Finite Elements

Rick Koster
Ruben Termaat

February 22, 2018

Contents

Chapter 1

1D-case

On the 1D interval of $x = [0, 1]$, we consider a steady-state convection-diffusion-reaction equation, with homogeneous Neumann boundary conditions. The following equations apply to this domain:

$$\begin{cases} -D\triangle u + \lambda u = f(x), \\ -D\frac{du}{dx}(0) = 0, \\ -D\frac{du}{dx}(1) = 0 \end{cases} \tag{1.1}$$

Here $\triangle$ equals the $\nabla \cdot \nabla$ operator. The function f(x) is a given funtion, where D and ∇ are positive real constants. In order to solve this boundary value problem (BVP), first the interval is divided in n-1 elements(n = positive integer). This results in the domain being divided in elements: $e_i = [x_i, x_{i-1}]$ where $i = 1, 2, .., n$.

In order to solve this BVP, the solutions for the given equations will first be calculated and then computed using MATLAB codes.

1.1 Boundary value problem 1D

In order to find the Weakform of the given equations of (1.1), we first multiply both sides by ϕ and integrate both sides over the domain ω.

$$\int_\Omega \phi(-D\triangle u + \lambda u)d\Omega = \int_\Omega \phi f(x)d\Omega \tag{1.2}$$

Now by rewriting and then using partial integration the following equation can be found:

$$\int_\Omega (\nabla \cdot (-D\phi\nabla u) + D\nabla\phi\nabla u + \phi\lambda u)d\Omega = \int_\Omega \phi f(x)d\Omega \tag{1.3}$$

Applying Gauss on the first term on the left side of equation(1.3):

$$\int_\Omega \vec{n} \cdot (-D\phi\nabla u)d\tau + \int_\Omega (D\nabla\phi \cdot \nabla u + \phi\lambda u)d\Omega = \int_\Omega \phi f(x)d\Omega \tag{1.4}$$

Using the boundary conditions from equations(1.1) the boundary integral equals to 0 and then the following weak formulation(WF) is found:

(WF):

$$\begin{cases} \text{find u } \epsilon \sum = \{u \ smooth\} \text{ Such that:} \\ \int_\Omega (D\nabla\phi \cdot \nabla u + \phi\lambda u)d\Omega = \int_\Omega \phi f(x)d\Omega \\ \forall\phi \ \epsilon \sum \end{cases} \tag{1.5}$$

The next step is to apply the Galerkin equations to the found diferential equation, where u is replaced by $\sum_{j=1}^{n} c_i\phi_j$ and $\phi(x) = \phi_i(x)$ with $i = [1, .., n]$. Filling this in equation (1.5) the following equation is found:

$$\sum_{j=1}^{n} c_i \int_0^1 (D\nabla\phi_i \cdot \nabla\phi_j + \lambda\phi_i\phi_j)d\Omega = \int_0^1 \phi_i f(x)d\Omega \tag{1.6}$$

Which is of the form of $S\vec{c} = \vec{f}$

1.2 Element matrix

Now the found Galerkin equations can be used to compute S_{ij} the element matrix, over a generic line element e_i.

$$S\vec{c} = \sum_{j=1}^{n} c_i \int_0^1 (D\nabla\phi_i \cdot \nabla\phi_j + \lambda\phi_i\phi_j)d\Omega \tag{1.7}$$

Now to solve S we solve the following equation, over the internal line element.

$$S_{ij}^{e_i} = -D\int_{e_k} \nabla\phi_i \cdot \nabla\phi_j d\Omega + \lambda \int_{e_k} \phi_i\phi_j dx \tag{1.8}$$

1.3 Element vector

Again using the found Galerkin Equations(1.6) in order to compute the element vector f_i over a generic line-element.

$$f_i^{e_k} = \int_{e_k} \phi_i f dx \tag{1.9}$$

$$f_i^{e_k} = \frac{|x_k - x_{k-1}|}{(1+1+0)!} f(\vec{x}) = \frac{|x_k - x_{k-1}|}{2} \begin{bmatrix} f_{k-1}^{e_n} \\ f_k^{e_n} \end{bmatrix} \tag{1.10}$$

1.4 Boundary value problem 1D MATLAB routine

1.4.1 mesh and elmat code

The first step in order to solve the BVP is to write a MATLAB routine that generates an equidistant distrubition of points over the given interval of $[0, 1]$(generate a mesh with n-1 elements).

```
function [ x ] = GenerateMesh(int, N_elem)
%GenerateMesh Creates a mesh for 1D problems


% int = [0,1];
% N_elem = 100;

x = linspace(int(1,1),int(1,2),N_elem);
```

Using the codes to generate a mesh and the elmat, it is easier to use this 1D problem and adapt to a higher dimensional problem. The next step is to write a code that generates a two dimensional array, called the elmat.

```
function [ elmat ] = GenerateTopology( N_elem )
%GenerateTopology Creates the topology for a 1D problem given mesh 'x'.

% global N_elem
elmat = zeros(N_elem,2);
elmat(i,1) = i;
elmat(i,2) = i + 1;
```

1.4.2 Element matrix

Now that the base MATLAB codes are made the element matrix and element vector codes can be written. The first step in this process is, is to compute the element matrix S_{elem}.

```
function [ Selem ] = GenerateElementMatrix( k, elmat, int, N_elem, D, lambda)

% global D
% global lambda
% global N_elem

Selem = zeros(2,2);

i = elmat(k,1);
j = elmat(k,2);

mesh = GenerateMesh(int, N_elem);

x1 = mesh(i);
x2 = mesh(j);

element_length = abs(x1-x2);
```

```
slope = 1/element_length;

% MUCH FASTER for some reason

Selem(1,1) = element_length*((-1)^(abs(1-1))*D*slope^2 + (1+1)*lambda/6);
Selem(1,2) = element_length*((-1)^(abs(1-2))*D*slope^2 + (1)*lambda/6);
Selem(2,1) = element_length*((-1)^(abs(2-1))*D*slope^2 + (1)*lambda/6);
Selem(2,2) = element_length*((-1)^(abs(2-2))*D*slope^2 + (1+1)*lambda/6);

% for m = 1:2
%     for n = 1:2
%         M = sym(m);
%         N = sym(n);
%         Selem(m,n) = element_length*((-1)^(abs(m-n))*D*slope^2 +
%(1+kroneckerDelta(M,N))*lambda/6);
%     end
% end

end
```

To generate a n-by-n matrix S, the sum over the connections of the vertices in each element matrix, over all elements has to be calculated. The following code computes this matrix.

1.4.3 Assemble matrix S

```
function [ S ] = AssembleMatrix( N_elem, int, lambda, D)
% global N_elem

elmat = GenerateTopology(N_elem);

S = zeros(N_elem,N_elem);

for i = 1:N_elem-1
        Selem = GenerateElementMatrix(i, elmat, int, N_elem, D, lambda);
        for j = 1:2
                for k = 1:2
                        S(elmat(i,j), elmat(i,k)) =
                        S(elmat(i,j), elmat(i,k)) +     Selem(j,k);
                end
        end
end
```

All the previous code will generate a large matrix S, from the element matrices S_{elem} over each element.

1.4.4 Element vector MATLAB routine

The next step In order to solve the equation $S\vec{c} = F$ is to create a code to generate the element vector. This element vector provides information about note i and node $i+1$, which are the vertices of element e_i.

```
function [ felem ] = GenerateElementVector( i, elmat, int, N_elem )


felem = [0;0];

% k_i = [elmat(i,1), elmat(i,2)];
%
% x_i = mesh(k_i);

k1 = elmat(i,1);
k2 = elmat(i,2);

mesh = GenerateMesh(int, N_elem);

x1 = mesh(k1);
x2 = mesh(k2);



element_length = abs(x1-x2);
```

```
felem = (element_length/2*arrayfun(@functionBVP,[x1,x2]))';
```

To generate the vector f, the sum over the connections of the vertices in each element matrix, over all elements $i \epsilon \{1, ..., n-1\}$ has to be calculated.

```
function [ f ] = AssembleVector( N_elem, int, lambda, D )

f = zeros(N_elem,1);
elmat = GenerateTopology(N_elem);

for i = 1:N_elem-1
        felem = GenerateElementVector(i, elmat, int, N_elem);
        for j = 1:2
                f(elmat(i,j)) = f(elmat(i,j)) + felem(j);
        end
end
```

1.4.5 Computing S and f

Now if the previously shown matlab codes are run the following happens. Firstly a mesh and 1D topology is build, which is needed for the S matrix and f vector. The second step is to calculate the S matrix and f vector through the found equations of section 1.2 and 1.3. The final step is to use the found matrix and vector to solve the equation $Su = \vec{f}$.

1.5 Main program

The main program is simple written by assembling the previous created matlab code AssembleMatrix and AssembleVector and deviding the vector f by the matrix S.

```
function [ u ] = SolveBVP(  N_elem, int, lambda, D )

S = AssembleMatrix( N_elem, int, lambda, D);
f = AssembleVector( N_elem, int, lambda, D);

%% Calculate u
x = linspace(int(1),int(2),100);

u = S\f;
plot(x,u);
```

1.6 Solution for u

The final step is to use the main program to solve $u = S\ \vec{f}$. Previously the S matrix and f vector were computed for $n = 100$. Now u will be calculated for $f(x) = 1$, $D = 1$, $\nabla = 1$ and $n = 100$. The result of this is plotted in figure(1.1).

1.7 Experiment

The next step is to see what happens when changing f(x) to $f(x) = sin(20x)$ and to see the difference for several values for n ($n = 10, 20, 30, 40, 80, 160$)

```
function [f] = functionBVP(x)
% f = sin(20*x);

% f = x;
f = 1;

end
```

Chapter 2

2D-case

We consider a square reservoir (a porous medium) with several wells where water is extracted. This is an important application in countries like Bangladesh where fresh water is extracted from the subsurface. Far away from the reservoir, the water pressure is equal to the hydrostatic pressure. Since we are not able to consider an in

nite domain, we use a mixed boundary condition which models the transfer of water over the boundary to locations far away. To this extent, we consider a square domain with length 2 in meter, that is $\Omega = (-1;1)x(-1;1)$ with its boundary $d\Omega$. In this assignment, we consider a steady-state equilibrium determined by Darcy's Law for fluid velocity, given by

$$\vec{v} = -\frac{k}{\mu}\nabla p \tag{2.1}$$

where p, k, μ and v, respectively denote the fluid pressure, permeability of the porous medium, viscosity of water and the fluid flow velocity. Since we only consider a plane section of the reservoir in this assignment, the effect of gravity is not important. Next to Darcy's Law, we consider incompressibility, where the extraction wells are treated as point sinks (this assumption can be justified by the fact that the well diameter is much smaller than the dimensions of the porous medium), that extract at the same rate in each direction, hence

$$\nabla \cdot \vec{v} = -\sum_{p=1}^{n_{well}} Q_p \delta(x - x_p) = 0, (x, y) \epsilon \Omega \tag{2.2}$$

where Q_p denotes the water extraction rate by well k, which is located at xp. We use the convention x = (x; y) to represent the spatial coordinates. Further, represents the Dirac Delta Distribution, which is characterized by

$$\begin{cases} \delta(x) = 0, x \neq 0 \\ \int_\Omega \delta(x) d\Omega = 1, \text{ where } \Omega \text{ contains the origin.} \end{cases} \tag{2.3}$$

Next to the above partial differential equation, we consider the boundary condition

$$\vec{v} \cdot \vec{n} = K(p - p^H),\ (x, y)\, \epsilon\, \partial\Omega \tag{2.4}$$

Here K denotes the transfer rate coefficient of the hormon between the boundary of the domain and its surroundings, and pH represents the hydrostatic pressure. For the computations, we use the following values:

Table 2.1: Values of input parameters

Symbol	Value	Unit
Q_p	50	m^2/s
k	10^{-7}	m^2
μ	$1.002 \cdot 10^{-3}$	$Pa \cdot s$
K	10	m/s
p^H	10^6	Pa

We consider six wells, located at:

$$\begin{cases} x_p = 0.6\cos(\frac{2\pi(p-1)}{5}) \\ x_p = 0.6\sin(\frac{2\pi(p-1)}{5}) \end{cases} \tag{2.5}$$

2.1 Boundary value problem 2D

The first step to solving these equations using finite elements is to find to find the boundary value problem to solve. This is done by filling in equation(2.1) in both equation 2.2 and the boundary condition(2.4) in order to find the BVP in terms of p:

BVP:

$$\begin{cases} -\frac{k}{\mu}\triangle \cdot \vec{p} = -\sum_{p=1}^{n_{well}} Q_p \delta(x - x_p) = 0, \ (x, y)\, \epsilon\, \Omega \\ -\frac{k}{\mu}\nabla\vec{p} \cdot \vec{n} = -\frac{k}{\mu}\frac{dp}{dn} = K(p - p^H), \ (x, y)\, \epsilon\, \partial\Omega \end{cases} \tag{2.6}$$

The next step is to compute the weak formulation using the previous found BVP(2.6). By multiplying both sides by $phi(x)$ and integrating both sides over the domain Ω the weak formulation can be found.

$$\int_\Omega \phi(x)\nabla \cdot (-\frac{k}{\mu}\nabla\vec{p})d\Omega = \int_\Omega -\sum_{p=1}^{n_{well}} \phi Q_p \delta(x - x_p) \tag{2.7}$$

Using integrating by parts on the left side of equation(2.7) results in:

$$\int_\Omega \nabla \cdot \phi(-\frac{k}{\mu}\nabla\vec{p}) + \frac{k}{\mu}\nabla\phi(x) \cdot \nabla p d\Omega = -\int_\Omega \sum_{p=1}^{n_{well}} \phi(x) Q_p \delta(x - x_p) d\Omega \tag{2.8}$$

Next is to apply Gauss on the first term of the left side.

$$\int_{d\Omega} \vec{n} \cdot (\phi(x)(-\frac{k}{\mu}\nabla\vec{p}))d\tau + \int_\Omega \frac{k}{\mu}\nabla\phi(x) \cdot \nabla p d\Omega = -\int_\Omega \sum_{p=1}^{n_{well}} \phi(x) Q_p \delta(x - x_p) d\Omega \tag{2.9}$$

Switching the integral and summation on the right side of equation(2.9) and simplifying terms:

$$\int_{d\Omega} (\phi(x)(-\frac{k}{\mu}\frac{d\vec{p}}{dn}))\delta\tau + \int_\Omega \frac{k}{\mu}\nabla\phi(x) \cdot \nabla p d\Omega = -\sum_{p=1}^{n_{well}} \int_\Omega \phi(x) Q_p \delta(x - x_p) d\Omega \tag{2.10}$$

On the right side of equation(2.10) there is now

$$\int_\Omega \delta(x) f(x) d\Omega \ = \ f(0) \tag{2.11}$$

Using equation(2.11) and the boundary condition result in the following equation(2.12):

$$\int_{d\Omega} \phi(x) K(p - p^H)\delta\Gamma + \int_\Omega \frac{k}{\mu}\nabla\phi(x) \cdot \nabla p d\Omega = -\int_\Omega \sum_{p=1}^{n_{well}} \phi(x_p) Q_p \tag{2.12}$$

Rearranging equation(2.12) so that the variable parts are on the left and the constant parts on the right will result in the WF:

(WF):

$$\begin{cases} \text{find p } \epsilon \sum = \{p \ smooth\} \text{ Such that:} \\ \int_{d\Omega} \phi(x) K p \delta\Gamma + \int_\Omega \frac{k}{\mu}\nabla\phi(x) \cdot \nabla p d\Omega = -\sum_{p=1}^{n_{well}} \phi(x_p) Q_p + \int_{d\Omega} \phi(x) K p^H \delta\Gamma \\ \forall\phi \ \epsilon \sum \end{cases} \tag{2.13}$$

To solve the WF the Galerkin equations are applied, where p is replaced by $\sum_{j=1}^{n} c_i\phi_j$ and $\phi(x) = \phi_i(x)$ with $i = [1, .., n]$.

$$\sum_{j=1}^{n} c_i \int_{d\Omega} \phi_i K \phi_j d\Gamma + \int_\Omega \frac{k}{\mu}\nabla\phi(x) \cdot \nabla\phi_j d\Omega = -\sum_{p=1}^{n_{well}} \phi(x_p) Q_p + \int_{d\Omega} \phi_i K p^H \delta\tau \tag{2.14}$$

Equation(2.14) now is of the form $S\vec{c} = \vec{f}$ and like with the 1D problem can be computed. First the element and boundary elements are determined from the Galerkin equations.

2.2 Element matrix and element vector

First the galerkin equation is seperated in its element and boundary components. The element matrix $S_{ij}^{e_i}$ and the element vector $f_i^{e_k}$ are given in equations (2.15) and 2.16 respectively.

$$S_{ij}^{e_i} = \int_{e_k} \frac{k}{\mu}\nabla\phi_i \cdot \nabla\phi_j d\Omega \tag{2.15}$$

$$f_i^{e_k} = -\sum_{p=1}^{n_{well}} \phi(x_p) Q_p \tag{2.16}$$

2.3 Boundary matrix and boundary vector

The boundary matrix $S_{ij}^{b_l}$ and boundary vector $f_i^{b_l}$ can be found in the following equations:

$$S_{ij}^{b_l} = \int_{b_l} K\phi_i\phi_j dx \tag{2.17}$$

$$f_i^{b_l} = Kp^H \int_{b_l} \phi_i dx \tag{2.18}$$

2.4 Assignment 6

2.5 Assignment 7

Program the

nite-element code (GenerateElementMatrix, GenerateEle-mentVector, GenerateBoundaryElementMatrix, GenerateBoundaryElementVector), and evaluate the solution. Use mesh re

nement to evaluate the quality of the solution. Plot your solution in terms of contour plot and a three-dimensional surface plot.

2.6 Assignment 8

Use Darcy's Law, equation (1), to compute the velocities in both directions, by writing both components of equation (1) in a weak form, and by subsequent derivation of the Galerkin equations. Implement this where you solve the resulting systems of linear equations: Mvx = Cxp; Mvy = Cyp; (6) to get vx and vy. Note that the

les Give a plot of the components by adjusting the

les GenerateElementMatrix, GenerateElementVector, Gener- ateBoundaryElementMatrix, GenerateBoundaryElementVector, and now also BuildMatricesandVectors. In the

le WI4243Post, you can add gure(4); quiver(x,y,vx',vy'); axis([-1 1 -1 1]);

2.7 Assignment 9

2.8 Assignment 10

What happens if K = 0? Explain the results.