial \xi} z_i & \frac{\partial z}{\partial \eta} = \sum_{i=1}^{NoEN} \frac{\partial N_i^e}{\partial \eta} z_i & \frac{\partial z}{\partial \mu} = \sum_{i=1}^{NoEN} \frac{\partial N_i^e}{\partial \mu} z_i \end{bmatrix} \tag{5}$$

where$\frac{\partial N_i^e}{\partial \xi}$ is an $e-th$ element's shape function's derivative, x_i, y_i, z_i are proper components of global coordinate of $i-th$ nodal point.

In the method adopted in this project Gaussian integration was used to estimate triple integral limiting parent element's domain. Determination of such matrix is quite simple and proceeds as follows

$$\begin{bmatrix} \frac{\partial N_1^n}{\partial \xi} & \frac{\partial N_1^n}{\partial \eta} & \frac{\partial N_1^n}{\partial \mu} \\ \vdots & \vdots & \vdots \\ \frac{\partial N_{20}^n}{\partial \xi} & \frac{\partial N_{20}^n}{\partial \eta} & \frac{\partial N_{20}^n}{\partial \mu} \end{bmatrix}^T \times \begin{bmatrix} x_1^e & y_1^e & z_1^e \\ \vdots & \vdots & \vdots \\ x_{20}^e & y_{20}^e & z_{20}^e \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \mu} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} & \frac{\partial y}{\partial \mu} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} & \frac{\partial z}{\partial \mu} \end{bmatrix}^{n,e} \tag{6}$$

where first matrix contains set of all nodes' derivatives calculated with $n-th$ Gaussian point, middle one contains global coordinates of $e-th$ element and last one is the *component* Jacobian matrix. In order to determine $[\mathbf{J}]^e$ a simple summation of all components$[\boldsymbol{J}]^{n,e}$must be carried out hence

$$[\boldsymbol{J}]^e = \sum_{n=1}^{NoGP} [\boldsymbol{J}]^{n,e} = \sum_{n=1}^{NoGP} \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \mu} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} & \frac{\partial y}{\partial \mu} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} & \frac{\partial z}{\partial \mu} \end{bmatrix}^{n,e} \tag{7}$$

where $NoGP$ is number of the Gaussian points (in this work $NoGP = 27$).

Determinant of the Jacobian matrix of the element e and $n - th$ GP is calculated as follows

$$\det[\boldsymbol{J}]^{e} = \sum_{n=1}^{NoGP} \det[\boldsymbol{J}]^{n,e} \tag{8}$$

where $\det[\boldsymbol{J}]^{n,e}$ is computed with use of the Sarrus' rule [15].

2.5 Element's Stiffness Matrix [K]^{e}.

Element's Stiffness Matrix **[K]^{e}** (or conductance matrix) contains a set of coefficients indicating mutual relationship between all nodes in the considering sub-domain. Such a matrix is diagonal having a size of Number of the Element's Nodes x Number of the Element's Nodes (*NoEN x NoEN*). In case of 20 – node hexahedron such a matrix is 20 x 20. Determination of the matrix **[K]^{e}** is as follows

$$\begin{aligned}[\boldsymbol{K}]^{\boldsymbol{e}} = \sum_{n=1}^{NoGP} \det[\boldsymbol{J}]^{n,e} \,(k_x \left[\frac{\partial N}{\partial x}\right]^{n,e} \left[\frac{\partial N}{\partial x}\right]^{n,e,T} \\ + k_y \left[\frac{\partial N}{\partial y}\right]^{n,e} \left[\frac{\partial N}{\partial y}\right]^{n,e,T} + k_z \left[\frac{\partial N}{\partial z}\right]^{n,e} \left[\frac{\partial N}{\partial z}\right]^{n,e,T})\end{aligned} \tag{9}$$

and

$$\left[\frac{\partial N}{\partial x}\right]^{n,e} = \begin{bmatrix} \frac{\partial N_1^{n,e}}{\partial x} \\ \vdots \\ \frac{\partial N_{20}^{n,e}}{\partial x} \end{bmatrix}, \quad \left[\frac{\partial N}{\partial y}\right]^{n,e} = \begin{bmatrix} \frac{\partial N_1^{n,e}}{\partial y} \\ \vdots \\ \frac{\partial N_{20}^{n,e}}{\partial y} \end{bmatrix}, \quad \left[\frac{\partial N}{\partial z}\right]^{n,e} = \begin{bmatrix} \frac{\partial N_1^{n,e}}{\partial z} \\ \vdots \\ \frac{\partial N_{20}^{n,e}}{\partial z} \end{bmatrix} \tag{10}$$

where $\left[\frac{\partial N}{\partial x}\right], \left[\frac{\partial N}{\partial y}\right], \left[\frac{\partial N}{\partial z}\right]$ are arrays of derivatives transformed to global coordinate system. In order to gather more specific information about such transformation, please refer to chapter 4 p. 154-157 of [8], chapter 2 p. 42-44 of [7] and chapter 2 p. 58-61 of [11]. Moreover, **[K]^{e}** is full.

2.6 Element's Mass Matrix [M]^{e}.

Element's Mass Matrix **[M]^{e}** (or capacitance matrix) contains approximated masses in nearest surroundings of all nodes of the element after multiplication by density. Otherwise sum of all values is equal to det**[J]**. In other words, each node is approximation of a part of the parent element it belongs to. In 3D case, such node represents some volume of a solid which has some density (mass) and specific heat. Such a matrix has the same dimensions as **[K]^{e}** and its computation is as follows [11]

$$[\boldsymbol{M}]^{\boldsymbol{e}} = \sum_{n=1}^{NoGP} \det[\boldsymbol{J}]^{n,e}\,[\boldsymbol{N}]^{n,e}[\boldsymbol{N}]^{n,e,T} \tag{11}$$

where

$$[\boldsymbol{N}]^{n,e} = \begin{bmatrix} N_1^e \\ \vdots \\ N_{20}^e \end{bmatrix}^n \tag{12}$$

It should be stressed that multiplication by density and specific heat occurs after global mass matrix assemblage. Moreover, similarly to **[K]**$^{\mathbf{e}}$, **[M]**$^{\mathbf{e}}$ is full.

2.7 Assembly of a global stiffness [K] and mass [M] matrices.

Assemblage of such matrices consists of a rewriting of all local**[K]**$^{\mathbf{e}}$ and **[M]**$^{\mathbf{e}}$ into one global matrix **[K]**$^{\mathbf{e}}$ and **[M]**$^{\mathbf{e}}$ respectively. Size of the **[K]** and **[M]** depends on a total number of nodes and is equal to $TNoN \times TNoN$. It would be convenient to discuss assemblage on the example. Element's stiffness matrix

$$K^e = \begin{bmatrix} K_{1,1} & \cdots & K_{1,NoEN} \\ \vdots & K_{i,j} & \vdots \\ K_{NoEN,1} & \cdots & K_{NoEN,NoEN} \end{bmatrix} \tag{13}$$

Provides us with pattern of local nodal indexing, hence

$$\left.\begin{aligned} ii &= [(1{:}\,1{:}\,NoEN)(1{:}\,1{:}\,NoEN)\cdots\,(1{:}\,1{:}\,NoEN)] \\ jj &= [(1,1,\ldots,NoEN)(2,2,\ldots,NoEN)\cdots\,(NoEN,NoEN,\ldots NoEN)] \end{aligned}\right\} \tag{14}$$

whereii and jj are local i – indices and j- indices respectively. For the first element (e = 1) global indices equals local ones, but for any other element do not. Because of matrix dimensions 20x20, global indices increase by 400 for each subsequent element.

At this point it suffices to state that, method adopted in this work is fast and effective. It should be noted that **[K]** and **[M]** are sparse, which means a lot of their elements are equal to zero. MATLAB's function *sparse()* allows to store only non – zero elements in the matrix. Such solution saves a lot of memory. Example of stiffness and mass matrices are shown in Figure 7. To get more specific information about stiffness and mass matrices assembly, please refer to [7] or [11].

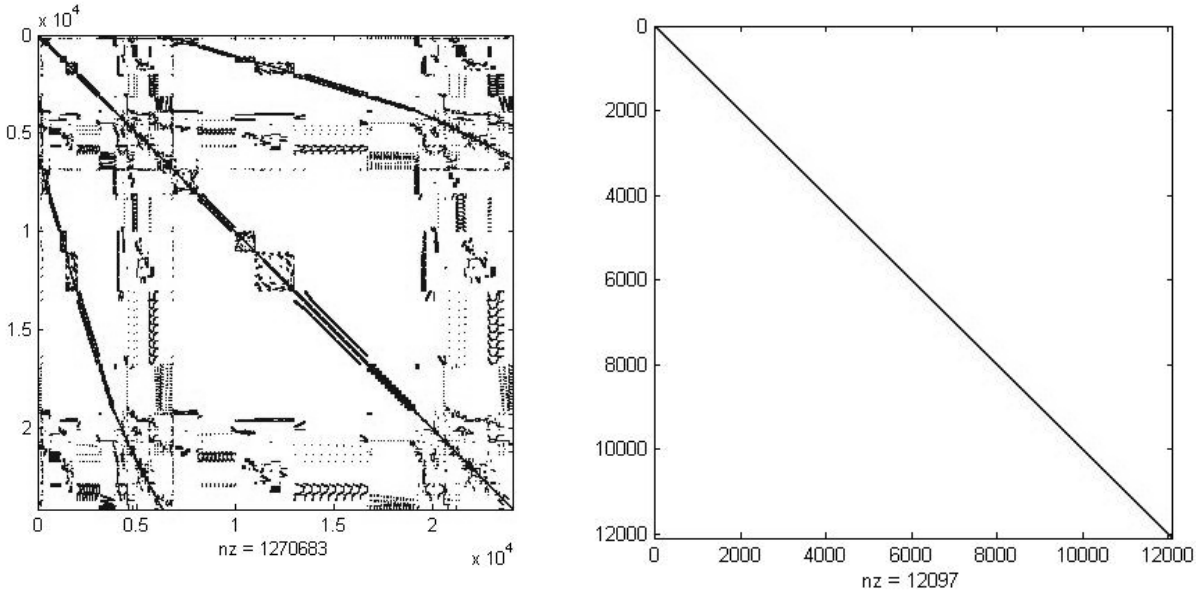


Figure 7: Stiffness matrix **[K]** and Consistent Mass Matrix **[M]** (left hand side) and Diagonally Lumped Mass Matrix (right hand side). Black dots denote non – zero values. Generated with MATLAB's *spy()* function.

It is important to state that, those matrices (left hand side of the Figure 7) are highly decomposed. In order to save computational effort one should strive for narrowing all off – diagonal elements and concentrate them along main diagonal by optimization of nodal numbering. Mass matrix with identical non – zero elements' distribution as well as the same dimensions as **[K]** is known in the literature as Consistent Mass Matrix (CMM). Besides CMM, in the literature known is also Diagonally Lumped Mass Matrix (DLMM) being approximated version of CMM. Such a matrix has significantly reduced number of elements what leads to faster computations but, due to additional approximation, lowers the accuracy. Generally DLMMs are used in cases of very large geometries with millions of nodes when computation time is more important than the accuracy [16].

Such approximation consists of a reduction of all off – diagonal elements to one lying in the main diagonal, row – wisely. However, there are some requirements that have to be met. For instance, in case of serendipity element, rows corresponding to their corner nodes result in negative masses after summation of all elements which belong to them. In such a case, conservation condition is not satisfied so use of different method of lumping may be necessary. In the literature one can distinguish three main ways of mass matrix lumping:

- summation of all off – diagonal elements into one lying on the main diagonal,
- scaling diagonal elements by proper factor – in this method det[**J**] is divided by sum of all diagonal elements and next is multiplied by each of them separately,
- Lobatto's method – is very similar to Gaussian quadrature.

Each of mentioned above methods, has its own pros and cons. In all of them lumping is applied to local (element's) mass matrices which are assembled later. In order to get more details, please refer to [16].

2.8 Neumann's Boundary Condition.

In the work at hand boundary condition of the second kind is used to model the laser flash. In case of a 3D geometry, laser flash (Heat Flux) can be prescribed to the surface only (see Figure 8).

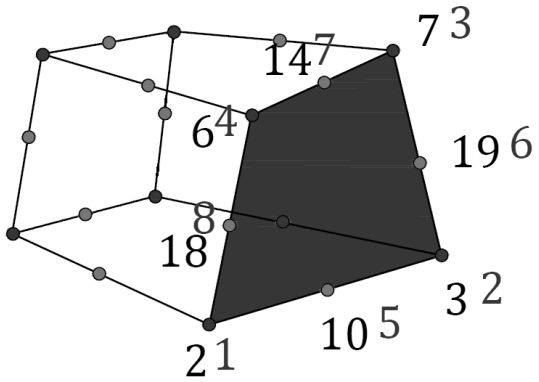


Figure 8: Example of applying HF and new 2D nodal numbering (grey).

As a result appears a necessity of determination of the Jacobian 2D as well as use of proper shape functions together with Gaussian Points. Determination of the $[\boldsymbol{J}]^{\boldsymbol{2D}}$ carries out in the same way as the $[\boldsymbol{J}]$ for 3D case. More information about 2D case can be found in [7], [8], [9] and [11].

Such surface has 8 nodes and, in case of 3rd order quadrature, 9 GPs. Heat flux coefficients stored in RHS vector **F** are determined as follows

$$\boldsymbol{F}_i^{2D} = \sum_{n=1}^{9} Ni^{2D}(n)\,\det|J|^{2\mathrm{D}}\,(n) \tag{15}$$

where $i = 1{:}\,8$ is nodal number, $n = 1{:}\,9$ is GP number and $\boldsymbol{F^{2D}}$ denotes temporary array with the BC coefficients. Next, all of these coefficients are multiplied by desired heat flux.For single element, there is 8 coefficients determined in this way that are (with use of the connectivity matrix) inserted into proper rows of the RHS array **F** (according to their global order).

2.9 Backward Difference Method.

BDM assumes that, nodal temperature in step s+1 depends on all other nodal temperatures from step s. In this work BDM has been chosen because of three main reasons:

- as an implicit scheme, is always convergent,
- there is no 'inversion' of **[M]** ([6], [11]),
- is widely used by commercial software.

First point refers to the time step Δt which has no impact on convergence of the solution [11]. The second one means that, in case of dense mesh, lumping of CMM will not be necessary to keep reasonable calculation time because there is no mass matrix 'inversion' in Eq.(16). Another advantage of this method is simplicity, because implementation of the BDM does not require much effort in comparison to Central Difference Method (CDM) or to Forward Difference Method (FDM) [11]. System of equations can be written as

$$\boldsymbol{T}^{s+1} = \frac{(\boldsymbol{M}\boldsymbol{T}^{s} + \Delta t\boldsymbol{F})}{(\boldsymbol{M} + \Delta t\boldsymbol{K})} \tag{16}$$

where $\boldsymbol{T}^{s+1}$ is vector of nodal temperatures in step s+1, $\boldsymbol{T}^{s}$ is vector of nodal temperatures in previous step s.

It is important to notice that in the denominator of Eq.(16) is [**M**] together with [**K**] which has to be 'inversed' with no exception. [**M**] in the nominator, in this particular scheme, does not have to be inversed. Eq.(16) is recomputed each time when time step Δt changes and can be rewritten

$$\boldsymbol{T}^{s+1} = (\boldsymbol{M} + \Delta t\boldsymbol{K})\backslash(\boldsymbol{M}\boldsymbol{T}^{\boldsymbol{s}} + \Delta t\boldsymbol{F}) \tag{17}$$

where \ is MATLAB's left division. Behind \ is hidden advanced equation solver using a wide range of numerical methods [17], but has one essential drawback. Namely, solving many systems of equations like $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{A}\boldsymbol{y} = \boldsymbol{c}$ requires factorization each time, when \ command is executed. There is no possibility of storing factorized matrix in order to reuse it in another system of equations. Matrix **A** inversion could be carried out once and then reused but such solution should be excluded at the very beginning because of three main reasons:

- inversion process is highly inaccurate – a lot of elementary operations are burdened with numerical errors,
- such process requires a lot of time,
- in case of sparse matrices (huge systems of equations), so called, fill – in phenomenon occurs, what leads directly to full matrix instead of sparse and to a lot of memory requested in order to store such matrix.

There are two ways of solving huge systems of equations of type $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{A}\boldsymbol{y} = \boldsymbol{c}$:

- direct methods - fast and accurate but requiring a lot of memory,
    - factorization (Cholesky, LU, LDL^T, QR [20], Gaussian Elimination),
- iterative methods - not so fast and not so accurate but saving a lot of memory.
    - Preconditioned Conjugate Gradient (PCG),
    - Jacobi Conjugate Gradient (JCG),
    - Cholesky Conjugate Gradient (CCG) [18].

Geometry and mesh, in this work and in the project [2] as well, are rather small and simple so use of factorization is possible. Because of the decomposition of matrices [**K**] and [**M**] Gaussian Elimination is inefficient, but works goods for diagonal matrices with narrowly

distributed off – diagonal entries. Besides that, PCG has been tested also (because of the best performance described in [18]) and it turned out it is comparable to factorization in MATLAB 2011a in contrast to 2009b.

Through Mathworks' File Exchange [5],Timothy A. Davis of the University of Florida has released his own factorization algorithm [19] in 2009 that has been permanently implemented to MATLAB 2014b as toolbox. This algorithm has been used, in this work, to solve Eq.(17).

2.10 Errors.

All errors presented in this work are calculated as follows

$$Err_i = \sqrt{\left(\frac{\sqrt{T_{i,DLMM}^2} - \sqrt{T_{i,CMM}^2}}{\sqrt{T_{i,CMM}^2}} \cdot 100\% \right)^2} \tag{18}$$

where T is nodal temperature, i denotes node number, $DLMM$ and CMM are Diagonally Lumped Mass Matrix and Consistent Mass Matrix respectively.

In the case of mixed BC described in Chapter 3.1.2 (steady – state), nodal temperatures denoted with use of DLMM and CMM subscripts in the Eq.(18) are replaced by nodal temperatures obtained with 2nd and 3rd order quadratures respectively using Eq.(19).

3 Numerical study – benchmark.

Before the program has been written, two benchmarks had been solved. First one in 2D, second one in 3D to make sure the FEM works properly. It was very important to implement new features step by step because of the complexity of the Finite Element Method so finding potential mistake could be extremely difficult. At first, learn about integration between local and global systems of coordinates was necessary (Gaussian quadrature, Jacobian). After that proper assemblage of global stiffness matrix had to be done.

3.1 2D benchmark.

Pattern example has been generated with PATRAN and consisted of two cases of steady – state heat transfer:

- pure Dirichlet's Boundary Condition (DBC),
- mix of Dirichlet's and Neumann's BCs.

This benchmark was carried out in order to learn how to apply BCs properly. Applying of these BCs is discussed in chapters 3.1.1 and 3.1.2 using the specific examples for the educational purposes. Also implementation of the BCs is described more specifically when discussing the two following examples, mainly due to greater readability.

The geometry from PATRAN (Figure 9) was a simple plate composed of 27 elements and 106 nodes of length 0.5 m and width 0.2 m. All elements were 8 – nodal hexahedrons and were equal to each other. Nodal coordinates as well as connectivity matrix were exported from

PATRAN to a text file. It was, and it meant to be, a simple test and its purpose was to validate the developed program.

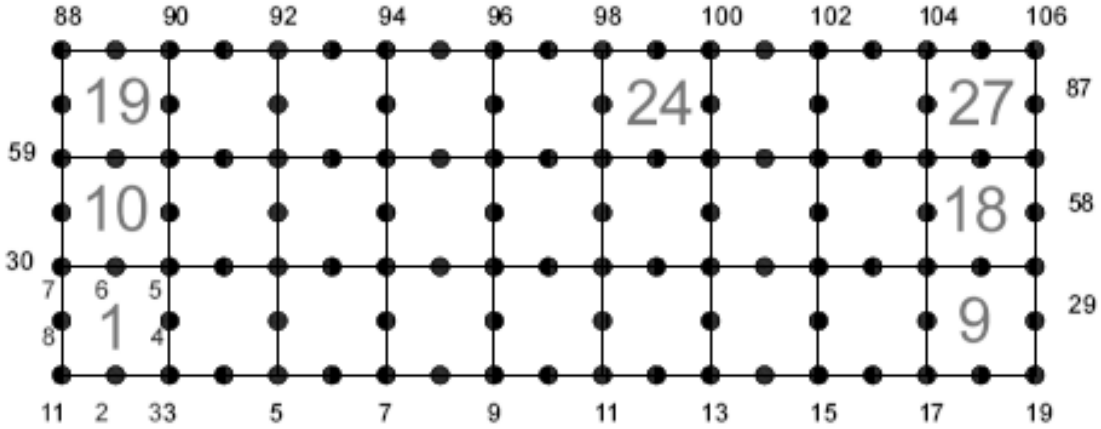


Figure 9: The geometry generated with PATRAN.

PATRAN's nodal numbering were adopted so all nodes correspond to each other. Another thing worth of mention is that, to approximate double integral Gaussian quadrature of the second order was used. As shown later research, third order quadrature met the requirements concerning the accuracy. To solve steady – state problem, the following equation is used

$$\boldsymbol{T} = \boldsymbol{K} \backslash \boldsymbol{F} \tag{19}$$

Results in the following subsections have been obtained for thermal conductivity $k = 25 \frac{W}{m\ ^\circ C}$.

3.1.1 Dirichet's Boundary Condition.

Boundary condition of the first kind, also known as Dirichlet's BC, has been applied in the way described in [8] on pages 103 – 104 with use of, so called, penalty method.

To the left hand side of the elements 1, 10 and 19 (see Figure 9) 50˚C has been prescribed, to the right hand side of the elements 9, 18 and 27, 150˚C. Figure 10 proves temperature distribution in both cases is the same and isotherms are arranged into symmetrical, vertical strips of equal width. Gaussian order of the second order was used.

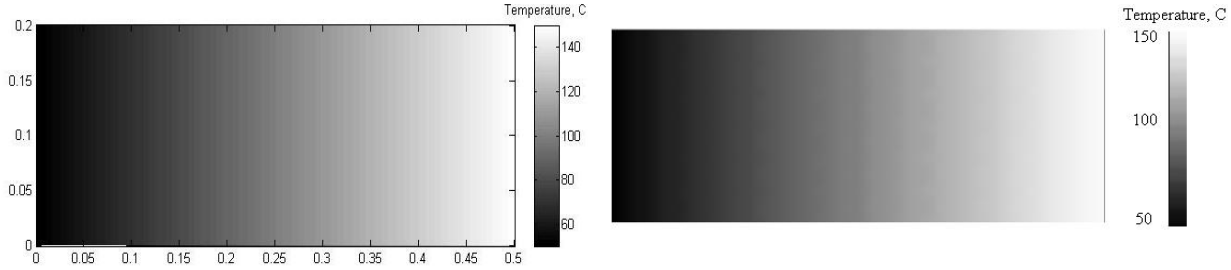


Figure 10: Temperature distribution from:
MATLAB (left hand side), PATRAN (right hand side).

It should be stressed, that presenting the results provided by different programs, in a similar manner, is problematic. For instance, in MATLAB it is easy to generate such figure (Figure 10) in grayscale in contrast to PATRAN, where it is much more complicated. During the analysis has been discovered that, in this particular case of pure DBC, results do not depend on values of coefficients in a stiffness matrix as well as on Gaussian quadrature's order. But they do depend on sign – negative or positive and to be exact, on distribution of negative and positive coefficients in [**K**]. Nevertheless, investigation of the nodal temperatures (chosen ones has been gathered in Table 1) has confirmed the fact that, the results are correct.

Table 1: Comparison of chosen nodal temperatures (second order quadrature).

Self-made code in MatLAB		Patran
Node:	Nodal temperature, ˚C:	
106	149.9999	150.0000
95	88.8889	88.8888
80	72.2222	72.2221
56	127.7777	127.7780
14	122.2221	122.2220
1	50.0000	50.0000

As one can notice, the results are nearly the same with some negligible differences after third decimal place. Obviously, the code, in this particular case, works properly.

3.1.2 Mixed Dirichlet's and Neumann's Boundary Condition.

Next thing that had to be done, was implementation of the BC of the second kind (imitating the heat flux), crucial to the next stages of this work. In this case DBC remained unchanged and NBC has been added. Normal heat flux ($\dot{q}_s = 40000\ \frac{W}{m^2}$) has been prescribed to northern edge of the 24 – th element (see Figure 9).

From the point of view of the heat flux (normal to the boundary), the edge to which it is assigned is a single line described by nodes 98, 99 and 100. Single line means one dimensional case. For that reason an approximation of the edge in local coordinates system has been done. That requires different shape functions, new Jacobian and new set of the Gaussian points. Further implementation of the NBC (RHS coefficients) is carrying out according to Eq.(15).

Temperature fields obtained from MATLAB and PATRAN are presented in Figure 11. On the left hand side of the plate DBC (50 °C) is prescribed that keeps applied there temperature (cools the plate), second DBC (on the right side) holds there 150 °C. Applied heat flux to the northern edge of the 24 – th element is sufficient to locally heat the plate to temperature over 200 °C. The isotherms are unsymmetrical due to two different (but symmetrically oriented) DBCs.

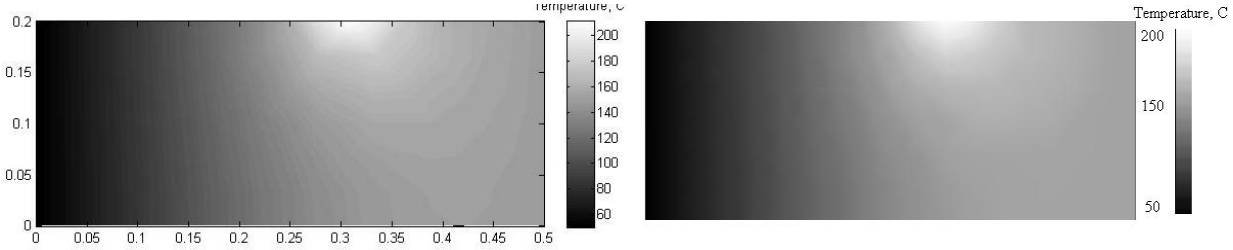


Figure 11: Temperature distribution obtained from: MATLAB (left hand side) and PATRAN (right hand side). Mixed BC.

Table 2 shows chosen nodal temperatures of the considering case obtained with 2nd and 3rd Gaussian quadrature's order. Nodes to which the NBC is applied are set in bold.

Table 2. Chosen nodal temperatures obtained with MATLAB and PATRAN.

Self – made code in MATLAB			PATRAN
Quadrature's order	2	3	3
Node:	Nodal temperature, ˚C:		
106	149.9999	150.0000	150.0000
100	**200.8242**	**200.7404**	**200.7380**
99	**212.5521**	**211.8979**	**211.8959**
98	**192.5137**	**192.4287**	**192.4279**
95	127.5782	127.6529	127.6530
80	92.6374	92.6137	92.6132
56	156.7796	156.7883	156.7879
14	150.0737	150.0577	150.0570
1	50.0000	50.0001	50.0000

It is easy to notice that, second order quadrature is not accurate enough in case of mixed BC or pure NBC, but in case of pure DBC is sufficient. Moreover, the closer to the nodes that NBC is prescribed to (98, 99, 100), the higher differences between the nodal temperatures are. All errors were calculated with use of Eq.(18) and take the maximum values directly at NBC – nodes and are negligible after exceeding a certain distance (from the NBC), see Figure 12.

It is clear that error's distribution concentrates in region of NBC and spreads in all available directions. Despite the fact that, applied DBCs are at varying distances from the NBC, error distribution is almost perfectly symmetrical. At the nodes where DBC is prescribed, those differences are negligible (see Table 2, nodes 1 and 106).

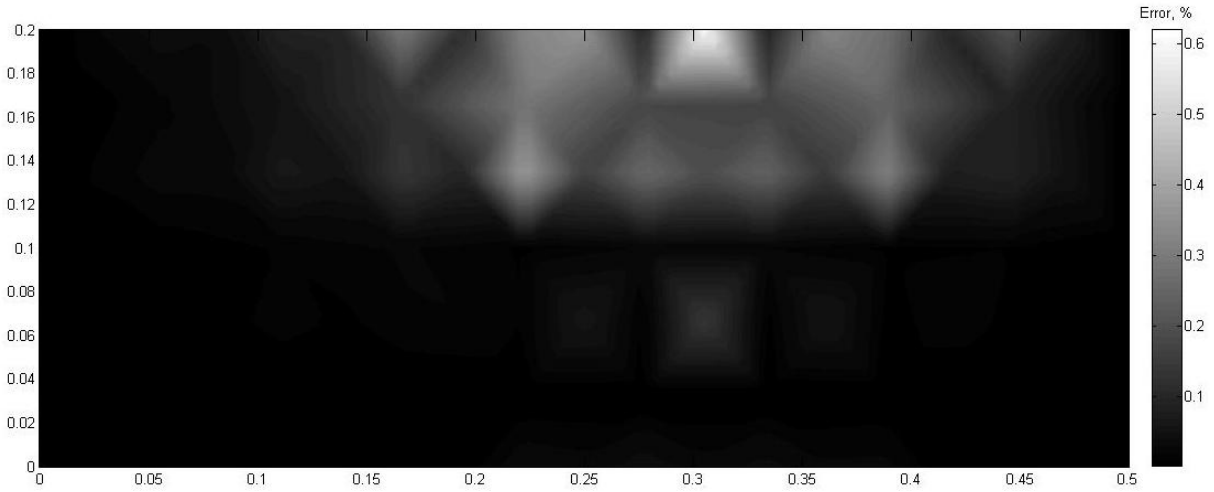


Figure 12: Graphical interpretation of the errors' distribution due to applying the NBC with use of the 2nd order Gaussian quadrature in comparison to the 3rd order quadrature.

Obviously, 2nd order quadrature does not provide satisfying accuracy in contrast to the 3rd order quadrature that simultaneously involves slightly higher computational effort (instead of 4 GP points there are 9 for 2D case). Because of the accuracy, 3rd order quadrature has been permanently implemented in the program.

3.2 3D benchmark.

At this phase, extending 2D FEM to 3D was necessary. Main difficulty was finding proper formulas. In the literature 2D cases are described in detail, but in most cases when it was coming to 3D, authors just limited themselves to a perfunctory statement, that all formulas are similar to 2D. As it turned out later, they were right. Nevertheless, stiffness matrix assemblage provided a lot of difficulties. At this phase, again, development of the code proceeded step by step in order to evade mistakes that could be hard to find.

Analysis setup and material properties:

- Thermal conductivity $k = 25 \frac{W}{m\,°C}$.
- Density $\rho = 1091\ \frac{kg}{m^3}$.
- Specific heat $c = 900 \frac{J}{kg\,°C}$.
- Time step $\Delta t = 20\ s$.
- Time of analysis (total time)$tt = 2500\ s$.
- Heat flux $q_s = 40000\ \frac{W}{m^2}$.

Applying the BCs has been carried out in the same way as for 2D benchmark in the MATLAB. The geometry has been created in PATRAN together with the BCs and imported to the MATLAB. A cuboid, presented in Figure 13, consists of 60 hexahedral elements and 406 nodes.

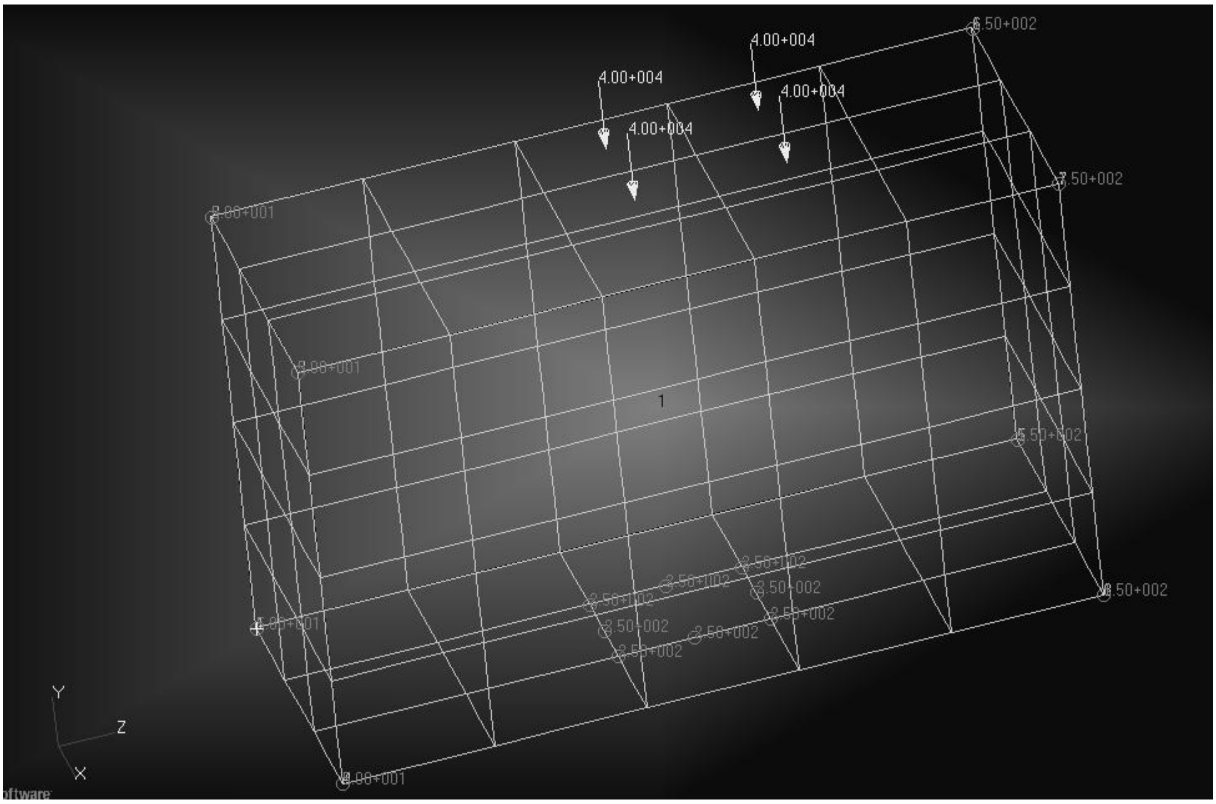


Figure 13: 3D geometry prepared with PATRAN and prescribed all boundary conditions. Heat flux were applied later, in the next analyzes.

3.2.1 Dirichlet's Boundary Condition, steady – state.

In this case DBC was ascribed at three regions:

- lower X – Y plane: 50˚C (first),
- upper X – Y plane: 150˚C (second),
- one element in X – Z plane: 250˚C (third).

Three different DBCs are prescribed in order to eliminate special case described in chapter 3.1.1concerning pure DBC in 2D, where values of coefficients in stiffness matrix did not have any impact on the solution. Third DBC prevents such situation. Figure 14 shows temperature distributions obtained from PATRAN and MATLAB. Unfortunately, rotating a 3D figure in MATLAB is limited so setting similar view to PATRAN's figure is not possible. Third DBC (250˚C) makes isotherms irregular and much more concentrated on the wall it is prescribed to, especially between the first DBC and the third.

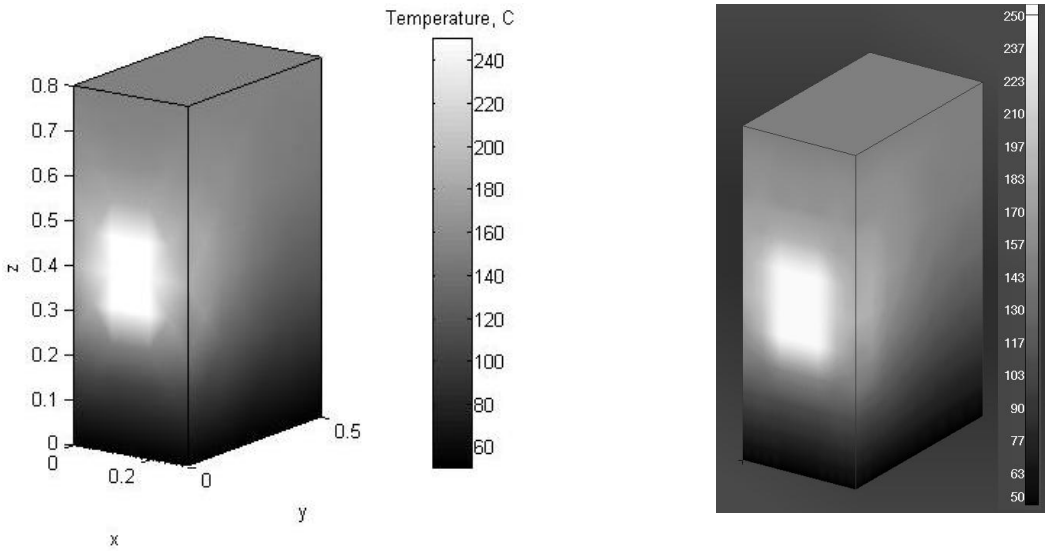


Figure 14: Temperature distribution obtained from: MATLAB (left hand side), PATRAN (right hand side). Steady – state, triple DBC.

Table 3 presents chosen nodal results from MATLAB and PATRAN. As one can see, 3rd order quadrature applied to MATLAB's code is sufficient and allows for getting accurate temperatures in comparison to PATRAN.

Table 3: Results from MATLAB (3rd order quadrature) and PATRAN.

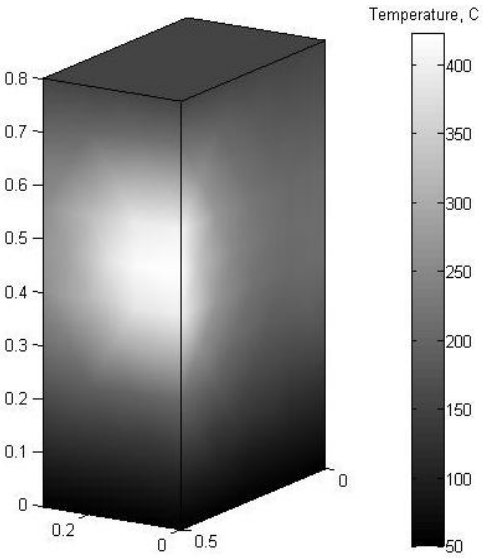
Self-made code in MATLAB		PATRAN
Node:	Nodal temperature, ˚C:	
1	49.9999	50.0000
50	50.0000	50.0000
100	92.9081	92.9082
150	166.0193	166.0200
200	168.7079	168.7079
250	138.8943	138.8939
264	135.7371	135.7370
300	159.2606	159.2610
350	149.0673	149.0670
400	150.0000	150.0000
406	150.0000	150.0000

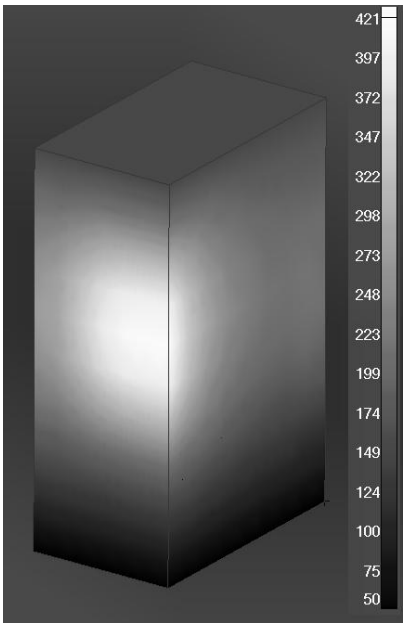
This benchmark proved that, the stiffness matrix **[K]**in 3D case has been assembled correctly. Otherwise, corresponding nodal temperatures would not be consistent. The differences between them are negligible.

3.2.2 Mixed Boundary Condition, steady - state.

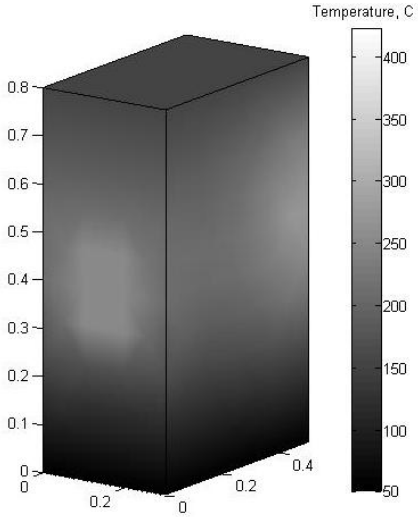
In this case, the correctness of applying mixed BC has been verified. To triple DBC described in chapter 3.2.1, NBC has been added for the following elements: 34, 35, 46, 47 (presented on Figure 13). Figure 15 presents graphical interpretation of the obtained temperature fields provided by MATLAB and PATRAN. Applied heat flux, is sufficient for increasing the maximum temperature to higher value than third (the highest) DBC.

a) b)





c) d)



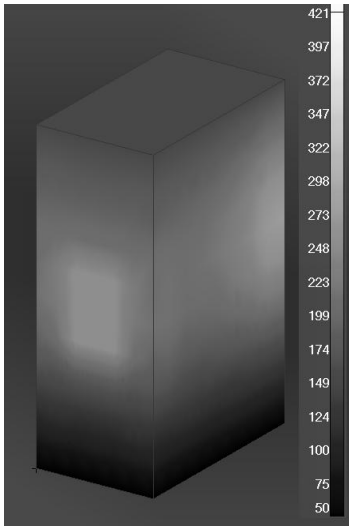


Figure 15: Temperature distribution with triple DBC and single NBC (mixed BC). a) NBC side, MATLAB, b) NBC side, PATRAN, c) 3rd DBC side, MATLAB, d) 3rd DBC side, PATRAN.

In Figure 15 c) and d), one can notice rectangular shape slightly brighter than its nearest surroundings – it is third DBC (250 °C) applied to the four chosen elements' surfaces. To the lower base (lower X – Y plane) first DBC is prescribed, what is confirmed by dark shade of gray. Similarly upper base (upper X – Y plane) to which second DBC is prescribed and which shade corresponds to 150 °C. Big bright stains in Figure 15 a) and b) present applied heat flux and are located near the same edge. Except for the number of isotherms, figures from the MATLAB look similarly to those from the PATRAN. Chosen nodal temperatures are presented in Table 4 and are nearly the same.

Table 4: Results from MATLAB (3rd order quadrature) and PATRAN. Mixed BC.

Self-made code in MATLAB		PATRAN
Node:	Nodal temperature, ˚C:	
1	49.9999	50.0000
50	50.0000	50.0000
100	125.7416	125.7409
150	192.7073	192.7070
200	206.0997	206.0989
250	266.3079	266.3060
264	269.3892	269.3880
300	187.2779	187.2780
350	188.7256	188.7250
400	149.9999	150.0000
406	149.9999	150.0000

This benchmark proved that 3rd order quadrature is sufficient to obtain results that are comparable to PATRAN's. In addition to that, stiffness matrix assembly (3D) has been carried out properly as well as applying of the NBC (2D – surface).

3.2.3 Mixed Boundary Condition, transient, CMM and DLMM.

The next step was an implementation of the BDM – transient heat transfer equation (Eq.(16)) in place of steady – state equation (Eq.(19)). Second and third DBCs (150˚C and 250˚C) have been removed. First DBC (50˚C) remained and simultaneously this temperature was an initial value also. NBC remained unchanged as well as density, TC and all others. The following figure (Figure 16) is a set of two figures from MATLAB and two from PATRAN. It presents temperature distribution at 20 and 2500 second of heating a cuboid with heat flux. The temperature increases gradually around the area to which NBC is applied. Because of the first DBC that holds 50˚C on the basis of the presented cuboid, upper part of the solid heats up to a much greater extent. Back side, however, still remains 'cold' but it is only a matter of time before the temperature become more uniform. It is noteworthy that steady – state has not been achieved.

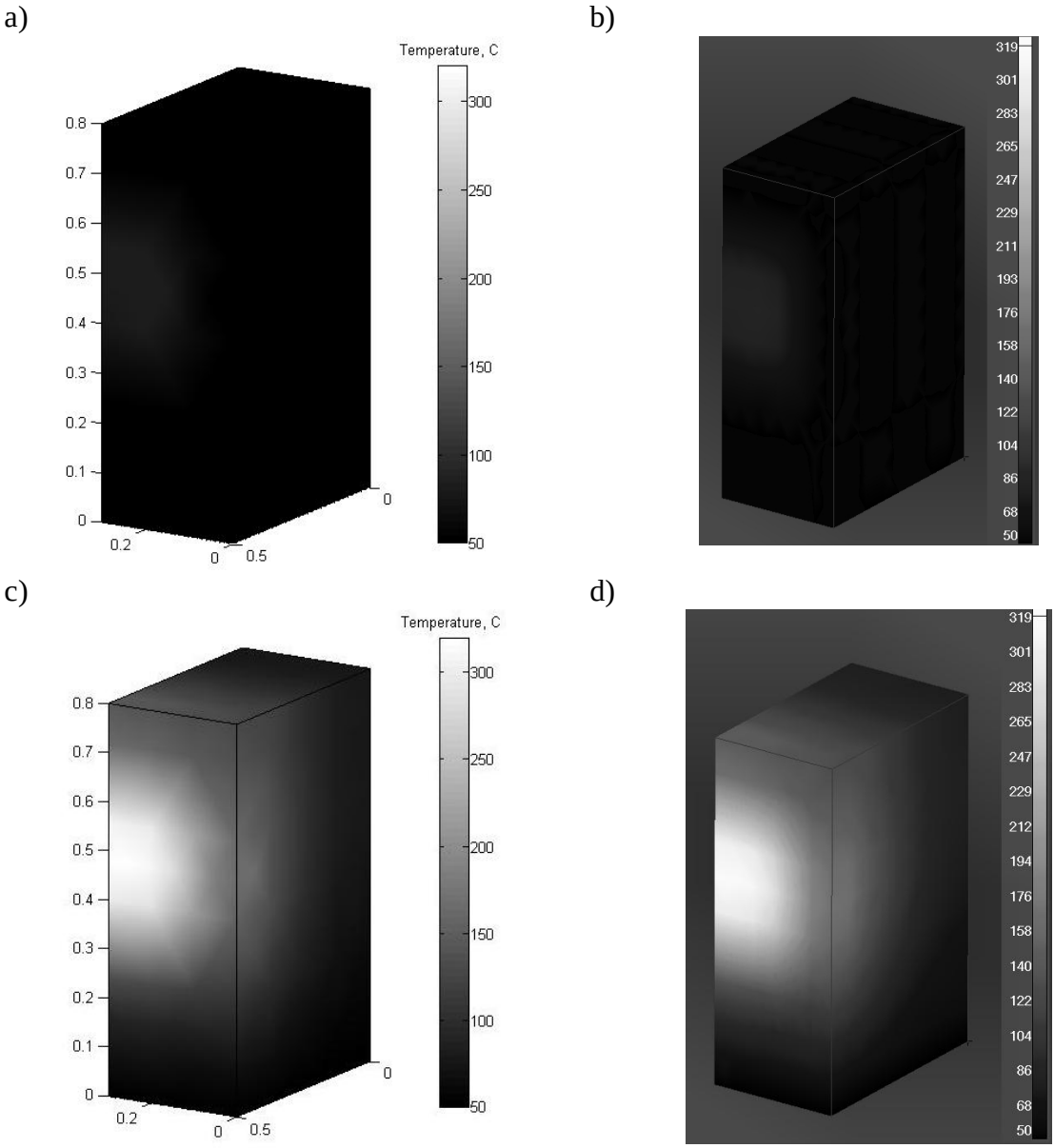


Figure 16:Temperature distribution from MATLAB (left hand side) and PATRAN (right hand side): a) and b) after 20 s; c) and d) after 2500 s.

As one can see, PATRAN's shade interpolation looks smoother. Nevertheless, the temperature distribution in both cases is similar. In

Table 5 are presented results from MATLAB and PATRAN. Setup of the analysis was the same in both cases. Node to which the NBC is applied is set in bold. Calculations have been carried out with CMM and with use of factorization algorithm written by Timothy Davis [20].

Table 5: Results from MATLAB and PATRAN. Single DBC and NBC. CMM.

	MATLAB			PATRAN		
Time, s:	20	1500	2500	20	1500	2500
Node:	Temperature, ˚C:			Temperature, ˚C:		
1	50.0000	50.0000	50.0000	50.0000	50.0000	50.0000
50	50.0000	50.0000	50.0000	50.0000	50.0000	50.0000
100	50.1069	61.9589	72.1512	50.1068	61.9588	72.1509
150	49.9937	57.41512	71.0562	49.9937	57.4151	71.0560
200	49.9918	63.5700	80.9149	49.9918	63.5698	80.9146
250	51.9898	133.0161	161.7480	51.9896	133.0149	161.7469
264	**82.2932**	**288.2891**	**319.2235**	**82.2923**	**288.2869**	**319.2210**
300	50.0164	63.5048	84.1807	50.0164	63.5046	84.1801
350	51.0058	103.2670	134.5854	51.0057	103.2669	134.5850
400	49.3207	95.0306	128.3418	49.3207	95.0303	128.3410
406	48.6141	109.3724	143.0194	48.6141	109.3720	143.0180

Nodal temperatures in both cases are nearly the same. Because of the negligible impact of the mass matrix lumping on a calculation time it has not been taken into account at this stage. However, in this particular case lumping has serious impact on results. Table 6 shows nodal temperatures after mass matrix lumping involved.

Table 6: Results from MATLAB and PATRAN. Single DBC and NBC. DLMM.

	MATLAB			PATRAN		
Time, s:	20	1500	2500	20	1500	2500
Node:	Temperature, ˚C:			Temperature, ˚C:		
1	50.0000	50.0000	49.9999	50.0000	50.0000	50.0000
50	50.0000	50.0000	50.0000	50.0000	50.0000	50.0000
100	50.0047	61.1171	71.2368	50.0047	61.1170	71.2367
150	50.0000	57.6899	70.8089	50.0000	57.6898	70.8087
200	50.0033	64.0506	81.0321	50.0033	64.0503	81.0317
250	50.0082	130.0064	159.7537	50.0082	130.0050	159.7519
264	**51.3226**	**282.8759**	**316.1018**	**51.3230**	**282.8739**	**316.0989**
300	50.0141	62.9922	82.888695	50.0141	62.9921	82.8883
350	50.2543	102.7736	134.0306	50.2542	102.7730	134.0299
400	49.8913	91.7538	125.2727	49.8913	91.7536	125.2720
406	50.8818	105.5617	139.8922	50.8816	105.5609	139.8910

It should be noticed that lumping introduces some errors. In case of early iterations such error reaches even 37 % (Table 7) and decreases over time. The temperature at node 264 in the first

iteration (20 s) with CMM equals to about 82.3˚C while the temperature at the same node and iteration, but with DLMM, is equal to about 51.3˚C. This gives the difference of 30˚C (37 %). Research project (described in chapter 1.1) requires as much accuracy as possible and, unfortunately, lumping noticeably decreases program's reliability simultaneously without providing any benefits (for instance, shorter calculation time).

Table 7: Errors generated by mass matrix lumping.

Node:	MATLAB			PATRAN		
	Error, %:			Error, %:		
1	0	0	3.8E-07	0	0	0
50	0	0	0	0	0	0
100	0.20	1.36	1.27	0.20	1.36	1.27
150	0.01	0.48	0.35	0.01	0.48	0.35
200	0.02	0.76	0.14	0.02	0.76	0.14
250	3.81	2.26	1.23	3.81	2.26	1.23
264	**37.63**	**1.88**	**0.98**	**37.63**	**1.88**	**0.98**
300	0.00	0.81	1.53	0.00	0.81	1.53
350	1.47	0.48	0.41	1.47	0.48	0.41
400	1.16	3.45	2.39	1.16	3.45	2.39
406	4.66	3.48	2.19	4.66	3.48	2.19

3.3 Target geometry.

Previously carried out tests have shown that, the program works properly so from this moment it was treated as a 'black box'. This approach is dictated by large number of nodes (and data in general) what indicates much more difficult debugging. An additional function had been implemented for adjusting the diameter of the laser's spot which, during the measurements, may be changed. Function for proper locating of the heat flux had been added also. At this phase, real sample's geometry (that will be used in the future in the project) and real heat flux (heat flux that actually can be applied during the measurements) have been modeled in Ansys Workbench and were used as final benchmark which includes:

- mass matrix lumping impact on the solution,
- comparison of the Ansys' and the developed code's calculation times and memory usage,
- comparison of different factorization methods,
- comparison of factorizations and PCG,

Additional example including quarter of a cube has been investigated also.

All of the following analyzes were carried out with use of the same hardware and software: Intel® Core™ i7-3770 CPU @ 3.40GHz 3.4GHz with 8Gb RAM on 64-bit operating system Windows 7 SP1, Ansys 16.2 (student license), MATLAB 7.9 2009b (academic license).

The geometry includes whole sample showed at Figure 17thatcube with dimensions 0.05 x 0.05 x 0.05 m. The mesh is obtained with Sweep method. Number of elements: 7085. Number of nodes: 31150.

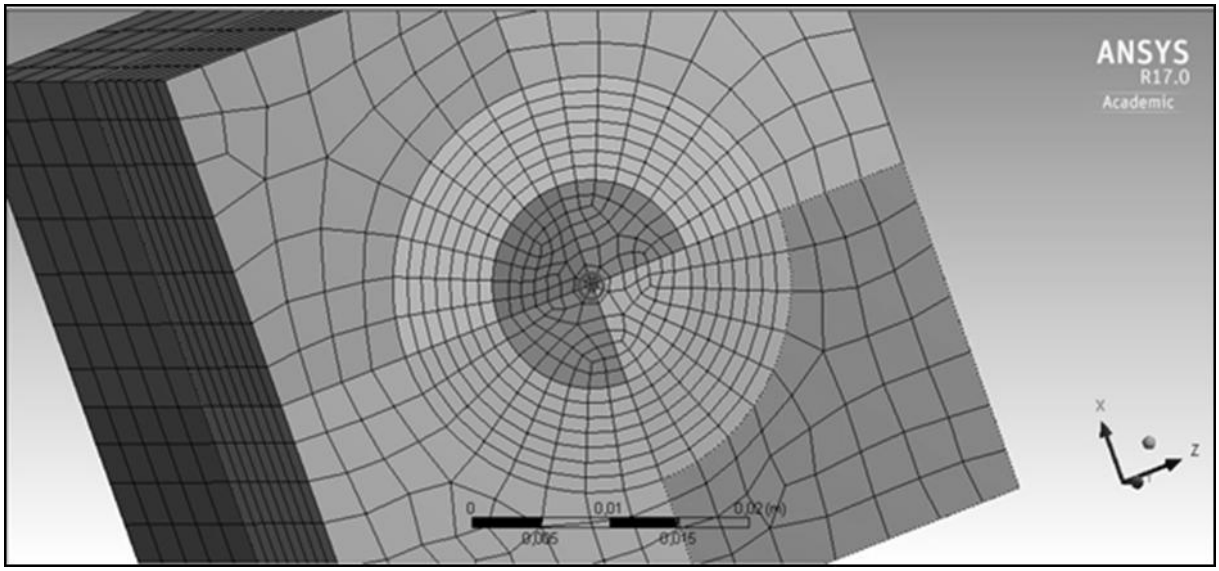


Figure 17: Whole sample. Emission face is marked with the circle.

It should be stressed that, the elements of the mesh presented above has low quality and the mesh is insufficiently fine so the results are not accurate. However, main purpose of this analysis was to check if the developed program gives the accurate results in comparison with Ansys.

Benchmark sample's material is overwritten structural steel in Ansys Workbench with the following properties:

- Density – 1091 $^{kg}/_{m^3}$.
- Orthotropic TC:
  - X: 5.5 $^{W}/_{m\,°C}$.
  - Y: 5.5 $^{W}/_{m\,°C}$.
  - Z: 5.5 $^{W}/_{m\,°C}$.
- Specific Heat – 900 $^{J}/_{kg\,°C}$.

All calculations were carried out with the same setup which consists of:

- Laser's beam radius: 0.0005 m.
- Laser's heat flux: 8.5158e+007 $^{W}/_{m}$.
- Initial temperature: 18˚C.
- Laser emission time: 0.19 s.

However, two cases were analyzed with different number of iterations and time steps:

- Short – 362 iterations.
- Long – 436 iterations.

More information about considered analysis' variants can be found in Table 8. In case of long analysis, IR camera recording time steps are in random order, so factorization had to be performed almost every two iterations – this means increased computational effort.

Table 8: Specification of the performed analyzes.

Analysis' variants.						
Short:			Long:			
Steps:	Time step, s:	Time, s:	Steps:	Time step, s:	Time, s:	
190	0.001	0.19	190	0.001	0.19	Laser's emission.
1	0.0012	0.1912	1	0.0009	0.1909	Indirect time step.
-	-	-	1	0.0011	0.192	
166	0.0018	0.49	125	0.0018	0.417	Recording the temperature field (random time steps).
1	0.0018	0.4918	1	0.007	0.424	
1	0.0019	0.4937	1	0.008	0.432	
1	0.002	0.4957	1	0.015	0.447	
1	0.0021	0.4978	42	0.016	1.119	
1	0.0022	0.5	68	0.017	2.275	
-	-	-	3	0.018	2.329	
-	-	-	1	0.019	2.348	
-	-	-	2	0.026	2.4	

3.3.1 Results – full cube.

Figures below (Figure 18, Figure 19and Figure 20) present results of most accurate methods. For MATLAB it is factorization with CMM and for the Ansys it is direct solver with CMM as well. Only heated surface is presented. Additionally, Figure 18, Figure 19 and Figure 20 have different scales. If they had not been different, due to high temperature difference (over 5400°C) Figure 18 and Figure 20 would not have been readable (there would not have been anything to see).

Figure 18 shows numerically obtained temperature distributions in first time step (after 0.001 s) in MATLAB and in Ansys respectively. This figure provides important information: in MATLAB, location of the applied heat flux is the same as in Ansys (what means previously added function works well), diameters of the obtained bright dots are not noticeably different from each other.

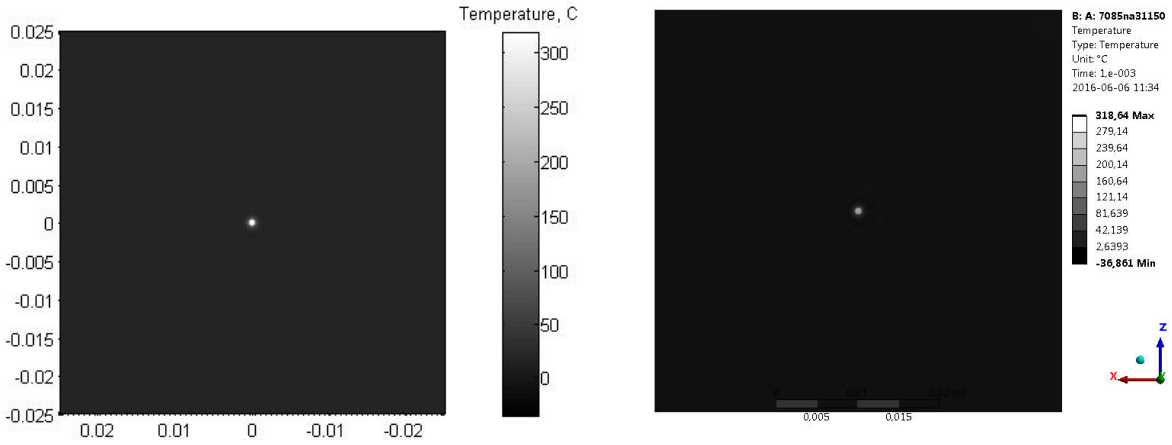


Figure 18: Temperature distribution at 0.001 s.
MATLAB (left hand side): Max: 317.878˚C. Min: -34.432˚C.
Ansys (right hand side)Max: 318.64˚C. Min: -36.861˚C.

Negative temperatures occurred due to way of applying NBC. In case of serendipity 8 – node quadrilateral elements (surface of hexahedron to which NBC is prescribed), coefficients corresponding to corner nodes are negative. When NBC is applied to several adjacent elements (so they have common nodes) the negative coefficient in RHS (in case of common corner node) is sum of negative coefficients from all elements that this particular node belongs to. In addition to that, those coefficients in RHS depend on heat flux (in this analysis it is 8.5158e+007 $^{W}/_{m}$) and on det[**J**]$^{\mathbf{2D}}$. So huge heat flux in comparison to the surface (it is 66.83 W) and very short time step results in negative temperatures at the corner nodes of specified surfaces of the boundary elements.

Figure 19 and shows numerically obtained temperature distributions in last iteration when NBC is applied (0,19 s) – end of laser's emission (maximum temperature during the whole process).

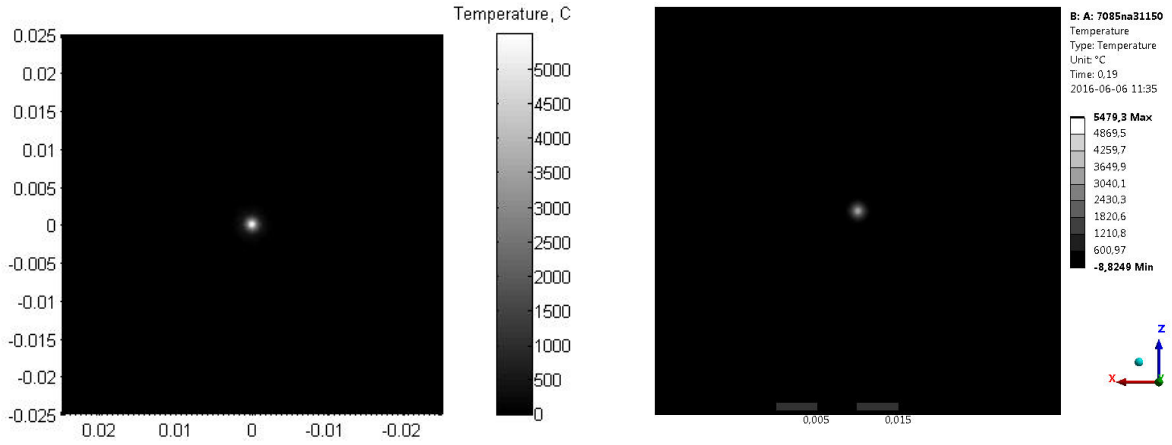


Figure 19: Temperature distribution at 0.19 s.
MATLAB (left hand side): Max: 5477.259˚C. Min: -8.904˚C.
Ansys (right hand side) Max: 5479.3˚C. Min: -8.825˚C.

Considering now again the temperatures, one should notice that the maximum temperatures are extremely high. Main reason for this situation is that only NBC was applied, with no heat losses due to radiation and convection. Another thing worth mentioning is the negative temperatures. Obviously 0.19 s is too short time interval to model heat flux properly (physically).

Figure 20 shows numerically obtained temperature distributions in last iteration of the short analysis' variant.

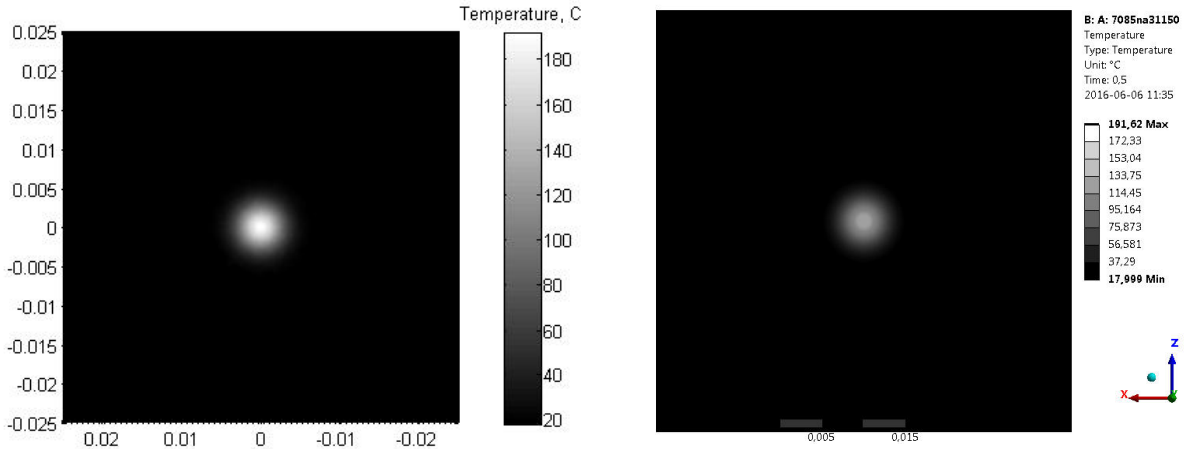


Figure 20: Temperature distribution at 0.5 s.
MATLAB (left hand side): Max: 191.329˚C. Min: 17.999˚C.
Ansys (right hand side): Max: 191.62˚C. Min: 17.999˚C.

In this case all temperatures are positive, moreover minimum temperatures are equal to the initial temperature. Differences between corresponding to each other nodal temperatures are negligible also. This leads to the conclusion that, the results become more reliable after switching off the laser's source (NBC). Obviously use of the results from the final iterations (some time after the end of the laser's emission) for determining the TC will provide maximum accuracy. Moreover, spot's diameter noticeably increased, mainly due to time (the heat has time to spread) and smaller temperature difference (18 – 191˚C) which affects the color scale (displaying the results).

3.3.2 Solvers' performances – full cube.

Simulations in MATLAB have been carried out with use of two methods:

- Direct method with use of factorizations (LU, LDL^T, Cholesky, QR).
- Iterative method with use of PCG provided by MATLAB [21] with no preconditioner.

Somehow, at least in MATLAB 2009b, preparation of the preconditioner takes much longer than the analysis with use of PCG without preconditioners, so it has been used without them. Another disadvantage of MATLAB's PCG is its syntax, namely, while no preconditioner is used, change of the initial guess becomes impossible due to the lack of input arguments required by the function to work. When initial guess is not defined, PCG starts each iteration

from 0 what leads directly to lose of the efficiency. In other words, to define the initial guess all other input arguments of the function are necessary.

In further analyzes Ansys' direct method with Consistent Mass Matrix has been assumed as the one providing an exact solution. All other methods were referenced to it. Moreover, in the following part several abbreviations appear with common pattern Software_method_mass matrix:

- A_d_cmm (Ansys_direct_Consistent Mass Matrix),
- A_d_dlmm (Ansys_direct_Diagonally Lumped Mass Matrix),
- A_it_cmm (Ansys_iterative_CMM),
- A_it_dlmm (Ansys_iterative_DLMM),
- ML_chol_cmm (MATLAB_Cholesky_CMM),
- ML_chol_dlmm (MATLAB_Cholesky_DLMM),
- ML_ldl_cmm (MATLAB_LdL^T_CMM),
- ML_ldl_dlmm,
- ML_lu_cmm (MATLAB_LU_CMM),
- ML_lu_dlmm,
- ML_qr_cmm (MATLAB_QR_CMM),
- ML_pcg_cmm.

Figure 21 presents accuracy of Ansys' direct and iterative solvers with and without mass matrix lumping. All errors were calculated with Eq.(18). Differences have been computed for all nodes corresponding to each other and then mean value was taken into account. As the A_d_cmm has been chosen as a reference solution, it lies directly on the abscissa.

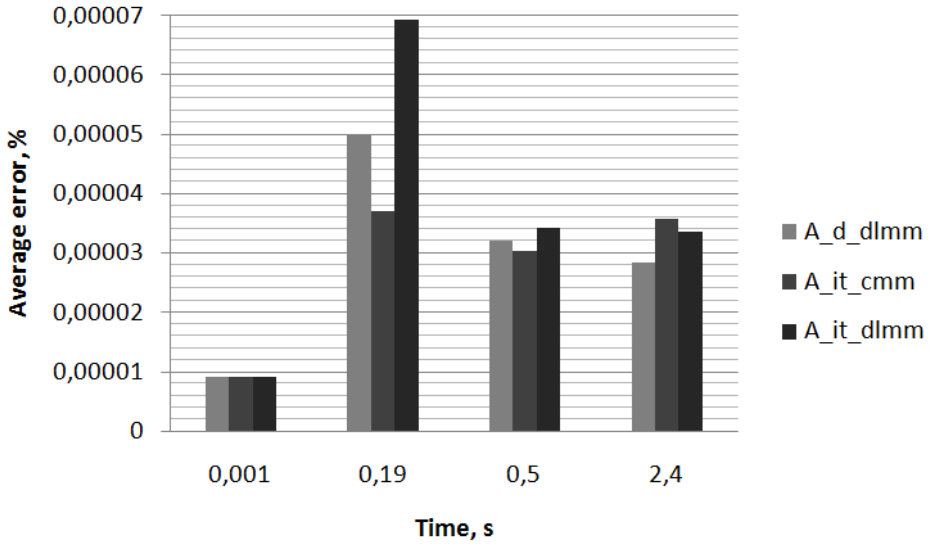


Figure 21: Average differences in chosen time steps. Only Ansys' solvers included.

As one can notice, all of the Ansys' solvers are very accurate because of insignificant results' differences. Nevertheless, those differences increase over time until NBC is switched off and then their stabilizes themselves.

Figure 22 presents accuracy of solvers used in MATLAB. Average differences are calculated in the same manner as in case of Ansys' solvers described above.

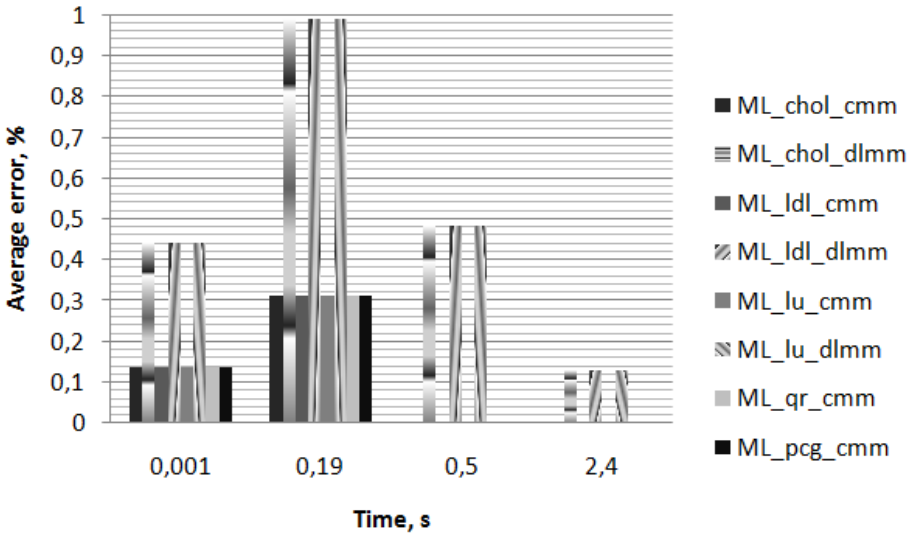


Figure 22: Average differences in chosen time steps. MATLAB.

It is important to notice that, all factorizations with use of CMM have the same accuracy. Similarly factorizations with use of DLMM. Differences between them are much higher than in case of Ansys' solvers. Moreover, mass matrix lumping has significant impact on the results contrary to the lumping in Ansys. It also should be noted that in both cases differences grow with time due to the growth of the nodal temperatures. Obviously, NBC applied in MATLAB is burdened with inaccuracy.

Table 9 contains the measured analyzes times and required memory of all tested methods of solving systems of linear equations. In case of ML_qr_cmm and ML_pcg_cmm long analysis variant has not been carried out due to very long analysis time of the short variant. Required memory as well as analyzes times for Ansys' solvers have been found in solver output files (located at the bottom of folder with project's files). In case of MATLAB, required memory has been computed as a sum of memory used by all variables existing in the workspace right after factorization or first iteration (in case of PCG).

Table 9: Analysis time and required memory by tested methods.

Method:	Short:		Long:	
	Analysis time, s:	Requiredmemory, Mb:	Analysis time, s:	Requiredmemory, Mb:
A_d_cmm	649.0	174.0	759.3	174.0
A_d_dlmm	628.8	174.0	746.9	174.0
A_it_cmm	629.4	174.0	743.4	174.0
A_it_dlmm	624.4	174.0	753.5	174.0
ML_chol_cmm	537.7	541.2	882.2	541.2
ML_chol_dlmm	530.3	515.7	884.0	515.7

ML_ldl_cmm	594.7	547.6	1000.4	547.6
ML_ldl_dlmm	592.0	522.1	1000.1	522.1
ML_lu_cmm	410.6	992.1	983.8	992.1
ML_lu_dlmm	401.8	966.7	991.3	966.7
ML_qr_cmm	2512.8	1325.5	-	-
ML_pcg_cmm	1246.9	55.0	-	-

Figure 23 presents analyzes times obtained with all tested methods. QR factorization and MATLAB's PCG (with no preconditioner) have significantly worse performances so they have been omitted in the short analysis with DLMM and in the whole long variant. Presented times contain all computational processes from the very beginning to the end of the analyzes. In Ansys it is from the start of the solution to its end and in MATLAB from import data from Ansys' solver output file .dat through preparing all necessary matrices (building system of equations) to the end of calculations. It is important to keep in mind that, the MATLAB used to the analyzes has no parallel computing toolbox, so only one core has been used. For this reason all times concerning analyzes carried out in Ansys are the total CPU time summed for all threads (it can be also found in solver output file).

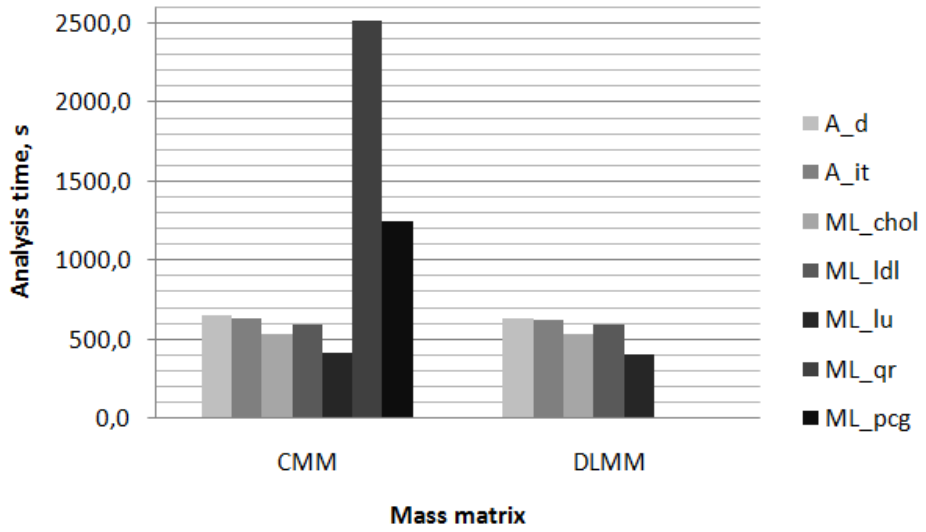


Figure 23: Analyzes times of all tested methods – short variant.

It is important to notice that, mass matrix lumping has no significant effect on the analyzes' times. It confirms the fact known from the literature [16], that in case of implicit scheme (no mass matrix 'inversion') lumping has no major sense. What is more, the program with use of LU, LdL^T and Cholesky factorizations is slightly faster than Ansys (short variant).

Figure 24 presents times of analyzes used in long variant of analysis. Five methods were tested in total.

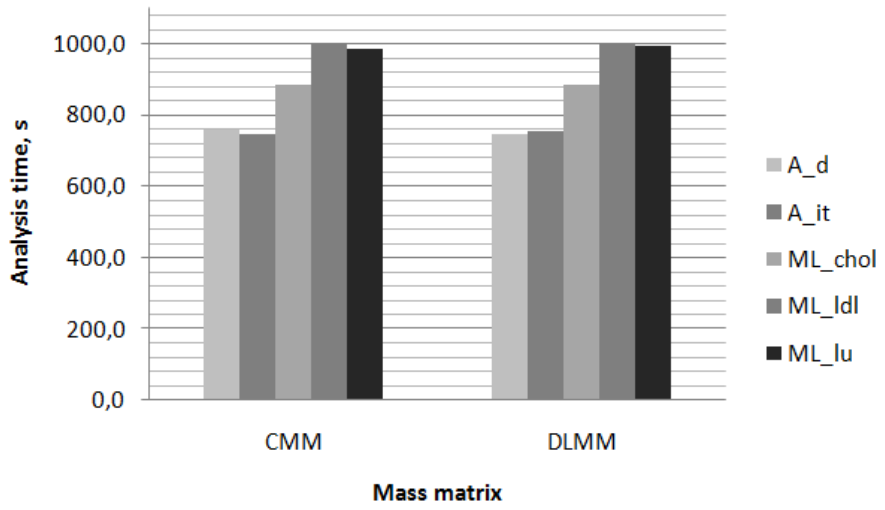


Figure 24: Analysis time of all tested methods – long variant.

In contrast to the short variant (Figure 23), Ansys in this case has better performance than the code written in MATLAB. Furthermore, once again times of the analyzes hardly depend on the mass matrix lumping which, in turn, seriously affects the results in case of the MATLAB. Additionally, LU factorization was the fastest method (in MATLAB) in short variant but in this case the Cholesky turned out to be faster. Explanation to this is as follows – solver written in MATLAB saves factorized matrices of different time steps to the hard drive as .mat files in temporary directory. Time consumed by saving as well as loading these matrices highly depends on their size. LU factorization is almost twice the size of the Cholesky factorization so saving and loading takes more time. Such an approach has sense in case of small meshes (like in this analysis) when saving and loading factorized matrices takes less time than carrying out whole new factorization and when time steps are not very diverse (infrequent saving, frequent loading of necessary factorized matrices). For instance, Cholesky factorization takes about 10 s, saving takes about 15 s but loading only about 2 s (these times are estimated, because each loading or saving takes different time due to temporary CPU usage by other processes).

Figure 25 presents memory used by variables. Ansys' solvers alone are far more economical than those written in MATLAB except for PCG but, as it was mentioned before, in MATLAB 2009b it had poor performance in comparison to the factorizations.

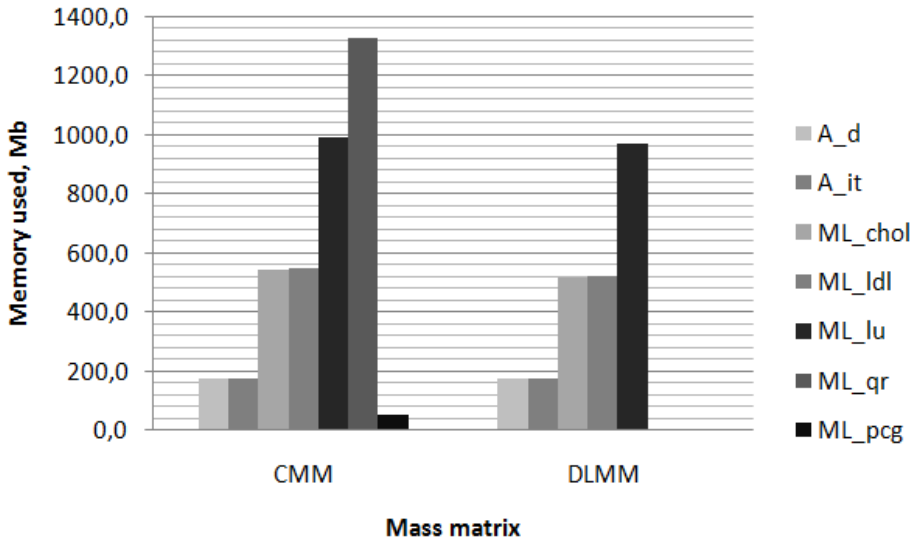


Figure 25: Memory used by tested methods.

Figure 25 does not show this clearly, but mass matrix lumping allows to save about 25 Mb of memory in case of considered small mesh (see Table 9). It is negligible advantage. To state if Ansys actually is more economical than the program written in MATLAB, one should take into account one additional factor, namely, memory requested by the software itself. In Table 10is summarized minimum amount of memory needed to carry out the necessary calculations.

Table 10: Memory required to run the analysis.

Ansys:	Memory, Mb:
Workbench:	280*
Mechanical:	240*
Solver:	2112
Total:	**2632**
MATLAB:	500
Variables:	600**
Total:	**1100**

* estimated value.

** estimated value on the assumption of the Cholesky or LDL^T factorization in case of similar number of elements as in already considered mesh.

To run the thermal analysis with use of the Ansys, one has to open Workbench, Mechanical and additionally start the solver. Unfortunately, during starting the solver Ansys allocates 2112 Mb of RAM although in cases described earlier, uses only 174 Mb. MATLAB, in turn, requires more memory during computations but, after all, smaller amount of available memory will met the requirements. What is more, if MATLAB is started as a command line alone, required memory reduces from about 500 Mb to a little over 100 Mb. Such approach allows to perform even complex numerical analyzes with use of old hardware when the available memory is limited.

In order to check impact of saving factorized matrices to the hard drive on the solver's performance, two most promising methods (LU and Cholseky) have been chosen and tested once again. In this analysis, solver written in MATLAB does not save any matrices to the hard drive. In Table 11are gathered analyzes times and memory usage of the considered methods.

Table 11: Analyzes times and memory usage of LU and Cholesky factorizations in case of modified solver.

Method:	Long:	
	Analysis time, s:	Required memory, Mb:
A_d_cmm	759.3	174.0
ML_lu_cmm	1044.1	992.1
ML_chol_cmm	749.4	541.2

Figure 26 presents graphical comparison of the analyzes times of Ansys and two chosen factorizations in MATLAB. Long analysis.

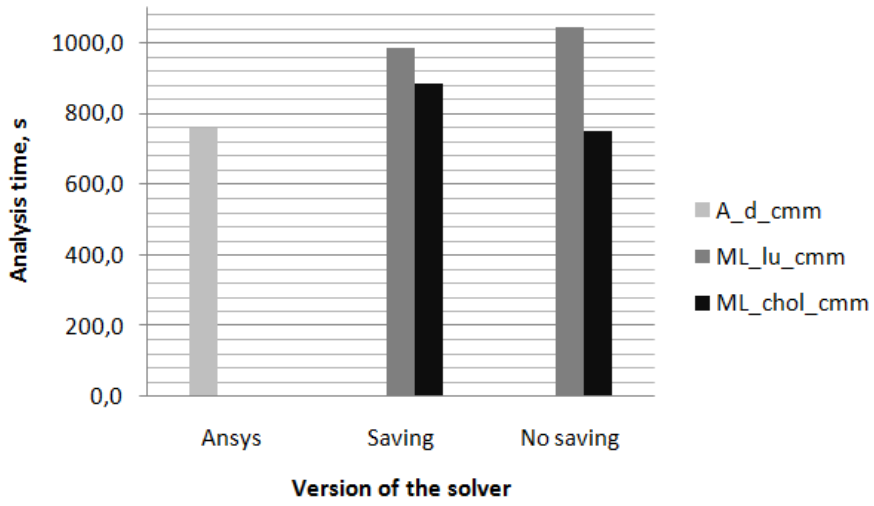


Figure 26: Solver performance with saving factorizations to the hard drive and without in comparison to Ansys' direct solver.

In case of LU factorization, solver which saves factorized matrices to the hard drive is slightly faster than one without saving. However, in case of Cholesky factorization the situation is contrary. As it was mentioned before, memory usage of LU is almost twice as the Cholesky, so saving and loading prepared matrices improves efficiency of the solver (instead of doing whole new factorization), but it is still much slower than Ansys. On the other hand Cholesky factorization based solver improved its efficiency after switching off saving and loading factorized matrices. Furthermore, its efficiency is slightly better than Ansys' direct solver.

Memory usage of these methods remained unchanged. From the viewpoint of the project, the analysis time and accuracy of the described program are crucial. However, one cannot forget about memory usage. In case of more complex meshes with increasing number of elements, memory requirements of the direct solver grow rapidly – double number of nodes results in,

approximately, fourfold increase in the size of the **[K]** and **[M]**. Excluding all other arrays, it gives 8 times more memory required to store these matrices (not mentioning the time required to build them), what affects the final performance (time needed to assembly and later for solving system of equations). Analysis of the solver always has to be supported by the investigation of the memory usage.

3.3.3 Analysis setup, geometry and mesh –quarter of a cube.

Carrying out the calculations with use of the full cube is inefficient when geometry together with boundary conditions are symmetric. In such a case, usage of part of a whole domain is far more economical because it significantly affects the computational effort simultaneously providing the same results. To do so, all body parts of the full cube have been suppressed except for the ones belonging to chosen quarter. Because of Ansys that keeps suppressed nodes, the selected quarter was re – meshed. As side research showed, the program written in MATLAB coped with that case (suppressed nodes are present in stiffness and mass matrices, but their coefficients are empty) and as a result, correct temperature field was obtained. Whereas for all suppressed nodes it was "NaN" (Not a Number) instead of temperature value. Analysis time differs from the one obtained after re – meshing and is a little bit higher.

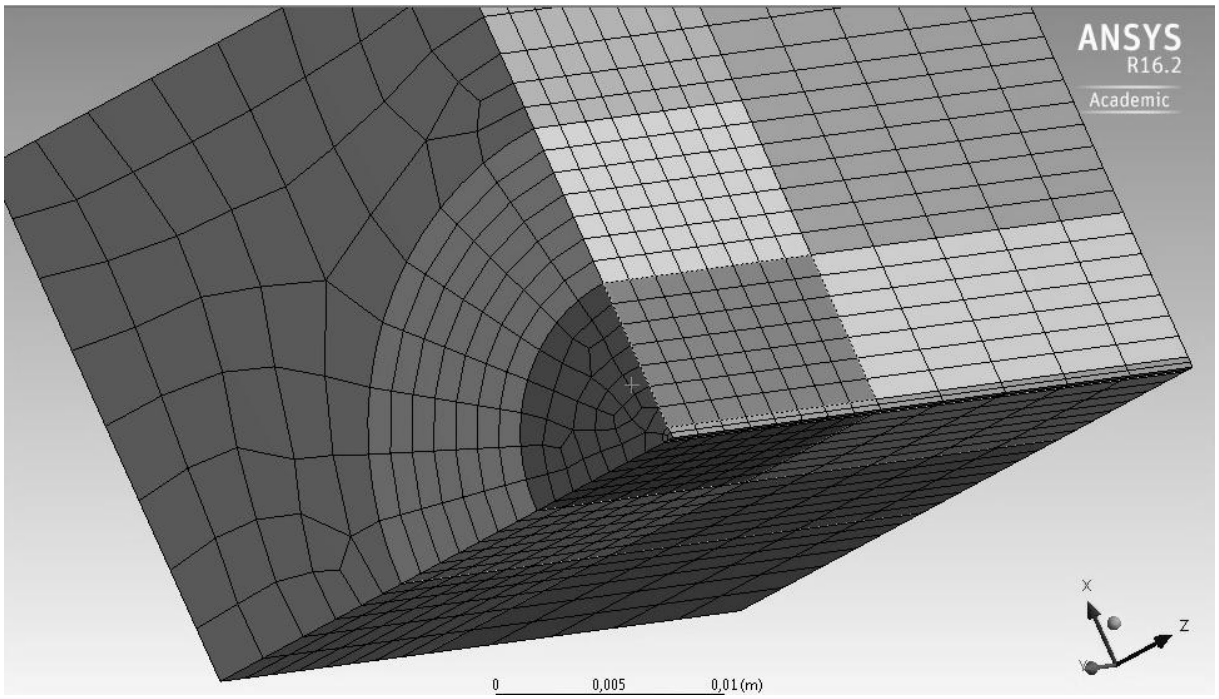


Figure 27: Mesh of a quarter of the cube. 2015 elements and 9782 nodes.

Mesh of a quarter of the cube (Figure 27) consists of: 2015 elements and 9782 nodes and has been generated in the same way as the mesh of the full cube. Whole analysis' setup remained unchanged. To check, if the program works properly, short variant (362 iterations) has been used.

3.3.4 Results – quarter of a cube.

In general, results from both analyzes (full and quarter of the cube, see Table 12) are almost identical which means that the program works properly in case of reduced domain along its axes of symmetry. Once again, too short time intervals result in negative temperatures due to negative coefficients in RHS vector. It should be noted that maximum and minimum temperatures hardly differs from those obtained in case of the full cube. Similar mesh gave similar results.

Table 12: Minimum and maximum temperatures of the full and quarter of the geometry.

	Geometry:	Full.		Quarter.	
	Time, s:	Temperature, ˚C.			
		Max.	Min.	Max.	Min.
Ansys	0.001	318.64	-36.86	318.80	-36.79
	0.19	5479.30	-8.82	5488.50	-11.09
	0.5	191.62	18.00	191.39	18.00
MATLAB	0.001	317.88	-34.43	318.06	-34.47
	0.19	5477.26	-8.90	5486.38	-12.42
	0.5	191.33	18.00	191.17	18.00

One should keep in mind that, the quarter has been re – meshed so its mesh differs from the one just after suppressing all other unnecessary bodies (after re – meshing number of elements increased by about 400) despite the fact that none of the mesh setup parameters has been changed. The maximum temperatures in case of the quarter are slightly higher than in case of the full cube. Obviously, in considered case, the results depend on the mesh so in the future further investigation will be necessary.

Figure 28 shows average percentage differences between nodal temperatures corresponding to each other. Once again, solution provided by Ansys direct solver without mass matrix lumping has been assumed as the accurate one.

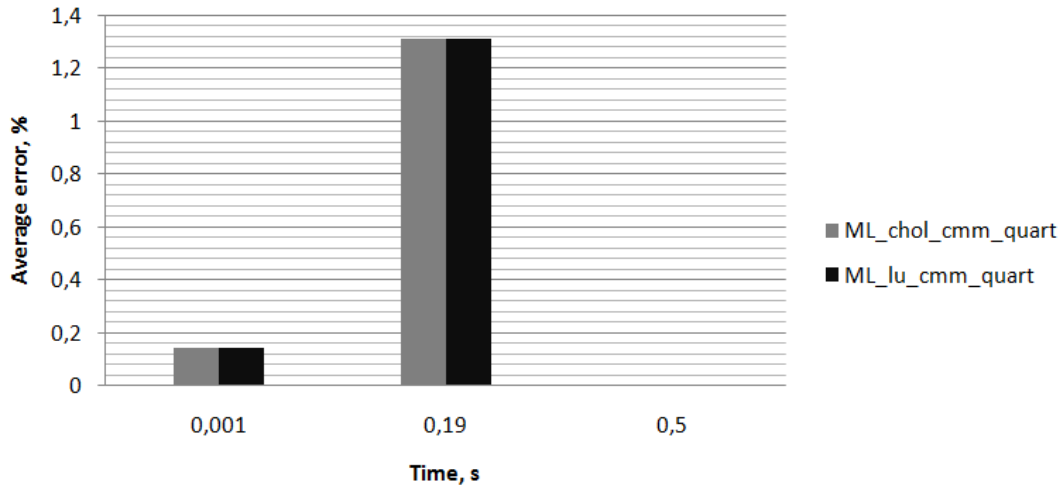


Figure 28: Accuracy comparison of the selected factorizations and Ansys' direct solver.

Again, accuracy decreases in the course of applying NBC and reaches its lowest value in the last step before turning off the heat flux. Both factorizations have the same accuracy.

Table 13 contains analyzes times and memory usage of the selected factorizations in comparison to the Ansys' direct solver. MATLAB clearly became more efficient than Ansys.

Table 13: Analyzes times and memory usage of the Ansys' direct solver, LU and Cholesky factorizations. Quarter of the geometry. Short variant.

Method:	Analysis time, s:	Memory used, Mb:
A_d_cmm	158.4	55
ML_chol_cmm	51.2	76.58
ML_lu_cmm	33.3	125.9

Figure 29 shows obtained times of the performed analyzes of the quarter of the geometry. These times are calculated in the same manner as times provided by case of whole geometry discussed before.

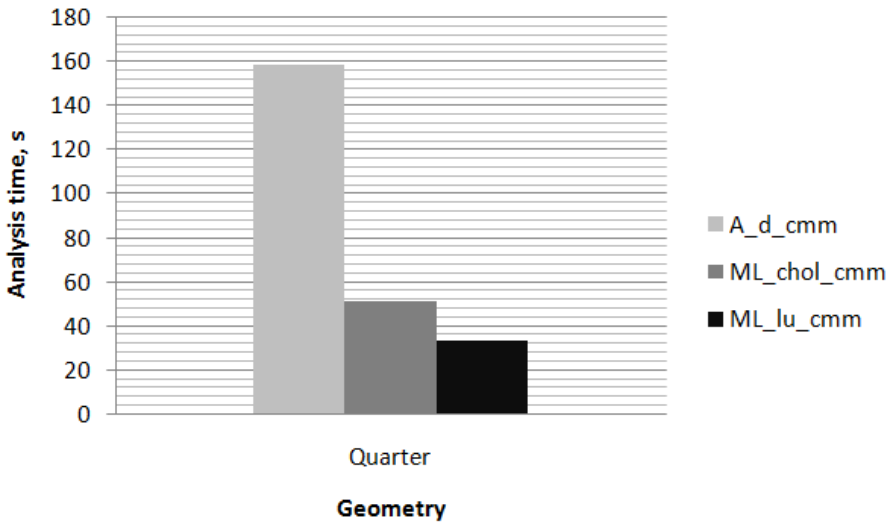


Figure 29: Analyzes times of the tested methods of solving linear systems of equations.

Reduced mesh (and geometry) results in improved performance of the program. It should be stressed that, use of the hard drive as a way of storing factorized matrices, in this analyzes, has been permanently turned off. Figure 30 presents memory usage of the considered methods. Used memory has been obtained in the same way as in the previous case.

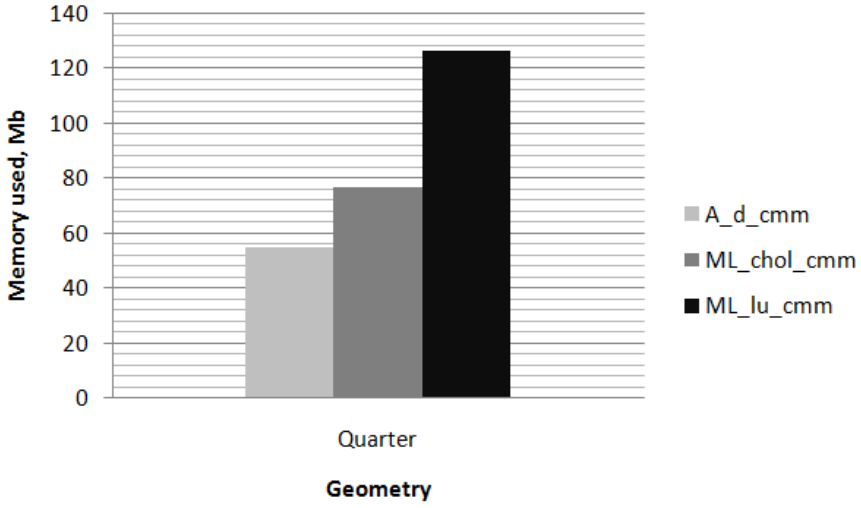


Figure 30: Memory usage of the tested methods of solving linear systems of equations.

As one can see, there is no major difference between Cholesky factorization carried out in MATLAB and Ansys' direct solver. LU factorization requires memory the most (approximately two times more than Ansys).

Finally, Cholesky factorization has been chosen as the method of solving Eq.(17) because of its satisfying performance and memory usage in both tested variants. Mass matrix will not be lumped due to serious loss of the accuracy and negligible impact on performance. Storing temporary factorized matrices will not be used because of the Cholesky factorization performance degradation.

4 Conclusions.

Performance of the code is one of the most important factors in the developing program for determining thermal conductivity in a way described at the beginning of this work. Such a program have to be efficient, because during the analysis it will be called and executed many times until convergence of the TC is reached. For example, the analysis takes 10 minutes for poorly optimized program. To determine the TC the program has to be called 100 times. This brings a total of 1000 minutes for one full analysis. Another program, much better optimized, needs 7 minutes only for completing the same one – run analysis. It also has to be called 100 times, but total time required to complete the full analysis amounts to 700 minutes only. This gives saving of 300 minutes which equals to 5 hours. This is a lot of time and that is why a good performance is so important to this program.

Mass matrix lumping is one of the available options provided by commercial software so it was tested in this work. Main cause for which it was not implemented, was lower accuracy of the code and negligible impact on the performance.

Obtained results from the MATLAB with use of 3rd order quadrature at phases 1 and 2 and with NBC applied, differ from, approximately, 0℃ to 1e-03℃ in comparison to the values provided by MSC.Patran. In case of the 2nd order quadrature those differences are much higher and they are from about 1e-03℃ to 0.9℃. They reach maximum values at the nodes to which NBC is applied and the smaller they are the further from the BC of the second kind. In case of pure DBC those differences are negligible and do not depend on the used quadrature's order.

In comparison to Ansys Transient Thermal (direct method without mass matrix lumping), differences of the results are much higher and their maximum values are, again, at the nodes to which NBC is applied. Global maximum differences are reached in the last step when NBC is applied and amount to even about 12℃ while maximum temperature is about 5500℃. After NBC removal, those differences drastically decreased to about 1e-02℃ or even less. The more distant node is (from the NBC) the lower difference is between corresponding to each other nodal temperatures provided by the Ansys' direct solver and factorization in MATLAB.

All of the Ansys' solvers (direct and iterative) were accurate with respect to each other regardless of whether lumping was used or not. Most likely some additional unknown procedure is performed, however manual does not refer to it.

Factorizations (direct method) have the same accuracy but they strongly differ from each other in case of performance. The selection of the type of the factorization should be preceded by the analysis of their efficiency for each particular case (number of nodes in the mesh, number of iterations, number of time steps, available memory). As an example, in short variant LU factorization was the best but in long variant Cholesky was better.

Mass matrix lumping in case of implicit scheme (BDM) has no major sense due to its negligible impact on the performance. On the other hand, it strongly affects the results damaging the accuracy, especially in early iterations. Lumping becomes justifiable in case of dealing with huge geometry with use of explicit scheme and when computation's time is more important than the accuracy.

Direct methods are faster and more accurate than iterative solvers but simultaneously they require far more available memory.

Neumann's Boundary Condition, which simulates the laser, generates errors that grow with time until it is turned off. The results from iterations after, approximately, 0.4 s from the beginning of the numerical analysis, have very good accuracy and can be successfully used in further computations.

Nature of the applying the NBC requires some amount of time to model temperature distribution in a physical way.

During this work the program has been developed for the purposes of the project. This program not only solves NBC – based transient heat transfer problems in orthotropic materials, but also imports geometry data, exports solution data, has log module, error reporting module and above all, is easy to manage, cheap (so to speak – free) and fully – customizable. Furthermore, its performance as well as accuracy is not worse than Ansys'. More information about the program features and capabilities can be found in Appendix A.

In the future, incorporation of the program into main algorithm for determining the TC will have to be done. This will require the implementation of the calling (this program), data managing and sharing(between the program and the main algorithm) procedures. In addition to that, radiative heat losses will have to be implemented in order to make the obtained temperature field more reliable. Last thing to do is to make density and specific heat temperature – dependent.

References

[1] Thermal Conductivity Measurements Methods, http://tpm.fsv.cvut.cz/student/documents/files/BUM1/Chapter16.pdf, access 14 June 2016.

[2] W.P. Adamczyk et al., *Retrieving thermal conductivities of isotropic and orthotropic materials*, Applied Mathematical Modelling (2015), http://dx.doi.org/10.1016/j.apm.2015.10.028.

[3] J. Parker, R. Jenkins, C. Butler, G. Abbott, Flash method of determining thermal diffusivity, heat capacity and thermal conductivity, J. Appl. Phys. 32 (9) (1961) 1679–1684.

[4] Description of MATLAB, website of the *MathWorks*, http://www.mathworks.com/products/matlab/, access 2 June 2016.

[5] Website of the *FileExchange* provided by the *MathWorks*, http://www.mathworks.com/matlabcentral/fileexchange/, access 2 June 2016.

[6] E. Kostowski, Przepływ Ciepła, Wydawnictwo Politechniki Śląskiej, Gliwice 2006, pp. 28.

[7] A. J. Nowak, ed, *Numerical Methods in Heat Transfer*, International Studies in Science and Engineering, Gliwice, 2009, pp.77-96.

[8] J. Szargut, *Modelowanie numeryczne pól temperatury*, Wydawnictwo Naukowo - Techniczne, Warszawa, 1992, pp. 113-115.

[9] O. C. Zienkiewicz, Metoda elementów skończonych, Warszawa, 1972, pp. 123-125.

[10] Finite Element Method – shape functions, website of the University of Colorado Boulder, http://www.colorado.edu/engineering/CAS/courses.d/AFEM.d/AFEM.Ch11.d/AFEM.Ch11.pdf, access 2 June 2016.
[11] R.W. Lewis, K. Morgan, H.R. Thomas, K.N. Seetharamu, *The Finite Element Method in Heat Transfer Analysis*, Wiley, 1996.
[12] Serendipity family of finite elements, website of the *Cornell University Library*, http://arxiv.org/pdf/1101.0645.pdf, access 2 June 2016.
[13] Gaussian quadrature, website of the *California State University Fullerton*, *Department of Mathematics*, http://mathfaculty.fullerton.edu/mathews/n2003/GaussianQuadMod.html, access 2 June 2016.
[14] Jacobian matrix, website of the *StachExchange*, *Mathematics*, http://math.stackexchange.com/questions/14952/what-is-jacobian-matrix, access 2 June 2016.
[15] Sarru's rule for calculating 3x3 matrix determinant, website of the *Wikipedia*, https://en.wikipedia.org/wiki/Rule_of_Sarrus, access 2 June 2016.
[16] Lumped and consistent mass matrices, website of the *Kielce University of Technology*, *Department of Informatics*, http://kis.tu.kielce.pl//mo/COLORADO_FEM/colorado/IFEM.Ch31.pdf, access 2 June 2016.
[17] Left division, website of the *MathWorks*, *Support*, http://www.mathworks.com/help/matlab/ref/mldivide.html?refresh=true, access 2 June 2016.
[18] P. Tarvydas, A. Noreika, *Usability Evaluation of Finite Element Method Equation Solvers*, Electronics and Electrical Engineering, 2007, No. 2(74).
[19] Factorization written by Timothy Davis, *FileExchange*, http://www.mathworks.com/matlabcentral/fileexchange/24119-don-t-let-that-inv-go-past-your-eyes--to-solve-that-system--factorize-, access 2 June 2016.
[20] T. A. Davis, *Algorithm 9xx, Factorize: an object – oriented linear system solver for MATLAB*, ACM Transactions on Mathematical Software, Vol. V, No. N, M 20YY, pp. 1-20.
[21] Preconditioned Conjugate Gradient, website of the *Mathworks*, *Support*, http://www.mathworks.com/help/matlab/ref/pcg.html, access 2 June 2016.

Acknowledgements

I would like to express my immense gratitude to my supervisor dr ing. Arkadiusz Ryfa for his infinite patience and all his time he devoted to me and for that, he taught me basics of the Finite Element Method.

Necessary hardware and software have been provided by the Institute of Thermal Technology for which I cordially thank.

I would like also wholeheartedly thank my dear mother Ewa Tokarska for all the support she shown to me during my studies. Thank you, Mum.

Analiza numeryczna pola temperatury w ortotropowej próbce

Mieszko Tokarski[2]

e-mail: tomiesio@gmail.com

Słowa kluczowe: MATLAB, Ansys, FEM, Warunek Brzegowy Neumann'a, Faktoryzacja

Streszczenie

W pracy tej opisano proces tworzenia pogramu opartego na Metodzie Elementów Skończonych a służącego do obliczania pola temperatury w ortotropowej próbce z wykorzystaniem warunku brzegowego Neumann'a. Program ten został stworzony na potrzeby projektu prowadzonego w Instytucie Techniki Cieplnej w Gliwicach. Stanowi on ważną część zautomatyzowanej procedury mającej wyznaczać współczynnik przewodzenia ciepła badanej próbki poprzez dopasowanie rozkładu temperatury otrzymanego na drodze obliczeń numerycznych z rozkładem temperatury otrzymanym z pomiarów. Z racji specyfiki pomiarów oraz samej procedury sterującej całą analizą, stworzony program charakteryzuje się dużą wydajnością porównywalną do wydajności Ansysa (dla tego samego przypadku), wystarczającą dokładnością oraz przygotowaniem do sprzęgnięcia z głównym algorytmem. Najważniejszymi cechami tego programu są: moduł do importu geometrii z Ansysa, moduł do eksportu danych dla algorytmu sterującego, dwa pliki tekstowe umożliwiające łatwe zarządzanie tym kodem (zwłaszcza w przypadku sterowania przez zewnętrzny program), logowanie działania programu oraz raportowanie o błędach.

[2]*Autor przygotował niniejszy rozdział podczas pracy nad projektem dyplomowym magisterskim wykonywanym przez autora w Instytucie Techniki Cieplnej na Wydziale Inżynierii Środowiska i Energetyki Politechniki Śląskiej pod opieką dr inż*. Arkadiusza Ryfy. *Prace były prowadzone w ramach projektu naukowego realizowanego w Instytucie Techniki Cieplnej.*

Appendix A

Features of the developed program

One can distinguish three main characteristic features of the code:

- Consistency – the code written in MATLAB is executed line by line, so it is logical and without any unnecessary commands, for example clearing variable just to load it again in another line. The same applies to the variables and external functions – they are well named to make sense without thinking too much. Repetition of the lines of the code is also eliminated, because this makes editing and debugging the code easier due to lack of necessity of modifying additional lines.
- Simplicity and transparency – program has been written in such a way that it can be easily debugged and modified. This has been achieved by the use of basic functions, for instance 'For' loops instead of more sophisticated nested functions like 'cellfun' or 'bsxfun'. In case of MATLAB, 'For' loop has better performance than most of the other functions. Another thing that makes debugging easier is modular design. Grouping of the code into smaller pieces responsible for strictly specified tasks is a good habit and it helps one finding procedural errors.
- Forethought – this program can be easily extended with new functions and procedures without major changes into already existing code.

The program is started with use of batch file to call MATLAB's command line and to declare all necessary directories automatically. It consists of:

- startPanel_0 - main procedure which calls and coordinates all others,
    - globalVar – structural variable storing all global variables (program's memory),
    - var_GDefine_1 – import module, uses UserSettings.txt,
        - import_NCoords_2 – subprogram for nodal coordinates import from Ansys' geometry output files,
        - import_Connectivity_3 – subprogram for connectivity matrix import import from Ansys' geometry output files,
        - import_MConstants_4 – subprogram for material properties import from Ansys' geometry output files,
        - import_FENodes_5 – subprogram for face and emission nodes import from Ansys' geometry output files,
        - import_USetup_6 – subprogram for user setup data import from UserSetup.txt file,
    - un_varDelete_0 – subprogram responsible for clearing all of unnecessary variables from memory,
    - MatCon - main subprogram for building system of equations,
        - Ansys_N_order – function responsible for proper nodal reordering,
        - Ansys_N_order_8n – function responsible for building connectivity matrix in order to apply NBC,
        - LaserRadius – function controlling laser's diameter,
        - Ni – function with shape functions 20 – node hexahedron, PATRAN's order,

        - Shape_functions_Patran_8n – function with shape function 8 – node quadrilateral, PATRAN's order,
        - dNdksi_Patran_20n – function with shape functions' 3d derivatives,
        - dNdksi_PAtran_8n – function with shape functions' 2d derivatives,
    - Solver – main subprogram for solving linear systems of equations,
        - Factorize – algorithm written by Timothy Davis,
        - MemoryCheck – function computing memory usage by variables in the workspace.

Whole program is controlled with use the of two text files (Figure 31 and Figure 32):

- UsetSettings.txt – defining geometry file and its components to import:
    - nodal coordinates,
    - connectivity matrix,
    - material properties (disabled by default),
    - face and emission nodes,
- UserSetup.txt – defining setup of the analysis:
    - density,
    - specific heat,
    - TC components x, y, z,
    - laser beam radius,
    - laser heat flux,
    - initial temperature,
    - laser emission time,
    - camera recording start time,
    - camera recording times (in fact time increments)– this vector can be almost freely long and that is why it is at the very end of the file.

The program checks for last modification date of UserSettings and UserSetup and compares them with ones already stored as variables in structural array globalVar. If they differ from each other then program overwrites them and executes proper procedures of import (when UserSettings date variable is overwritten) and/or overwrites setup variables (when UserSetup date variable is overwritten). In special case while such date variable or variables do not exist, they are created as new ones and again all necessary procedures are executed.

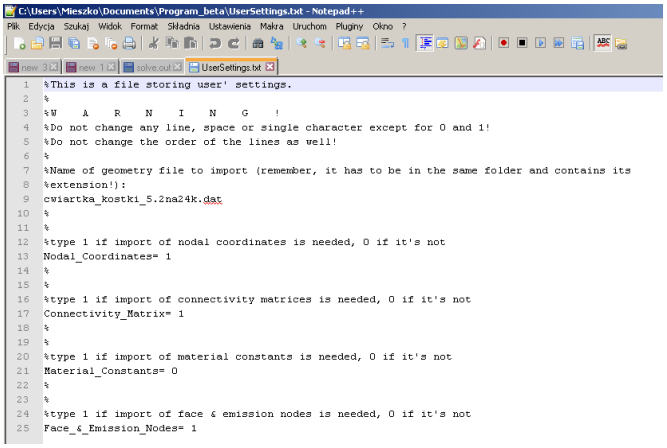


Figure 31: UserSettings.txt input file.

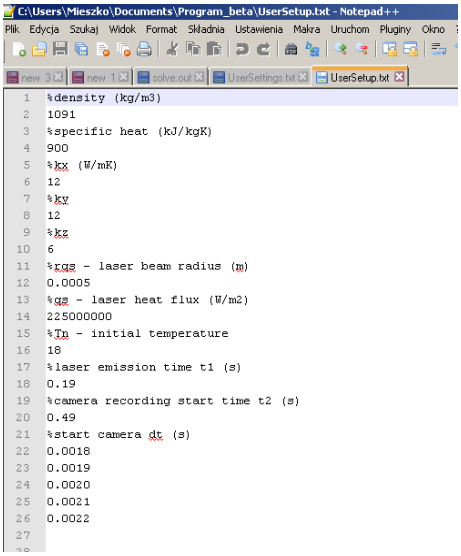


Figure 32: UserSetup.txt input file.

All subprograms and functions are vectorized and optimized in order to achieve as high performance as possible. Moreover, the program is fully automated – besides input text files (UserSettings.txt and UserSetup.txt) does not require any other user interference. It should be stressed that, with use of input files (that will be prepared by the main algorithm or user) it can be easily managed by the external procedures.

Results are saving into specially created folder in working directory named "Results_YYYY.MM.DD__hh.mm.ss", where " YYYY.MM.DD__hh.mm.ss" is current date and time:

- YYYY – year,
- MM – month,
- DD – day,
- hh – hours,
- mm – minutes,
- ss – seconds.

Results are stored as .dat files named "T(s).dat", where "(s)" is step number.

Figure 33 and Figure 34 show exemplary folder with exemplary files with results. Additionally to these files, analysis' configuration file is attached.

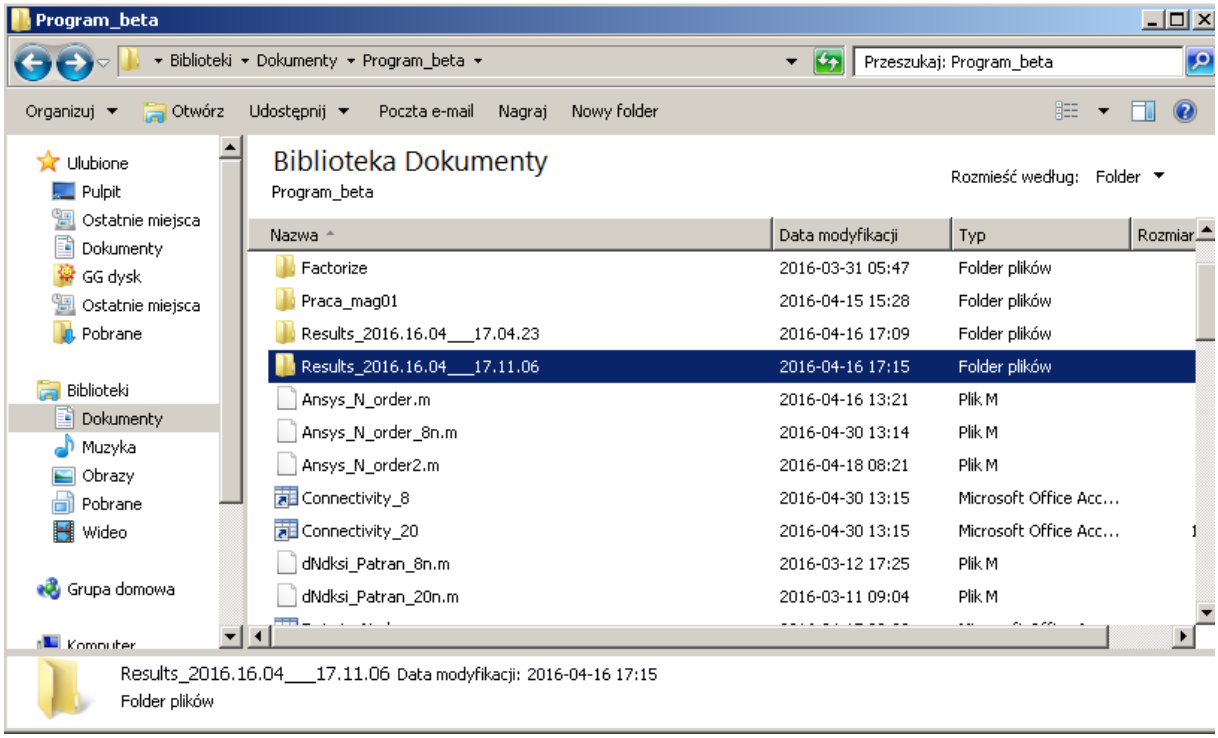


Figure 33: Folder containing analysis' results.

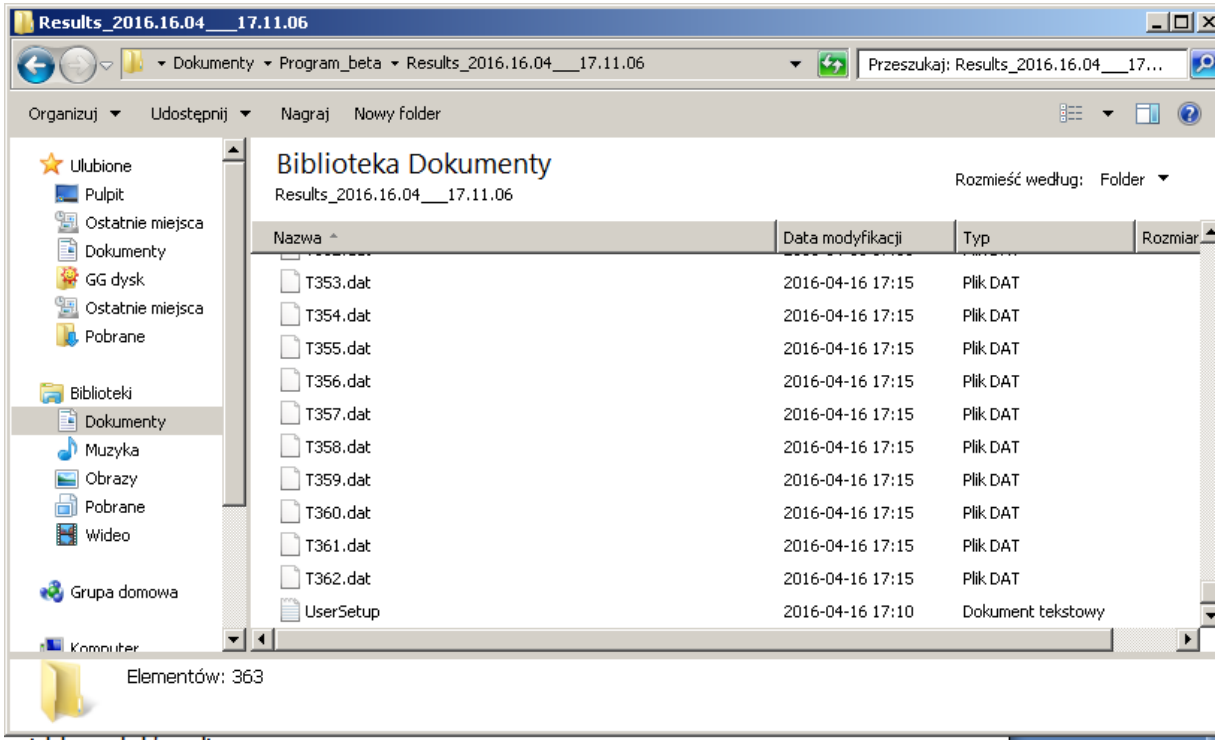
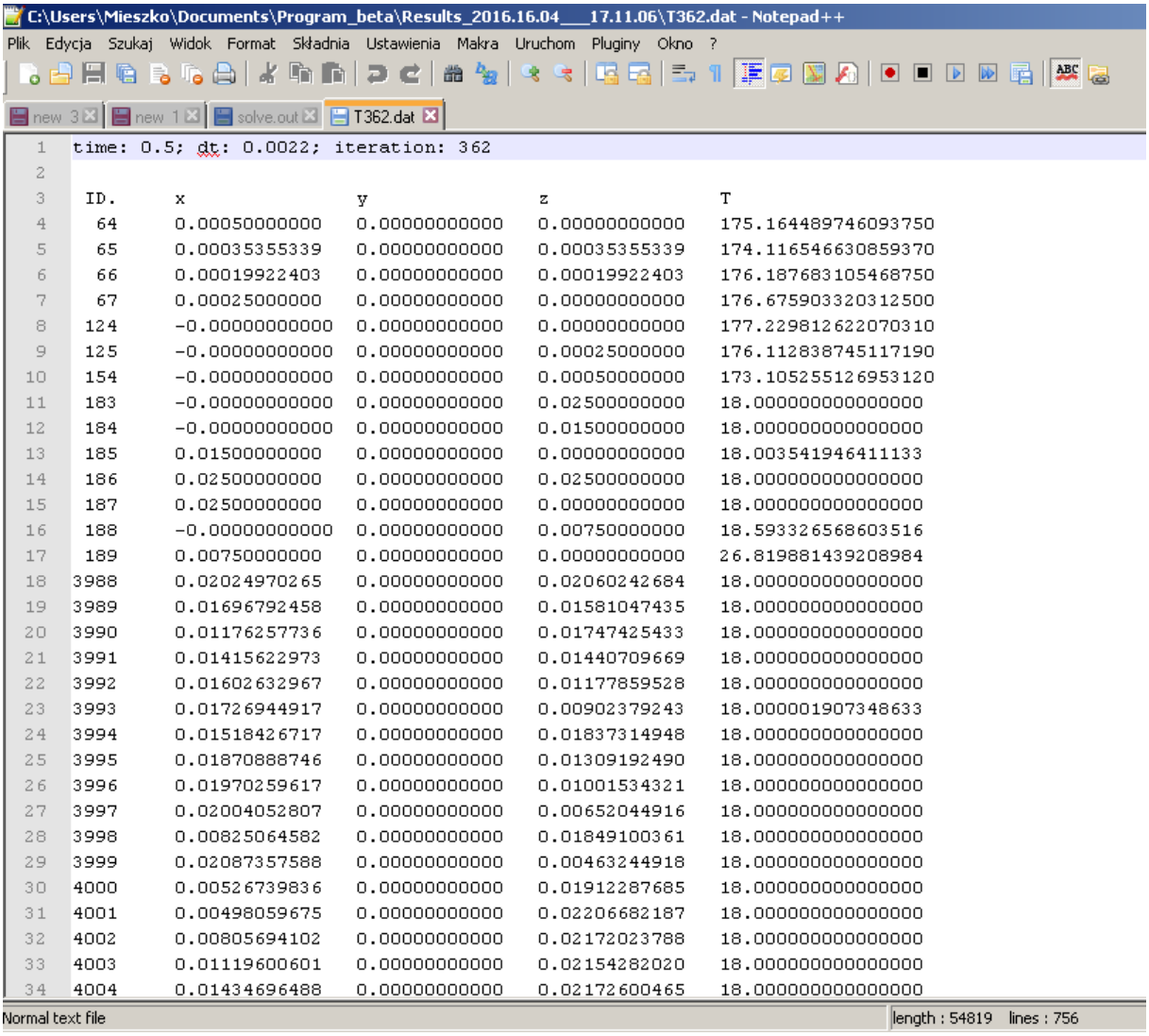


Figure 34: Files .dat with results together with UserSetup.txt which stores analysis' configuration.

Each T(s).dat file contains: all nodes of heating surface, nodal coordinates, nodal temperatures and, additionally, current analysis' time, current time step and current iteration number. Exemplary T(s).dat file is showed in Figure 35.



C:\Users\Mieszko\Documents\Program_beta\Results_2016.16.04___17.11.06\T362.dat - Notepad++

Plik Edycja Szukaj Widok Format Składnia Ustawienia Makra Uruchom Pluginy Okno ?

new 3 | new 1 | solve.out | T362.dat

```
time: 0.5; dt: 0.0022; iteration: 362

 ID.     x                y                z                T
  64     0.00050000000    0.00000000000    0.00000000000    175.164489746093750
  65     0.00035355339    0.00000000000    0.00035355339    174.116546630859370
  66     0.00019922403    0.00000000000    0.00019922403    176.187683105468750
  67     0.00025000000    0.00000000000    0.00000000000    176.675903320312500
 124    -0.00000000000    0.00000000000    0.00000000000    177.229812622070310
 125    -0.00000000000    0.00000000000    0.00025000000    176.112838745117190
 154    -0.00000000000    0.00000000000    0.00050000000    173.105255126953120
 183    -0.00000000000    0.00000000000    0.02500000000    18.000000000000000
 184    -0.00000000000    0.00000000000    0.01500000000    18.000000000000000
 185     0.01500000000    0.00000000000    0.00000000000    18.003541946411133
 186     0.02500000000    0.00000000000    0.02500000000    18.000000000000000
 187     0.02500000000    0.00000000000    0.00000000000    18.000000000000000
 188    -0.00000000000    0.00000000000    0.00750000000    18.593326568603516
 189     0.00750000000    0.00000000000    0.00000000000    26.819881439208984
3988     0.02024970265    0.00000000000    0.02060242684    18.000000000000000
3989     0.01696792458    0.00000000000    0.01581047435    18.000000000000000
3990     0.01176257736    0.00000000000    0.01747425433    18.000000000000000
3991     0.01415622973    0.00000000000    0.01440709669    18.000000000000000
3992     0.01602632967    0.00000000000    0.01177859528    18.000000000000000
3993     0.01726944917    0.00000000000    0.00902379243    18.000001907348633
3994     0.01518426717    0.00000000000    0.01837314948    18.000000000000000
3995     0.01870888746    0.00000000000    0.01309192490    18.000000000000000
3996     0.01970259617    0.00000000000    0.01001534321    18.000000000000000
3997     0.02004052807    0.00000000000    0.00652044916    18.000000000000000
3998     0.00825064582    0.00000000000    0.01849100361    18.000000000000000
3999     0.02087357588    0.00000000000    0.00463244918    18.000000000000000
4000     0.00526739836    0.00000000000    0.01912287685    18.000000000000000
4001     0.00498059675    0.00000000000    0.02206682187    18.000000000000000
4002     0.00805694102    0.00000000000    0.02172023788    18.000000000000000
4003     0.01119600601    0.00000000000    0.02154282020    18.000000000000000
4004     0.01434696488    0.00000000000    0.02172600465    18.000000000000000
```

Normal text file | length : 54819 lines : 756

Figure 35: Files .dat with results together with UserSetup.txt which stores analysis' configuration.

Logfile.txt (Figure 36) is created in the main directory. All statements about executed or skipped procedures are stored in that file. Also errors and warning reports as soon any occurs. Memory usage and analysis time are also reported. Furthermore, logfile.txt and UserSetup.txt are copied into results' folder to provide additional information about the process and its configuration each time the analysis is performed.

```
11:41:04 Removing unnecessary fields inside globalVar.
11:41:04 Removing completed.
11:41:04 globalVar variable saved.
11:41:04 Setup file has changed - importing new setup data.
11:41:04 Import of the setup data has been completed.
11:41:04 Geometry and/or setup data have/has changed - creating new matrices.
11:41:24 MatCon: Matrices K, M, F & Tn are ready.
11:41:24 Starting the analysis. This may take few minutes.
11:41:24 Currently used memory by variables, Mb: 55.0089
11:41:43 Currently used memory by variables, Mb: 541.2169
11:49:59 Analysis has been completed.
Elapsed time is 537.678607 seconds.
Maximum possible array:              12763 MB (1.338e+010 bytes) *
Memory available for all arrays:     12763 MB (1.338e+010 bytes) *
Memory used by MATLAB:                1089 MB (1.142e+009 bytes)
Physical Memory (RAM):                8056 MB (8.448e+009 bytes)

*  Limited by System Memory (physical + swap file) available.
  Name                  Size                    Bytes  Class                          Attributes

  A                 31150x31150            509826090  factorization_chol_sparse
  B                 31150x1                   249200  double
  Coordinates       31150x4                   996800  double
  F                 31150x1                   249200  double
  K                 31150x31150             27464280  double                         sparse
  M                 31150x31150             27464280  double                         sparse
  T                 31150x5                   1246000  double
  Tn                31150x1                   249200  double
```

Figure 36: Exemplary part of the logfile.txt. It provides information about executed or skipped procedures, analysis' time and memory usage.