Faculty of Electrical Eng, Mathematics and Computer Science

Users Manual 2015–2016
WI4243FEM

Finite-element analysis for Applied Physics

Fred Vermolen

Delft Institute of Applied Mathematics

Delft University of Technology

Contents

1 Introduction and file inventory

This documentation treats how to do the lab work that is associated with the course wi4243FEM for applied physics master students at the Delft University of Technology. The lab work is done in Matlab. The following files are needed in your working folder:

1. WI4243Mesh.m: This file is used for the mesh and topology (distribution of elements and vertices) generation;
2. WI4243Comp.m: This file is used to actually compute the solution;
3. WI4243Post.m: This file is used for the postprocessing: the solution is plotted in terms of a 3D-plot and a contour plot;
4. BuildMatricesandVectors.m: This file assembles the large discretisation matrix and right-hand side vector;
5. GenerateElementVector.m: *This file should be adapted by the participant.* It is used to compute the element vector for internal elements.
6. GenerateElementMatrix.m: *This file should be adapted by the participant.* It is used to compute the element matrix for internal elements.
7. GenerateBoundaryElementVector.m *This file should be adapted by the participant.* It is used to compute the element vector for boundary elements.
8. GenerateBoundaryElementMatrix.m: *This file should be adapted by the participant.* It is used to compute the element matrix for boundary elements.

2 How to run a finite-element simulation

To run the files, the user first needs to start MATLAB. To perform a simulation, type:

WI4243Mesh

WI4243Comp

WI4243Post

These commands, respectively, construct the finite-element mesh, perform the computation and show the solution. The default problem that is solved is given by:

$$\begin{cases} -\nabla \cdot (D\nabla u) = 1, & (x,y) \in \Omega, \\ \\ D\dfrac{\partial u}{\partial n} + ku = 1, & (x,y) \in \partial\Omega. \end{cases} \tag{1}$$

Here $\Omega = \{(x,y) \in \mathbb{R}^2 | x^2 + y^2 < 1\}$, and $\partial\Omega$ represents the boundary of Ω. Further, D and k, respectively, represent the diffusion coefficient and the transfer coefficient. We will go through the files sequentially in the next section.

3 Description of the files

This section informs the participant about the files that are needed for the finite-element simulation. We note that the code that is available could be used for very advanced finite-element solutions with time-dependency, coupled systems, or even with interfaces. The present lab work only aims at making

the applied physics familiar with the finite-element method. The enthusiastic and interested student will have a matlab basis tool for solving more advanced finite-element problems since the code can be extended easily to handle time-dependent problem, coupled problems (such as mechanical problems) in terms of systems of partial differential equations, as well as more elaborate elements such as quadratic, bilinear etc. Higher-order methods will require a revision in the mesh-generation.

3.1 The Mesh file: WI4243Mesh.m

In the file **WI4243Mesh**, the user specifies the physical parameters such as diffusion coefficients etc on the top of the file. Further, the user is allowed to specify the geometry here. In this course, we will limit ourselves to some simple geometries, such as a square and a circle. The square, circle (or L-shape) can be set by adjusting the line

Geometry = 'circleg';

Geometry = 'squareg';

Geometry = 'lshapeg';

The square has dimensions $(x, y) \in (-1, 1) \times (-1, 1)$. Here the matlab command 'initmesh' is used, of which the most important output is the set of nodal points, triangular elements with a topology, which is a list of the vertices (nodal points) for each (triangular) element. The command 'refine mesh' is used to refine the mesh. In principle all elements are divided into four triangles, hence, roughly speaking the diameter of each triangle is halved. This step can be applied several times to determine the mesh-sensitivity of the numerical approximation of the solution. The command 'pdemesh' is used to plot the mesh. The most important output parameters are

- x: the list of the x-coordinates of the nodal points/vertices;
- y: the list of the y-coordinates of the nodal points/vertices;
- n: the number of all (internally and on the boundary) nodal points;
- elmat: the list with the indices of the vertices of each internal (triangular) element, elmat(i,j) gives the index of the j-th node of element i;
- elmatbnd: the list with the indices of the vertices of each boundary (line-segment) element, elmatbnd(i,j) gives the index of the j-th node of boundary element i; %item f: the solution vector is initialised by zeros with length n.

3.2 The Computation File: WI4243Comp.m

In this file, the solution is computed. Input parameters are the physical parameters, the geometry, mesh and topology of the domain. The large discretisation matrix S, the large right-hand side vector f and the solution-vector u are the output of this routine. The file calls the assembly procedure **BuildMatricesandVectors** to build the large discretisation matrix S and large right-hand side vector f. Matlab uses a direct, decomposition method to solve the large system of linear equations

$$S\underline{u} = \underline{f}. \tag{2}$$

3.3 Building the linear problem: BuildMatricesandVectors.m

This files is called by WI4243Comp.m and it is responsible for building the large discretisation matrix S and large right-hand side vector f that are used to solve the finite-element problem. The file treats each element (both internally and on the boundary) sequentially and uses the element matrix and element vector of each internal and boundary element. Let n_{el} represent the number of internal elements, then regarding the internal elements, the sequence is given by

$S = \emptyset$ Initially S is a matrix containing zeros only
$For\ i\ =\ 1\ :\ n_{el}$
  $GenerateElementMatrix$;
  $For\ ind1\ =\ 1\ :\ 3$
    $For\ ind2\ =\ 1\ :\ 3$
      $S(elmat(i, ind1), elmat(i, ind2))\ =\ S(elmat(i, ind1), elmat(i, ind2))\ +\ Selem(ind1, ind2)$;
    end;
  end;
end;

For the right-hand side vector, and the boundary matrix and vector, the structure is analogous. Contributions from internal and boundary elements are just summed to get the large, global discretisation matrix and large, global right-hand side vector. The student does not to modify this program unless the student works on Assignment 7. The modification there is just the insertion of a couple of other matrices that need to assembled.

3.4 Construction of the internal element matrix: GenerateElementMatrix.m

This file is called in the routine BuildMatricesandVectors.m, and it is responsible for the computation of the element matrix. This file needs to be adjusted in some of the assignments. The file starts with getting the positions of the nodal point of the element of consideration (with index i). The local coordinates are given by the local lists xc and yc. The basis-functions are obtained in terms of α, β and γ. Further the absolute value of $Delta$ represents twice the area The output is the internal element matrix $Selem$, which is used in the program BuildMatricesandVectors.m.

3.5 Construction of the internal element vector: GenerateElementVector.m

This file is called in the routine BuildMatricesandVectors.m, and it is responsible for the computation of the element vector. This file needs to be adjusted in some of the assignments. The file starts with getting the positions of the nodal point of the element of consideration (with index i). The local coordinates are given by the local lists xc and yc. The basis-functions are obtained in terms of α, β and γ. The output is the internal element vector $felem$, which is used in the program BuildMatricesandVectors.m.

3.6 Construction of the boundary element matrix: GenerateBoundaryElementMatrix.m

This file is called in the routine BuildMatricesandVectors.m, and it is responsible for the computation of the boundary element matrix. This file needs to be adjusted in some of the assignments. The file starts with getting the positions of

the nodal point of the element of consideration (with index i). The local coordinates are given by the local lists xc and yc. The output is the internal element matrix $BMelem$, which is used in the program BuildMatricesandVectors.m.

3.7 Construction of the boundary element vector: GenerateBoundaryElementVector.m

This file is called in the routine BuildMatricesandVectors.m, and it is responsible for the computation of the element vector. This file needs to be adjusted in some of the assignments. The file starts with getting the positions of the nodal point of the element of consideration (with index i). The local coordinates are given by the local lists xc and yc. The basis-functions are obtained in terms of α, β and γ. The output is the internal element vector $bfelem$, which is used in the program BuildMatricesandVectors.m.

3.8 Postprocessing: WI4243Post

The input parameters are the coordinates of the nodes, x and y, and the solution vector u. The file generates a three-dimensional plot of the solution over space, as well as a coloured contour plot of the solution.

4 Example

As mentioned earlier, we consider the example

$$\begin{cases} -\nabla \cdot (D\nabla u) = 1, & (x,y) \in \Omega, \\ \\ D\dfrac{\partial u}{\partial n} + ku = 1, & (x,y) \in \partial\Omega. \end{cases} \tag{3}$$

Here $\Omega = \{(x,y) \in \mathbb{R}^2 | x^2 + y^2 < 1\}$, and $\partial\Omega$ represents the boundary of Ω. Further, D and k, respectively, represent the diffusion coefficient and the transfer coefficient. This example models a steady-state equilibrium of a diffusion process with an internal source, and transfer over the boundary. Running (by typing)

WI4243Mesh

gives a circular mesh as shown in Figure 1. To obtain a finer mesh (higher

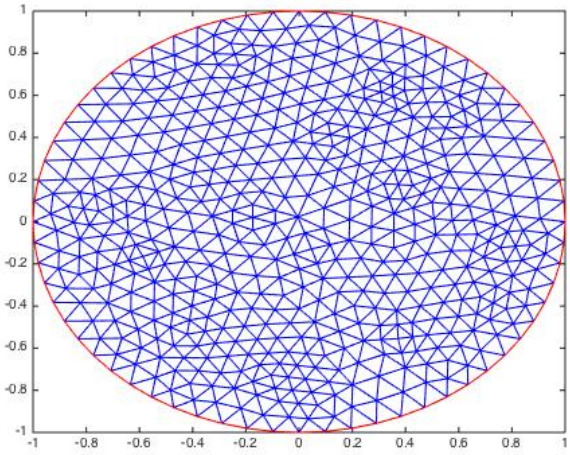

Figure 1: A finite-element mesh of a circle.

resolution), one can refine, see Section 3.1. Subsequently, one runs

WI4243Comp

To get the solution. Subsequently, running

WI4243Post

post processes the solution and one obtains a three-dimensional surface plot of the solution, as well as a coloured contour plot, see Figures 2 and 3.

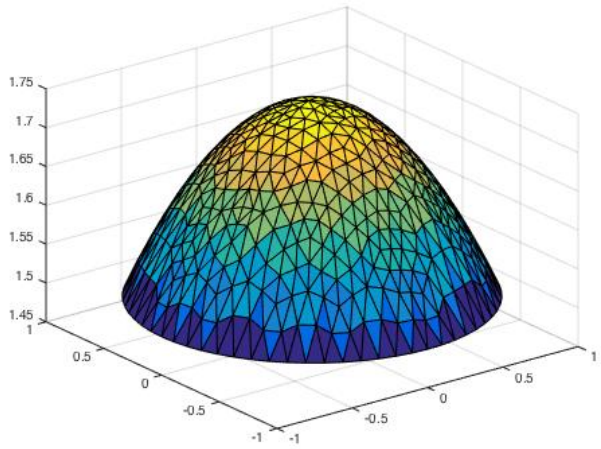

Figure 2: A finite-element solution represented by a surface plot.

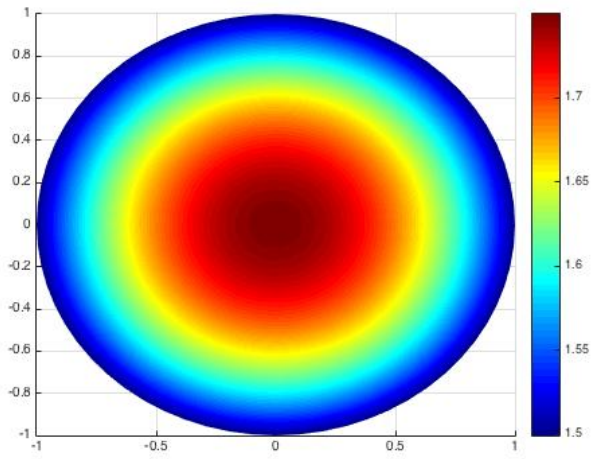

Figure 3: A finite-element solution represented by a coloured contour plot.

5 Treatment of Point Sources

In some of the assignments, one considers point sources. This approach is physically sound of the source or sink that, respectively, emits or consumes a certain quantity is much smaller than the domain of computation. Furthermore, many Green's Functions are based on point sources (or pulses in the initial

conditions in initial-boundary value problems). An example is given in the following problem

$$\begin{cases} -\Delta u + \lambda u = \delta(\mathbf{x}), & (x,y) \in \Omega, \\ \\ \dfrac{\partial u}{\partial n} = 0, & (x,y) \in \partial\Omega. \end{cases} \tag{4}$$

Here the source function is modeled by a Dirac (Dirac was a physicist by the way) Delta Distribution, which is characterised by

$$\begin{cases} \delta(\mathbf{x}) = 0, & \text{if } \mathbf{x} \neq \mathbf{0}, \\ \\ \displaystyle\int_{\Omega} \delta(\mathbf{x}) d\Omega = 1, & \text{where the origin } (0,0) \text{ is contained in } \Omega. \end{cases} \tag{5}$$

In many of the other discretisation methods, such as finite-differences, it is necessary to *regularise*, that is, smoothen, the source function. A possibility is to approximate the Dirac Delta function by a multivariate (2D) normal distribution with a 'sufficiently' small standard deviation. In finite-element strategies, this smoothing is not needed. The weak form of problem (4) is given by

$$\text{Find } u \in H^1(\Omega) \text{ such that } \int_{\Omega} \nabla u \cdot \nabla \phi + u\phi d\Omega = \phi(0,0), \text{ for all } \phi \in H^1(\Omega). \tag{6}$$

Here the integration property of the Dirac Delta Function has been used:

$$\int_{\Omega} \phi(x,y)\delta(x,y) d\Omega = \phi(0,0).$$

The element matrix can be obtained using the standard methods that have been presented in the lectures and in the book. For element e, the element vector becomes

$$f_i^e = \begin{cases} \phi_i(0,0), & \text{if } (0,0) \in e, \\ \\ 0, & \text{if } (0,0) \notin e. \end{cases} \tag{7}$$

It is needless to mention that this principle can be extended for multiple point sources located at different positions. To evaluate this element vector, it is necessary to determine whether the point source is located within the element e. In the following text, we use the convention $\mathbf{x} = (x, y)$ to denote the coordinates. Suppose that element e has vertices $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$, and let $\Delta(\mathbf{a}, \mathbf{b}, \mathbf{c})$ denote the triangle with general vertices $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$, further $|Q|$ denotes the area occupied by Q. Then the following easy-to-check geometrical criterion to check whether a point $\mathbf{a}$ is located in $\Delta(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ is the following

$$|\Delta(\mathbf{a}, \mathbf{x}_2, \mathbf{x}_3)| + |\Delta(\mathbf{x}_1, \mathbf{a}, \mathbf{x}_3)| + |\Delta(\mathbf{x}_1, \mathbf{x}_2, \mathbf{a})| = |\Delta(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)|. \tag{8}$$

We denote $\overline{e}$ as the extension of e with its boundary. The above criterion is violated if and only if $\mathbf{a} \notin \overline{e} = \overline{\Delta(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)}$ (that is, the point $\mathbf{a}$ is not located in the triangle and not on its boundary). In case of violation, *equality* is replaced with *larger than* in the above equation. The area of the triangle $\Delta(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$, is computed by

$$|\Delta(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)| = \frac{1}{2}(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1) = \frac{1}{2}\det \begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{pmatrix}. \tag{9}$$

It is convenient to use the determinant in your code. Regarding the implementation in matlab or any computer code, one has to aware of *rounding errors*. To this extent, in matlab, one should replace the above criterion with

$$|\Delta(\mathbf{a},\mathbf{x}_2,\mathbf{x}_3)|+|\Delta(\mathbf{x}_1,\mathbf{a},\mathbf{x}_3)|+|\Delta(\mathbf{x}_1,\mathbf{x}_2,\mathbf{a})|<|\Delta(\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3)|+eps, \quad (10)$$

where *eps* is a standard matlab variable that you can use. The variable *eps* indicates a measure for rounding errors. If there are multiple point sources in an element e then the element matrix consists of the sum over the locations. An example, suppose that $\mathbf{a}_1$ and $\mathbf{a}_2$ are located in triangular element e, then its element vector becomes

$$f_i^e=\phi(\mathbf{a}_1)+\phi(\mathbf{a}_2), \quad (11)$$

where the convention $\mathbf{x}=(x,y)$ has been used.

As an alternative to the determination whether a certain point $\mathbf{a}$ is located within the triangle with vertices $\mathbf{x}_1$, $\mathbf{x}_3$ and $\mathbf{x}_3$, one can make use of the so–called barycentric coordinates, which actually comes down to the use of the linear finite–element basis functions that *solely* live in $\Delta(\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3)$, in the textbook (Segal *et al.*), these functions are referred to as $\lambda_1(\mathbf{x})$, $\lambda_2(\mathbf{x})$ and $\lambda_3(\mathbf{x})$. One should test whether the values at the point $\mathbf{a}$ range between zero and one for the three λ–functions.

6 Final Remarks

After having done the lab work, the student has a basic finite-element tool in matlab that can be extended to solve more complicated problems, such as time-evolving problems, coupled systems (such as mechanical balances) of partial differential equations. Even more complicated elements can be dealt with (such as quadratic, bilinear (quads)). Furthermore, it is relatively straightforward to extend the code to $\mathbb{R}^3$. Note that in some cases, the topology and the mesh has to be adjusted. More advanced matlab-based mesh generators are available in distmesh for instance. It is noted that the routine has not been optimised such that the discretisation matrices have a small bandwidth. One needs a Cuthill-McKee procedure to improve the bandwidth. This issue becomes important for large ($\mathbb{R}^3$) problems. Further it is noted that the Dirichlet conditions needs another treatment, based on elimination of rows and columns of the large discretisation matrix and on removing entries of the large right-hand side vector, as can be read in the course book. An alternative straightforward to deal with Dirichlet (essential) boundary conditions, is using a penalisation approach by a mixed boundary condition, in setting $k\rightarrow\infty$ (or a very large value) in the boundary condition of equation (1).