

# FEJavaDemo

1. The considered problem.....	2
2. FE-discretisation .....	5
2.1. Type of mesh and mesh refinement .....	5
2.2. Choice of the shape functions .....	6
2.3. Storage scheme and renumbering of the nodes.....	8
2.4. Computation of the FE system of equations .....	10
2.4.1. Computation and assembling of the stiffness matrix and the right-hand side vector .....	10
2.4.2. Numerical integration .....	14
2.5. Solving process of the FE system of equations.....	16
3. Structure of the data files .....	18
3.1. The .net file .....	18
3.2. The .dat file .....	19
4. Execution of the program.....	21

# 1. The considered problem

The program FEJavaDemo enables us to solve problems of heat conduction using the finite elements method. We seek the stationary temperature field  $u(x)$  in a two-dimensional domain  $\Omega$ , which can be made of different subdomains each with its own coefficient of thermal conductivity  $\Lambda(x)$ .  $f$  represents a possible heat source inside the domain.

The software handles 3 different types of boundary conditions:

- The Dirichlet conditions (on  $\Gamma_1$ ): the temperature  $g_1(x)$  is set on certain parts of the domain's border.
- The Neumann conditions (on  $\Gamma_2$ ): the heat flux  $g_2(x)$  is given on different portions of the border (a region can be isolated by setting the flux to 0 on its borders).
- The Fourier/Robin conditions (on  $\Gamma_3$ ): a free heat exchange with the environment (the flux is proportional ( $\alpha(x)$  coefficient) to the difference between the border's temperature  $u(x)$  and the environment's temperature  $u_A(x)$ ).

*Classical formulation of the problem:*

We seek  $u \in C^2(\Omega) \cap C^1(\Omega \cup \Gamma_2 \cup \Gamma_3) \cap C(\bar{\Omega})$ , so that

$$\begin{aligned} -\operatorname{div}(\Lambda(x) \operatorname{grad} u(x)) &= f(x) & \forall x \in \Omega, \\ u(x) &= g_1(x) & \forall x \in \Gamma_1, \\ \frac{\partial u}{\partial N} &= g_2(x) & \forall x \in \Gamma_2, \\ \frac{\partial u}{\partial N} &= \alpha(x) [u_A(x) - u(x)] & \forall x \in \Gamma_3, \end{aligned}$$

with  $\Gamma = \partial\Omega = \bar{\Gamma}_1 \cup \bar{\Gamma}_2 \cup \bar{\Gamma}_3$  and  $\Gamma_i \cap \Gamma_j = \emptyset$  for  $i \neq j$

and  $\Lambda(x) = \begin{pmatrix} \lambda_{xx}(x) & 0 \\ 0 & \lambda_{yy}(x) \end{pmatrix}$  with  $\frac{\partial u}{\partial N} = \lambda_1(x) \frac{\partial u}{\partial x_1} n_1 + \lambda_2(x) \frac{\partial u}{\partial x_2} n_2$

and  $\vec{n}(x) = \begin{pmatrix} n_1(x) \\ n_2(x) \end{pmatrix}$ : the outwards normal unit vector  $x \in \Gamma$ .

*Variational formulation of the problem:*

We seek  $u \in V_{g_1}$ , so that

$$a(u, v) = \langle F, v \rangle \quad \forall v \in V_0$$

with

$$a(u, v) = \int_{\Omega} (\text{grad } v(x))^T \Lambda(x) \text{grad } v(x) dx + \int_{\Gamma_3} \alpha(x) u(x) v(x) ds ,$$

$$\langle F, v \rangle = \int_{\Omega} f(x) v(x) dx + \int_{\Gamma_2} g_2(x) v(x) ds + \int_{\Gamma_3} \alpha(x) u_A(x) v(x) ds ,$$

$$V_{g_1} = \{u \in H^1(\Omega) : u(x) = g_1(x) \text{ auf } \Gamma_1\},$$

$$V_0 = \{v \in H^1(\Omega) : v(x) = 0 \text{ auf } \Gamma_1\}.$$

Of course, the program cannot work with arbitrary functions.

- As far as the  $f$  function is concerned, many possibilities are given : the function can be different on each domain and it can be defined using the functions tan(), sin(), cos(), exp(), ln(), sqrt() and the constant  $\pi$ . For example,  $f(x,y) = 2 * \cos(\Pi * x * y)$  would be valid.
- $\Lambda$  is defined using  $\lambda_1$  and  $\lambda_2$ . Those coefficients are constant but can be different in each domain.
- For the Dirichlet boundary conditions, the temperature  $g_1(x)$  is given at some points of the mesh. Those points are called the Dirichlet nodes.

The other values which represent the boundary conditions are constant on each border. That is piecewise constant on the domain.

### **Galerkin method**

The basic idea of the Galerkin finite element method is to replace the infinite dimensional space where we seek the solution by a finite dimensional subspace  $V_h$ :

$$V_h = \left\{ v_h : v_h = \sum_{i \in \bar{\omega}_h} v_i p_i(x) \right\} = \text{span}\{p_i : i \in \bar{\omega}_h\} \subset V = H^1(\Omega)$$

with  $\bar{\omega}_h$  containing the numbers of all nodes, nodes being the points in the domain  $\Omega$  defined by a mesh where we seek this solution.

The  $p_i$  functions are the shape functions, which are linearly independent, and the coefficients  $v_i$  are chosen freely.

As we know the value of the solution at the Dirichlet nodes, we seek an approximate solution of the form

$$u_h(x) = \sum_{j \in \omega_h} u_j p_j(x) + \sum_{j \in \gamma_h} u_{*,j} p_j(x)$$

where only the  $u_j$  are unknown.

With  $\omega_h$  containing the numbers if the nodes in  $\Omega \cup \Gamma_2 \cup \Gamma_3$  and  $\gamma_h$  containing the numbers of the nodes on  $\bar{\Gamma}_1$ :  $\bar{\omega}_h = \omega_h \cup \gamma_h$ .

We then have the following system of equations:

We seek  $\underline{u}_h = [u_j]_{j \in \omega_h} \in \mathbb{R}^{N_h}$ :

$$\sum_{j \in \omega_h} u_j a(p_j, p_i) = \langle F, p_i \rangle - \sum_{j \in \gamma_h} u_{*,j} a(p_j, p_i) \quad \forall i \in \omega_h$$

i.e. we seek  $\underline{u}_h = [u_j]_{j \in \omega_h} \in \mathbb{R}^{N_h}$ :

$$K_h \underline{u}_h = \underline{f}_h$$

with

$$K_h = [a(p_j, p_i)]_{i,j \in \omega_h}$$

$$\underline{f}_h = \left[ \langle F, p_i \rangle - \sum_{j \in \gamma_h} u_{*,j} a(p_j, p_i) \right]_{i \in \omega_h} \in \mathbb{R}^{N_h}$$

$N_h$  is the number of nodes of the domain  $\Omega \cup \Gamma_2 \cup \Gamma_3$ .

## 2. FE-discretisation

### 2.1. Type of mesh and mesh refinement

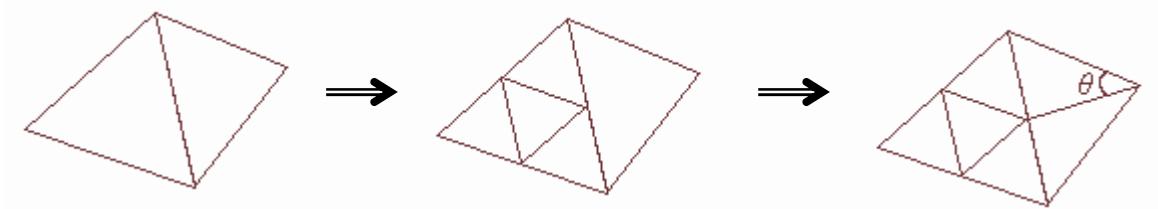
The domain  $\Omega$ , in which we seek the solution of the considered problem, is divided into elements  $T^{(r)}$ . In FEJavaDemo, triangles or quadrilaterals can be used.

If the program works with piecewise linear shape-function  $p_i$ , the mesh can be refined. After the refinement, all the steps of the process are recomputed with the new mesh. Before starting such a process, the program will ask for the maximum allowed temperature variation between two consecutive nodes, it will then refine the mesh as many times as needed in order for every temperature variation to be under the specified one. Entering a maximum variation of zero will refine the entire mesh once (every triangle will be split in four).

#### *Algorithm of the mesh refinement:*

- (1) Divide each triangle, for which the temperature between two nodes is too high, in four congruent triangles, i.e. take the mid-point of each border of the triangle and create the new triangles by connecting these points.
- (2) As long as at least one triangle has been divided
  - (i) Divide in four congruent triangles all the triangles of the refined mesh who have not been split and have, at least, two borders where new nodes have been generated by a former splitting process.
  - (ii) For each triangle which has exactly one border with a new node,
    - If a bisection of the triangle would leads to the creation of two triangle of low quality (i.e. if the angle  $\theta$  is too small, i.e.  $\theta < 25^\circ$ ) then the triangle is split in four.

- Otherwise, the triangle is cut in two by connecting the newly generated node to the opposite summit.
- (3) Solve the problem using the new mesh. If some temperature differences are still over the given value, go back to step (1).



*Mesh refinement with division in four triangles*

## 2.2. Choice of the shape functions

We choose polynomial functions  $\varphi_\alpha$  as shape functions over the reference element, such that for each node  $P_\beta$  of the reference element satisfy the equation  $\varphi_\alpha(\xi_\beta) = \delta_{\alpha\beta}$ .

The program can work with linear and quadratic functions.

In order to compute the entries of the stiffness matrix and the right-hand side, only the definition of the shape functions on the reference element are needed because the corresponding integral on any element can be computed like an integral on the references element. To do so, a transformation which turns the reference element into an element of the mesh  $T^{(r)}$  is used.

The following form functions are used:

	$\varphi_1^l(\xi) = -\xi + 1$ $\varphi_2^l(\xi) = \xi$
	$\varphi_1^q(\xi) = 2\xi^2 - 3\xi + 1$ $\varphi_2^q(\xi) = -4\xi^2 + 4\xi$ $\varphi_3^q(\xi) = 2\xi^2 - \xi$
	$\varphi_1^{l2}(\xi_1, \xi_2) = 1 - \xi_1 - \xi_2$ $\varphi_2^{l2}(\xi_1, \xi_2) = \xi_1$ $\varphi_3^{l2}(\xi_1, \xi_2) = \xi_2$
	$\varphi_{\alpha}^{q2}(\xi_1, \xi_2) = \varphi_{\alpha}^{l2}(2\varphi_{\alpha}^{l2} - 1), \quad \alpha = 1, 2, 3$ $\varphi_4^{q2}(\xi_1, \xi_2) = 4\varphi_1^{l2}\varphi_2^{l2}$ $\varphi_5^{q2}(\xi_1, \xi_2) = 4\varphi_2^{l2}\varphi_3^{l2}$ $\varphi_6^{q2}(\xi_1, \xi_2) = 4\varphi_3^{l2}\varphi_1^{l2}$
	$\varphi_1^{bl}(\xi_1, \xi_2) = \varphi_1^l(\xi_1)\varphi_1^l(\xi_2)$ $\varphi_2^{bl}(\xi_1, \xi_2) = \varphi_2^l(\xi_1)\varphi_1^l(\xi_2)$ $\varphi_3^{bl}(\xi_1, \xi_2) = \varphi_2^l(\xi_1)\varphi_2^l(\xi_2)$ $\varphi_4^{bl}(\xi_1, \xi_2) = \varphi_1^l(\xi_1)\varphi_2^l(\xi_2)$
	$\varphi_1^{bq}(\xi_1, \xi_2) = \varphi_1^q(\xi_1)\varphi_1^q(\xi_2)$ $\varphi_2^{bq}(\xi_1, \xi_2) = \varphi_3^q(\xi_1)\varphi_1^q(\xi_2)$ $\varphi_3^{bq}(\xi_1, \xi_2) = \varphi_3^q(\xi_1)\varphi_3^q(\xi_2)$ $\varphi_4^{bq}(\xi_1, \xi_2) = \varphi_1^q(\xi_1)\varphi_3^q(\xi_2)$ $\varphi_5^{bq}(\xi_1, \xi_2) = \varphi_2^q(\xi_1)\varphi_1^q(\xi_2)$ $\varphi_6^{bq}(\xi_1, \xi_2) = \varphi_3^q(\xi_1)\varphi_2^q(\xi_2)$ $\varphi_7^{bq}(\xi_1, \xi_2) = \varphi_2^q(\xi_1)\varphi_3^q(\xi_2)$ $\varphi_8^{bq}(\xi_1, \xi_2) = \varphi_1^q(\xi_1)\varphi_2^q(\xi_2)$ $\varphi_9^{bq}(\xi_1, \xi_2) = \varphi_2^q(\xi_1)\varphi_2^q(\xi_2)$

## 2.3. Storage scheme and renumbering of the nodes

As the shape functions have a local support, the stiffness matrix is sparse. Indeed, the  $K_{ij} = a(p_j, p_i)$  elements is non zero only when the intersection of the supports of the functions  $p_i$  and  $p_j$  is not empty.

With an appropriate numbering of the nodes, the matrix has a band-like structure. The bandwidth, i.e. the largest gap between a non zero element of a column and the corresponding diagonal, depends on the numbering of the nodes.

The storage of the matrix  $K$  takes those two properties into account. The program uses the so-called "*Skyline*" format in which the zero elements are stored only if they belong to the profile of the matrix, that is if they are located between a non zero element of a column and the corresponding diagonal.

The size of the profile of the matrix and its bandwidth depend on the numbering of the nodes.

This means that the number of elements that are stored depend on the numbering of the nodes. Indeed, it is important for the nodes of each element to have close numbers.

With a given numbering, the program can renumber the nodes using the Cuthill-McKee algorithm in order to reduce the size of the profile of the matrix.

The program can use quadratic or even cubic functions using the .net files originally written for linear shape functions, this means that only the three vertices of the triangle are numbered. In this case, the extra nodes (3 per triangle for quadratic functions) are numbered by the program using the numbers following those already used. In that case, it is possible to have the program renumber the nodes automatically as soon as the files are loaded. This speeds up the solving process. This option can be found in the "Parameters" menu.

### **The Cuthill McKee algorithm:**

The heuristic justification of the algorithm lies in the simple fact that neighbour nodes have to be taken into account as much as possible in the numbering process otherwise large differences in the indices of the nodes lead to large bandwidth in the matrix.

The idea is to number the nodes at the same level according to their growing degree (the degree  $d(z)$  of a node  $z$  is the number of its neighbors) so that the nodes with many neighbors must have global numbers as high as possible in order to keep low index differences in the next level.

1. Find the neighbors of the root  $r$ . These nodes have to be numbered according to their growing degree. When nodes have the same degree, an arbitrary choice is necessary. The nodes that are numbered at this step are at a distance of 1 of the root. They form the first level.
2. The nodes of the first level are considered according to their new number. For each of them, we number their non numbered neighbors by growing degree. The nodes that are numbered at this step form the second level of the renumbering process. This process is then repeated until all nodes have been renumbered.
3. It has been proven that the profile of the matrix can be reduced by inverting the order of the nodes generated by this algorithm. This is known as the Reverse Cuthill-McKee numbering.

### ***Implementation of the algorithm***

- (1) Find the initial node or “root”  $r$ . It receives the number 1.

$$S = \{r\}$$

- (2) As long as all nodes have not been renumbered:

- $S_{new} = \{\}$

- For each Number each neighbor of the node  $x$  which has not already been renumbered :  $\{k_1, k_2, \dots, k_r\}$ , the numbering is done according to their growing degree.

- $S_{new} = S_{new} + \{k_1, k_2, \dots, k_r\}$

- $S = S_{new}$

Invert the numbering using the substitution :  $k \rightarrow n+1-k$

### ***Algorithm for finding the root:***

For this step, the mesh is divided in levels.

$R(g) = (L_1, L_2, \dots, L_r)$  is called “the level tree structured with root  $g$ ”, if :

1. 1.  $\bigcup_{i=1}^r L_i$  contains all the nodes of the mesh
2.  $L_r$  is not empty
3.  $L_1 = \{g\}$
4. The shortest way between a node  $h$  of  $L_i$  and  $g$  contains  $i-1$  edges

$r$  is called the “depth” of the level structure. Its width  $k$  is the numbers of nodes of the level that contains the most nodes.

Gibbs, Pool, and Stockmeyer have suggested using level structures with the width as small as possible. This usually corresponds to the level structure with the largest depth. Indeed, as the number of levels increases, the number of nodes per level decreases.

Ideally, we should begin by finding the “diameter” of the mesh, which corresponds to the shortest way between two points of the mesh located as far as possible one from the other. The problem is, however, usually solved by using an “approximate” algorithm.

With such an algorithm,  $g$  is not always an end point of a diameter of the mesh. It is often referred to as “pseudo-diameter”.

The following algorithm is appropriate for finding a good starting node because it is not expensive in terms of computation time and of memory use, because, in general, only a few nodes have to be considered in order to find the pseudo-diameter.

- (1) Choose of a node  $g$  with minimal degree
- (2) Construct the levels of the structure  $R(g) = (L_1, L_2, \dots, L_r)$ . Sort the elements  $f_j$  of the last level  $L_r$  according to their growing degree, that is  $d(f_j) \leq d(f_{j+1})$
- (3) Set  $j = 1$
- (4) Compute the depth  $s$  of the level structure  $R(f_j)$ . If  $s > r$ , set  $g = f_j$  and return to step (2)
- (5) If  $j < l$ , add 1 to  $j$  and go back to step (4)

## 2.4. Computation of the FE system of equations

### 2.4.1. Computation and assembling of the stiffness matrix and the right-hand side vector

The computation of the stiffness matrix and the right-hand side vector is made at the element level. First of all, the program assembles the matrix and the right-hand side vector without taking the boundary conditions into account. To compute the element stiffness matrix  $K^{(r)}$ , the program performs a change of variable in order to compute integrals only on the reference elements.  $\bar{T}$  is the reference element and  $\zeta$  the coordinates on it.

On the element  $T^{(r)}$  the element stiffness matrix  $K^{(r)}$  is defined by:

$$K^{(r)} = \left[ \int_{\hat{T}} \left[ \left( \text{grad}_{\xi} \varphi_i(\xi) \right)^T \left( J^{(r)} \right)^{-1} \Lambda \left( J^{(r)} \right)^{-T} \text{grad}_{\xi} \varphi_j(\xi) \right] \left| \det J^{(r)} \right| d\xi \right]_{i,j=1}^{\hat{N}}$$

with:

-  $\hat{T}$  : reference element and  $\xi$  the coordinates on it.

-  $\varphi_i$  and  $\varphi_j$  are shape functions defined on the reference element.

-  $\Lambda = \begin{pmatrix} \lambda_{xx} & 0 \\ 0 & \lambda_{yy} \end{pmatrix}$  : the coefficients of thermal conductivity of the material.

-  $\hat{N}$  : the number of nodes per element

-  $J^{(r)}$ : the jacobian matrix of the transformation which turns the reference element  $\hat{T}$  into an element of the mesh  $T^{(r)}$ .

To compute the elements of the stiffness matrix, only the matrices  $J^{(r)}$ ,  $(J^{(r)})^{-1}$  and the values of the gradients of the shape functions defined on the reference element are needed. The explicit forms of the shape functions on the element  $T^{(r)}$  are not needed.

The element right-hand side  $f^{(r)}$  is computed using the same method as the element stiffness matrix  $K^{(r)}$ , i.e.

$$\underline{f}^{(r)} = \left[ \int_{\hat{T}} f(x_{T^{(r)}}(\xi)) \varphi_i(\xi) \left| \det J^{(r)} \right| d\xi \right]_{i=1}^{\hat{N}}$$

with:  $x_{T^{(r)}}(\xi) = J^{(r)}\xi + x_1^{(r)}$  : coordinates on the element  $T^{(r)}$

During this process, the stiffness matrix and the right-hand side are also modified to take the boundary conditions of first type into account.

We modify the right-hand size as follows :

$$f_i^{(r)} := f_i^{(r)} - \sum_{j \in \gamma_h^{(r)}} K_{ij} g_1(x_j) \quad \forall i \in \omega_h^{(r)}$$

with:

-  $g_1(x_j)$  : the value of the Dirichlet boundary condition at the point  $x_j$

-  $\omega_h^{(r)}$ : containing the numbers of the nodes of the element  $T^{(r)}$  which do not have Dirichlet boundary conditions.

-  $\gamma_h^{(r)}$ : containing the numbers of the nodes of the element  $T^{(r)}$  which have Dirichlet boundary conditions.

We also modify the stiffness matrix as:  $K_{ij}^{(r)} = K_{ji}^{(r)} = \delta_{ij}$   $\forall i \in \gamma_h^{(r)}, j \in \bar{\omega}_h^{(r)}$

with:  $\bar{\omega}_h^{(r)}$ : containing the numbers of the nodes of the element  $T^{(r)}$ .

These modifications could have also been done at the end of the assembling process, but it is easier to do them at the element level because the matrices and vectors are smaller. In addition, it is hard with the “*Skyline*” format to find, once it is fully assembled, the rows of the matrix that must be set to zero.

The element stiffness matrix and the element right-hand side are assembled using the following algorithm for every domain because two different domains may have different coefficients of thermal conductivity.

For each domain

For each element  $T^{(r)}$  of the domain

Computation of  $K^{(r)}$  and  $f^{(r)}$

Modification of  $f^{(r)}$  and  $K^{(r)}$  to take the boundary conditions of first type into account

For each node of the element, which has  $i$  as global number and  $k$  as local number.

$$f_i := f_i + f_k^{(r)}$$

For each node of the element, which has  $j$  as global number and  $l$  as local number.

$$K_{ij} := K_{ij} + K_{kl}^{(r)}$$

The next step is to take the boundary conditions into account:

### **Boundary conditions of type 2:**

The vector which must be assembled for the Neumann boundary conditions is defined by:

$$\underline{f}^{(e_2)} = \left[ \left( \int_0^1 \varphi_i(\xi) d\xi \right) g_2 \sqrt{(x_{n_2} - x_{n_1})^2 + (y_{n_2} - y_{n_1})^2} \right]_{i=1}^{\bar{N}_E}$$

with:

- $\varphi_i$  are shape functions defined on the reference interval
- $\bar{N}_E$  : is the number of nodes per edge
- $n_1$  and  $n_2$  : are the start and end nodes of the edge
- $g_2$ : is the value of the boundary condition on the edge

### **Boundary conditions of type 3:**

The vector which must be assembled for the Robin boundary conditions is defined by:

$$\underline{f}^{(e_3)} = \left[ \left( \int_0^1 \varphi_i(\xi) d\xi \right) \alpha u_A \sqrt{(x_{n_2} - x_{n_1})^2 + (y_{n_2} - y_{n_1})^2} \right]_{i=1}^{\bar{N}_E}$$

with  $\alpha$  and  $u_A$  : the values of the boundary condition on the edge

The matrix which must be assembled for the Robin boundary conditions is defined by:

$$K^{(e_3)} = \left[ \left( \int_0^1 \varphi_i(\xi) \varphi_j(\xi) d\xi \right) \alpha \sqrt{(x_{n_2} - x_{n_1})^2 + (y_{n_2} - y_{n_1})^2} \right]_{i=1}^{\bar{N}_E}$$

The program also modifies the matrix and the vector corresponding to the Robin boundary conditions to take the boundary conditions of type 1 into account. They are modified in the same way as the element stiffness matrix and the element right-hand side.

The algorithm for assembling those matrices and vectors is also the same as the one for assembling the element matrices and right-hand sides.

### **Boundary conditions of type 1:**

As the boundary conditions of first type are partially taken into account during the computation of the stiffness matrices and the right-hand sides without boundary conditions and the computation of the boundary conditions of third type, there are now only two more things to do: to set the values on the diagonal of the stiffness matrix to "1" and the entries of the right-hand size corresponding to Dirichlet nodes to the value of the boundary conditions.

For each Dirichlet node, which has  $i$  as global number,

$$K_{ii} = 1$$

$$f_i = g_I(x_i)$$

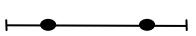
with  $g_I(x_i)$ : the value of the Dirichlet boundary condition at the point  $x_i$ .

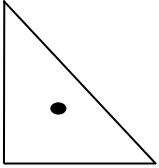
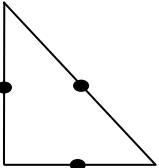
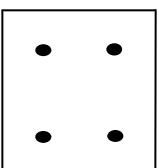
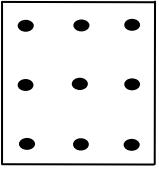
### 2.4.2. Numerical integration

The program has to compute numerous integrals and it is not always possible to get their analytic value. This is why it uses the Gauss Quadrature method to compute them. The principle of this method is to approximate the integral by a sum. The program uses the following quadrature formula:

$$\int_T w(\xi) d\xi \approx \sum_{i=1}^l \alpha_i w(\xi_i)$$

As the polynomial functions that we have to integrate are of a relatively low order, it is possible to use enough quadrature points in order to have an exact computation for the integrals:

Position of the points	Coordinates of the quadrature points $\xi_i$ or $(\xi_{i,1}, \xi_{i,2})$	Weight $\alpha_i$	Exact for polynomials of order $k$
	$\frac{3-\sqrt{3}}{6}, \frac{3+\sqrt{3}}{6}$	$\frac{1}{2}, \frac{1}{2}$	$k = 3$
	$\frac{5-\sqrt{15}}{10}, \frac{1}{2}, \frac{5+\sqrt{15}}{10}$	$\frac{5}{18}, \frac{8}{18}, \frac{5}{18}$	$k = 5$

	$\left(\frac{1}{3}, \frac{1}{3}\right)$	$\frac{1}{2}$	$k = 1$
	$\left(\frac{1}{2}, 0\right), \left(\frac{1}{2}, \frac{1}{2}\right), \left(0, \frac{1}{2}\right)$	$\frac{1}{6}, \frac{1}{6}, \frac{1}{6}$	$k = 2$
	$\left(\frac{3-\sqrt{3}}{6}, \frac{3+\sqrt{3}}{6}\right), \left(\frac{3+\sqrt{3}}{6}, \frac{3+\sqrt{3}}{6}\right)$ $\left(\frac{3-\sqrt{3}}{6}, \frac{3-\sqrt{3}}{6}\right), \left(\frac{3+\sqrt{3}}{6}, \frac{3-\sqrt{3}}{6}\right)$	$\frac{1}{4}, \frac{1}{4}$ $\frac{1}{4}, \frac{1}{4}$	$k = 3$
	$\left(\frac{5-\sqrt{15}}{10}, \frac{5+\sqrt{15}}{10}\right) \left(\frac{1}{2}, \frac{5+\sqrt{15}}{10}\right) \left(\frac{5+\sqrt{15}}{10}, \frac{5+\sqrt{15}}{10}\right)$ $\left(\frac{5-\sqrt{15}}{10}, \frac{1}{2}\right) \left(\frac{1}{2}, \frac{1}{2}\right) \left(\frac{5+\sqrt{15}}{10}, \frac{1}{2}\right)$ $\left(\frac{5-\sqrt{15}}{10}, \frac{5-\sqrt{15}}{10}\right) \left(\frac{1}{2}, \frac{5-\sqrt{15}}{10}\right) \left(\frac{5+\sqrt{15}}{10}, \frac{5-\sqrt{15}}{10}\right)$	$\frac{25}{324}, \frac{10}{81}, \frac{25}{324}$ $\frac{10}{81}, \frac{16}{81}, \frac{10}{81}$ $\frac{25}{324}, \frac{10}{81}, \frac{25}{324}$	$k = 5$

Quadrature formulas on the reference interval  $I = [0,1]$  and on the reference element

On quadrilaterals, the formulas are exact for functions which are polynomials of order  $k$  in each direction.

The numerical integration is used in various occasions:

- When we compute the element stiffness matrices and element right-hand sides. The program performs a change of variable in order to compute integrals only on the reference elements (triangle, square ...).
- When we take the boundary conditions into account. In the same way, FEJavaDemo only computes integrals on the reference interval  $I = [0,1]$ .

## 2.5. Solving process of the FE system of equations

After the whole assembling process, we end up with the following linear system of equations:

$$K u = f.$$

$K$  is the stiffness matrix,  $f$  is the right-hand side and  $u$  is the solution vector we are looking for.

In order to solve such a system given by the finite elements method, direct or iterative methods can be used. Direct methods consist of algorithms which compute the solution vector  $u$  in a certain number of steps depending only on the size of the system. The iterative methods build up a sequence  $(u_k)$  which starting from the first approximation  $u_0$  converges towards the solution vector  $u$ .

As we know that the matrix  $K$  is positive definite symmetric, we use the *Cholesky method* which is a direct method often used in such case.

The idea of the Cholesky Method is to write the matrix  $K$  as being the product of an upper triangular matrix and its transposed lower triangular matrix:

$K = S^T S$  where  $S$  is an upper triangular matrix and  $S^T$  the transposed matrix of  $S$ . During the decomposition, we also use the fact that the matrix  $K$  is a sparse matrix. This property is kept after having performed the Cholesky decomposition, so the two matrices  $K$  and  $S$  can be stored in similar areas of memory. In addition, the program can use a specific algorithm which doesn't compute the zero elements of  $S$  which are located outside the profile of the matrix:

*Decomposition algorithm ( $K = S^T S$ ) taking into account the structure of the matrix  $K$ :*

$$S_{11} = \sqrt{K_{11}}$$

Compute for  $j = 2, 3, \dots, N$

If  $l_0(j)+1 < j$ , compute for  $i = l_0(j)+1, l_0(j)+2, \dots, j - 1$  :

$$S_{ij} = \frac{1}{S_{ii}} \left( K_{ij} - \sum_{l=\max\{l_0(i), l_0(j)\}+1}^{i-1} S_{li} S_{lj} \right)$$

$$S_{jj} = \sqrt{K_{jj} - \sum_{l=l_0(j)+1}^{j-1} S_{lj}^2}$$

(  $l_0(j)$  represents the row index for which, in the  $j$ th column,  $K_{ij} = 0$  for all  $i$  satisfying  $0 < i < l_0(j)+1$  and  $K_{l_0(j)+1,j} \neq 0$  )

Solving  $Ku = f$  is then equivalent to solving  $S^T S u = f$ . This system of equations is solved in two steps using a forward substitution and a backward substitution. The program first solves  $S^T y = f$  and then  $S u = y$ . Those two equations are easy to solve because the matrices are triangular:

*Algorithm of the forward substitution:*

$$\begin{pmatrix} S_{11} & 0 & \cdots & 0 \\ S_{12} & S_{22} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ S_{1N} & S_{2N} & \cdots & S_{NN} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{pmatrix}$$

$$y_1 = \frac{f_1}{S_{11}}$$

$$y_i = \frac{1}{S_{ii}} \left( f_i - \sum_{j=1}^{i-1} S_{ji} y_j \right) \quad i = 2, 3, \dots, N$$

*Algorithm of the backward substitution:*

$$\begin{pmatrix} S_{11} & S_{12} & \cdots & S_{1N} \\ 0 & S_{22} & \cdots & S_{2N} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & S_{NN} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

$$u_N = \frac{y_N}{S_{NN}}$$

$$u_i = \frac{1}{S_{ii}} \left( y_i - \sum_{j=i+1}^N S_{ij} u_j \right) \quad i = N-1, N-2, \dots, 1$$

### 3. Structure of the data files

To be able to work with a mesh, two files have to be written: a “.net“ file and a “.dat“ file. The first file contains information about the mesh and the second about the boundary conditions and the  $f$  function.

#### 3.1. The .net file

The lines beginning with # are comments and are ignored by the program.

On the first line, the type of element is given (1 for triangles, 2 for quadrilaterals etc.).

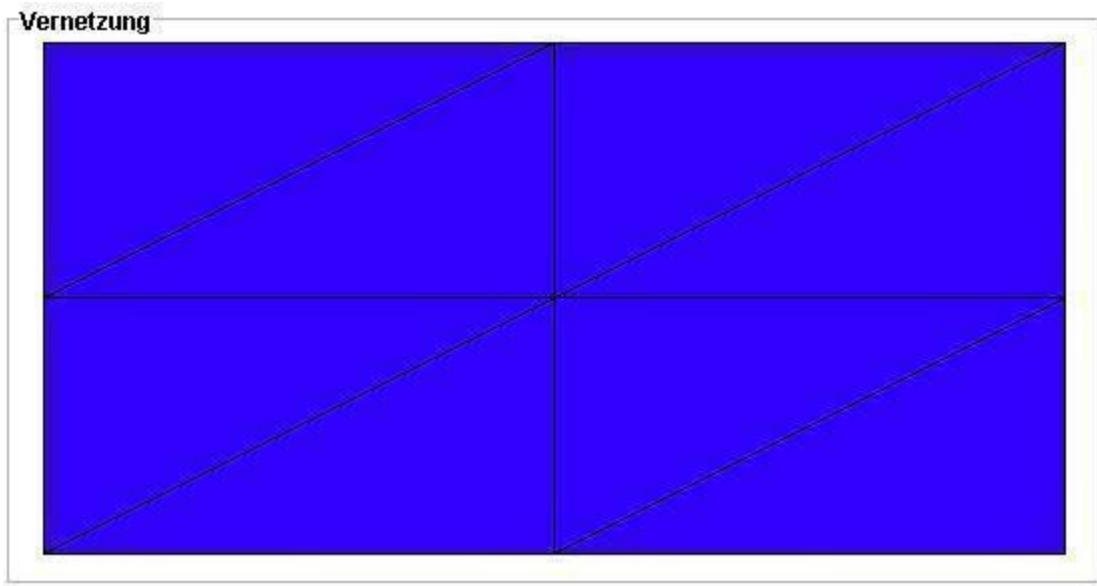
Then the number of nodes is written (i.e. the dimension of the matrix) followed by the number of elements.

For the definition of the nodes, three numbers are given: the global number of the node, the X-coordinate and the Y-coordinate.

Then the elements are defined with a global number, the global number of their vertices (three numbers for triangles, four for quadrilaterals ...) and the number of the domain to which they belong.

The next line contains the number of borders followed by the definition of those borders. They are defined with a global number and the corresponding start and end nodes.

The following example is the .net file of a simple mesh



```

# FEJavaDemo - Netfile
#
# Type of element (1 for triangles, 2 for quadrilaterals etc.)
1
#
# Number of nodes and elements
9 8
#
# Global number and coordinates of the nodes
1 0.0 0.0
2 0.0 1.25
3 0.0 2.5
4 2.5 0.0
5 2.5 1.25
6 2.5 2.5
7 5.0 0.0
8 5.0 1.25
9 5.0 2.5
#
# Global number of the nodes for each element (and number of the domain)
1 4 5 1 1
2 2 1 5 1
3 5 6 2 1
4 3 2 6 1
5 7 8 4 1
6 5 4 8 1
7 8 9 5 1
8 6 5 9 1
#
# Number of borders
8
#
# Begin and end node of the borders (and global number)
1 1 4
2 4 7
3 7 8
4 8 9
5 9 6
6 6 3
7 3 2
8 2 1

```

## 3.2. The .dat file

At the beginning of the file, the number of domains and then the coefficients of thermal conductivity  $\lambda_1$  and  $\lambda_2$  for each domain are entered.

Then the border domains are defined:

First of all the number of different border domains is written.

For each border domain, two numbers are given: the number of borders in the domain and the type of boundary conditions that the domain has (1 for Dirichlet, 2 for Neumann and 3 for Robin).

Then the values of the boundary conditions are entered with the global number of the border and the corresponding border data.

- For the Dirichlet boundary conditions, two values must be entered: the temperature of the start and end point of the border.
- For the Neumann boundary conditions, only one number has to be given, it represents the value of the heat flux on the border  $\left( \frac{\partial u}{\partial N} = a \right)$
- For the Robin boundary conditions two numbers are needed : a and b  

$$\left( \frac{\partial u}{\partial N} = a [b - u(x)] \right)$$

The last information is the definition of the  $f$  function. This function is fully written (for example  $2*x+\tan(y)+7$ ). it can be defined using the functions tan(), sin(), cos(), exp(), ln(), sqrt() and the constant  $\pi$ .

The following example is the .dat file corresponding to the first example:

```
# FEDemoJava - .dat file
#
# Number of domains
2
# Lambda1 and Lambda 2 in domain 1
200.0 200.0
# Lambda1 and Lambda 2 in domain 2
100.0 100.0
#
# Number of border domains
3
# Number of borders and type of boundary condition in border domain 1
4 1
# # Number of borders and type of boundary condition in border domain 2
2 2
# # Number of borders and type of boundary condition in border domain 3
2 3
#
# Global number and data (a (and b) coeff) for each border in domain 1
1 0. 10.
2 10. 20.
3 20. 30.
4 30. 40.
# Global number and data (a (and b) coeff) for each border in domain 2
5 5.
6 7.
# Global number and data (a (and b) coeff) for each border in domain 3
7 10. 50.
8 10. 50.
#
# Function F(x,y) in domain 1
3*cos(x*y*Pi)
# Function F(x,y) in domain 2
0
```

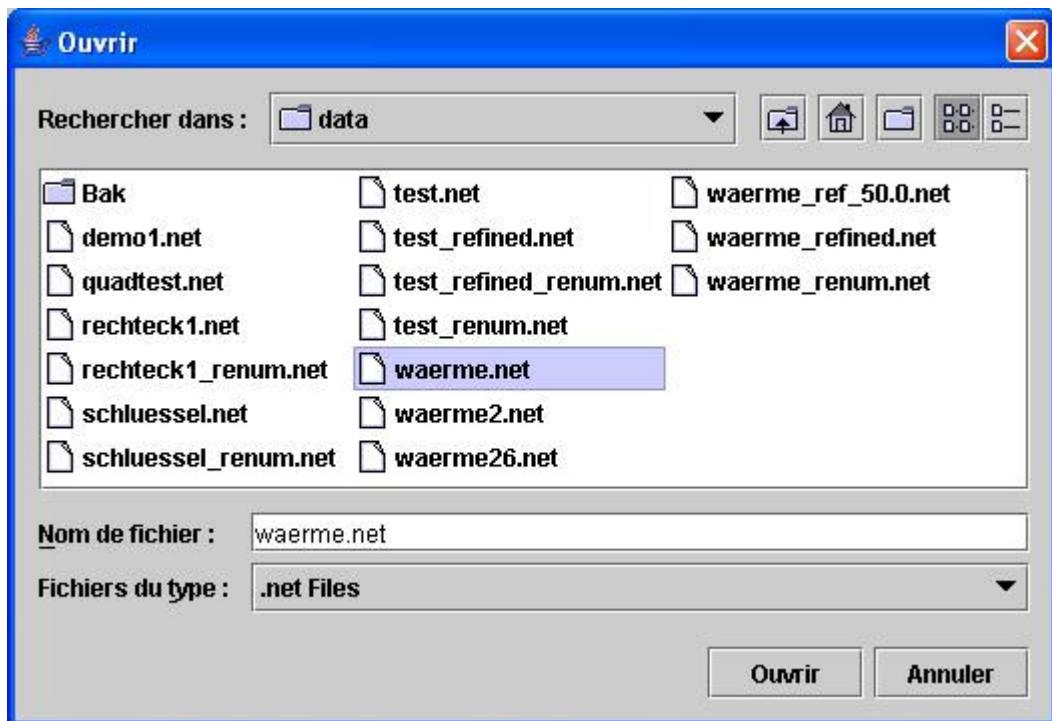
## 4. Execution of the program

### *Choice of the shape functions:*

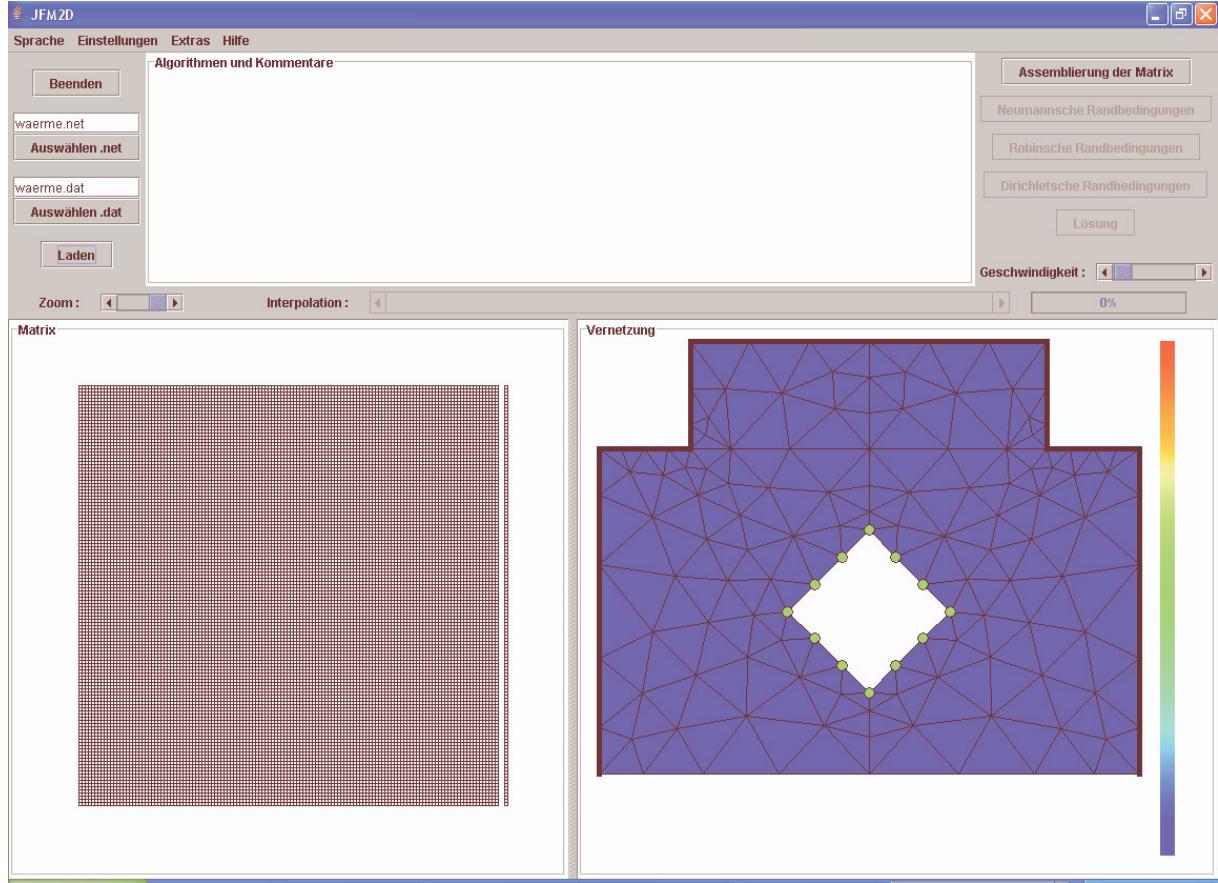
The default shape function type is linear functions, but quadratic functions can be chosen using the “Parameters → Shape-function type” menu.

### *Loading the necessary data files:*

By clicking on the “Browse .net” button (or “Browse .dat”) you will be able to choose among the available .net files (or .dat files) on the disk.



After choosing the wanted files, they can be loaded by clicking on the “Load” button.



***Computation of the stiffness matrix and right-hand side without taking the boundary conditions into account:***

After the loading process, the “Matrix Assembly” button is enabled, which, when clicked assembled the stiffness matrix and the right-hand side without taking the boundary conditions into account.

During the assembling process, the animation speed can be modified using the “Speed” scroll bar (if the scroll bar is all the way to the left, the animation is no longer displayed).

***Taking the boundary conditions into account:***

By clicking on the three next buttons, the boundary conditions are taken into account.

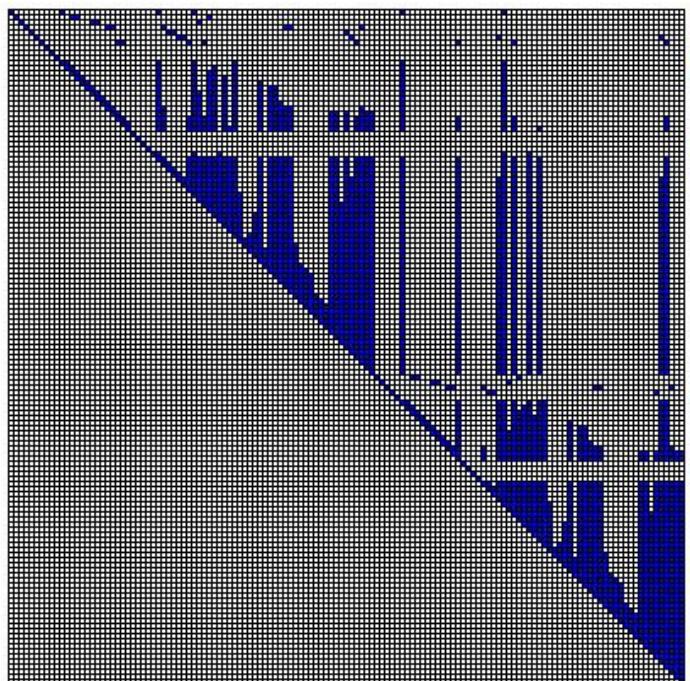
During the different assembling processes, the elements of the mesh and the entries of the matrix with which the program works are colored in red. The non zero entries are colored in blue so the user can see that the matrix is sparse. The green elements correspond to the Dirichlet nodes.

The zoom of the matrix can be changed with a scrollbar. If the zoom is big enough, the rows and column numbers are displayed, and if the zoom is all the way to the left, the values of the elements of the matrix are printed on the panel.

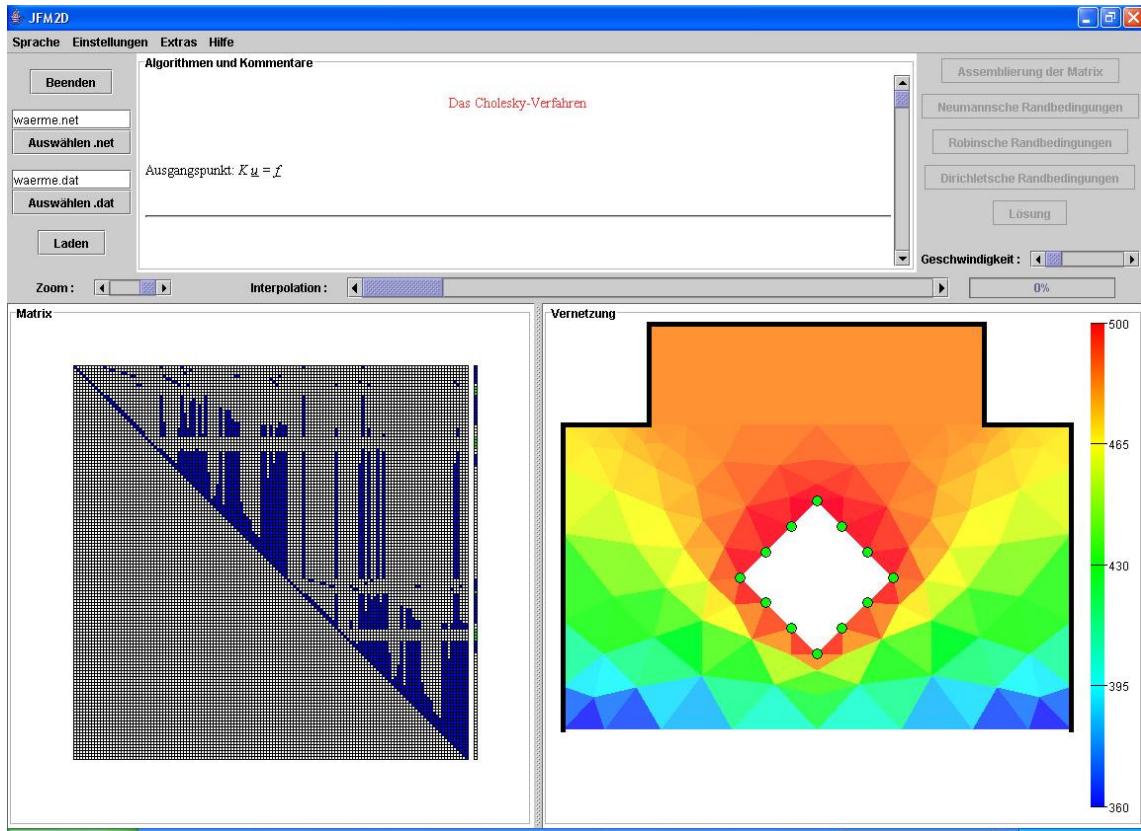
2.3	-1.5	0.0	0.0	-0.3	-0.3	
-1.5	4.4				-0.7	-0.7
0.0		1.8			-0.8	
0.0			1.8			-0.8
-0.3	-0.7	-0.8			3.8	
-0.3	-0.7		-0.8			3.8

### **Solution of the FE system of equations:**

When all the boundary conditions have been taken into account, the “Solution” button can be used to start the solving process. By selecting the “Parameters → Display → Display S’S Decomposition” menu item, the upper triangular matrix S of the  $S^T S$  decomposition is first displayed. Then an animation represents the forward and backward substitution algorithm.

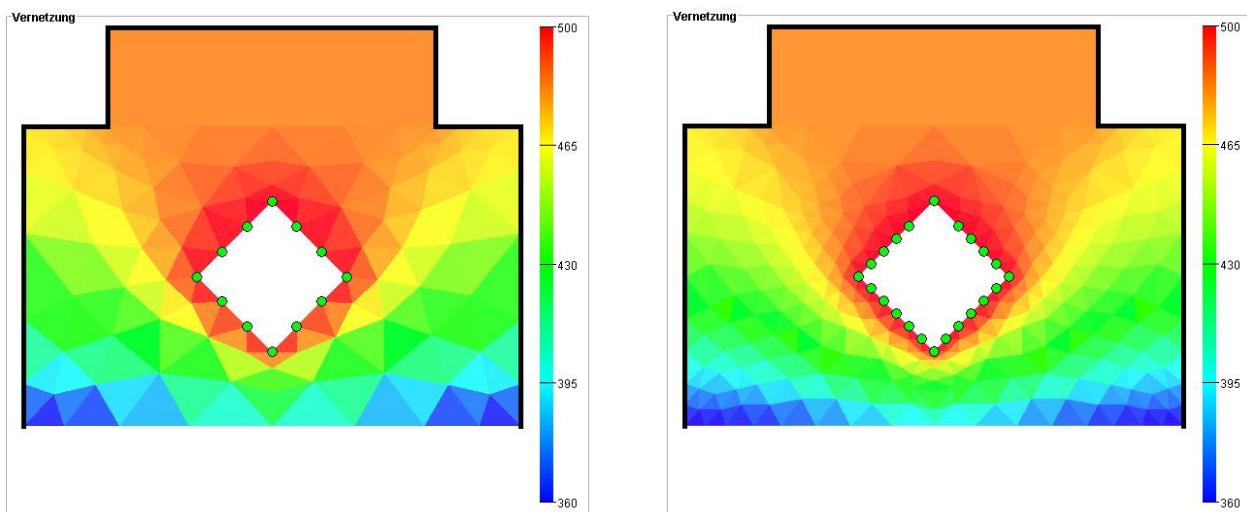


After the solving process, the temperature field is displayed along with a temperature scale.



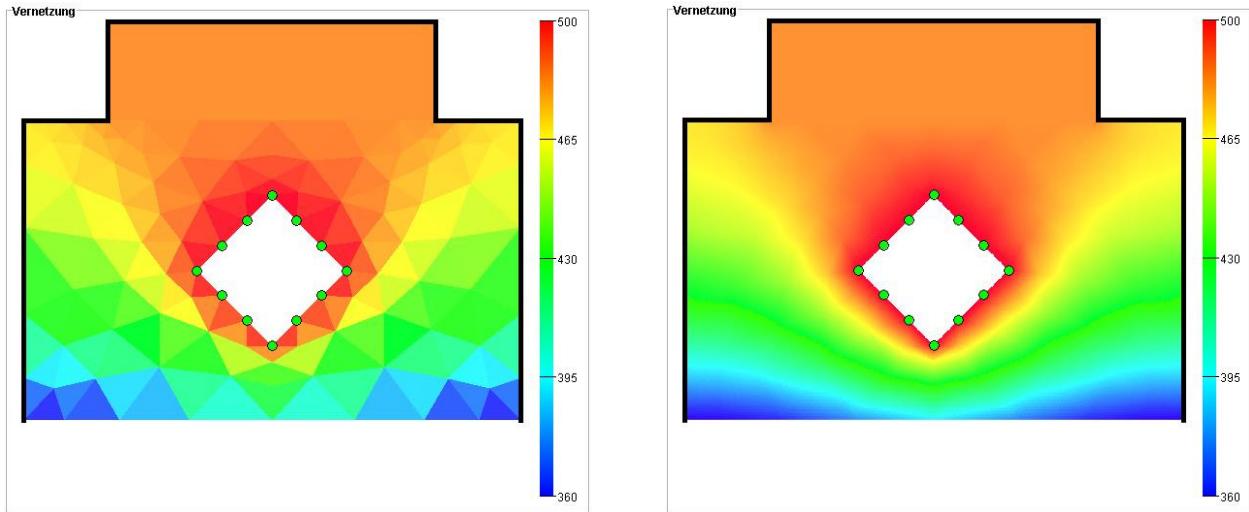
### **Mesh refinement:**

If the program works with piecewise linear shape-function, the mesh can be refined (“Accessories → Mesh Refinement”). Before starting such a refinement, the program will ask for the maximum allowed temperature variation between two consecutive nodes, it will then refine the mesh as many times as needed in order for every temperature variation to be under the specified one. Entering a maximum variation of zero will refine the entire mesh once (every triangle will be split in four). It is also possible to save the files corresponding to the refined mesh, to do so tick the corresponding option in the “Parameters → New files creation ...” menu.



### ***Interpolation of the solution:***

The representation of the solution can be enhanced using the “Interpolation” scroll bar: the triangles are cut into four triangles and the solution is computed on each new triangle using a linear interpolation. As the computation of this refinement can be a bit long a progress bar indicates the amount of work done.

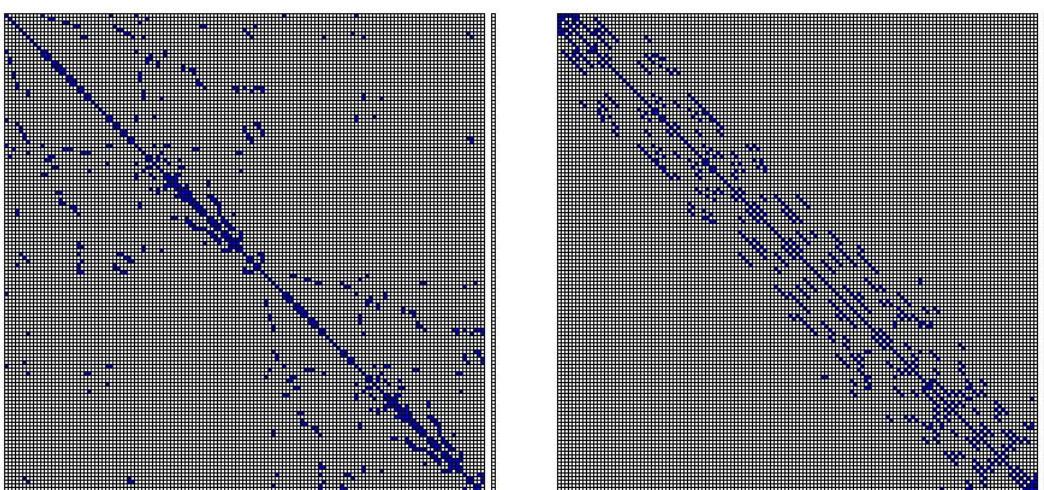


### ***Saving the solution:***

The solution (the coordinates of the nodes and their temperature) can be saved using the “Accessories → Save Solution” menu.

### ***Renumbering of the nodes:***

The nodes can also be renumbered with the “Accessories → Renumbering” menu (see Renumbering). The menu item “Auto-renumbering of nodes with quadratic functions” is selected by default. It enables the solving process to run faster. It is also possible to save the files corresponding to the renumbered mesh, to do so tick the corresponding option in the “Parameters → New files creation ...” menu.



## **Other possibilities:**

The “Language” menu can be used at any time to change the language.

During the assembling or the solving process, it is possible to save Jpeg images of the mesh and the matrix by using the “Accessories → Create Jpeg” menu.

The “Parameters → Display” menu has two more options: “Display boundary conditions” and “Display the node numbers”, these options turn on or off the displaying of the Dirichlet nodes (colored in green) and the highlighting of the borders isolating the mesh from the exterior, and also of the numbers of the nodes of the mesh.

