S. Mishra
L. Scarabosio
J. Sukys

Spring Term 2014

Numerical Methods for Partial Differential Equations

ETH Zürich
D-MATH

Homework Problem Sheet 3

Introduction.

This assignment if fully devoted to the Finite Element Method in 2D.
The first problem concerns the implementation of Linear Finite Elements for the diffusion equation with Dirichlet boundary conditions. The implementation will be developed step by step under the perspective of "finite element assembly" of the Galerkin matrix and the right-handside error.
The second problem is aimed to discretize the same Dirichlet problem but this time by means of *quadratic finite elements*.
Particular attention is given in the problems to the convergence properties of the solution.

In the online handout you can find the mesh data structures that you need to test your routines and perform the convergence studies.
Every file `*.mat` refers to a mesh and contains a struct. For the convergence studies, the meshes are ordered in increasing order for number of degrees of freedom.
Each struct, let's call it `Mesh`, contains the following fields:

- `Mesh.Coordinates`: $N_V \times 2$ array, with N_V the number of vertices, containing the vertex coordinates;
- `Mesh.Edges`: $N_E \times 2$ array, with N_E the number of edges; the i-th row contains the *indices* of the two vertices connected by the edge i;
- `Mesh.Elements`: $N_{El} \times 3$ array, with N_{El} the number of elements; the i-th row contains the *indices* of the three vertices of the element i;
- `Mesh.BdFlags`: $N_E \times 2$ array, with N_E the number of edges, containing the edge boundary flags; the convention is that the boundary flag is 0 is the edge is an interior edge, it is negative for boundary condition flags;
- `Mesh.Vert2Edge`: $N_V \times N_V$ array, with N_V the number of vertices; `Mesh.Vert2Edge(i,j)` contains the index of the edge connecting the vertices i and j.

To load the mesh data structures, in MATLAB you can use the command `load`, in Python the code would be

```
from scipy.io import loadmat
Mesh = loadmat(path_to_file)
print Mesh['Coordinates']
print Mesh['Edges']
```

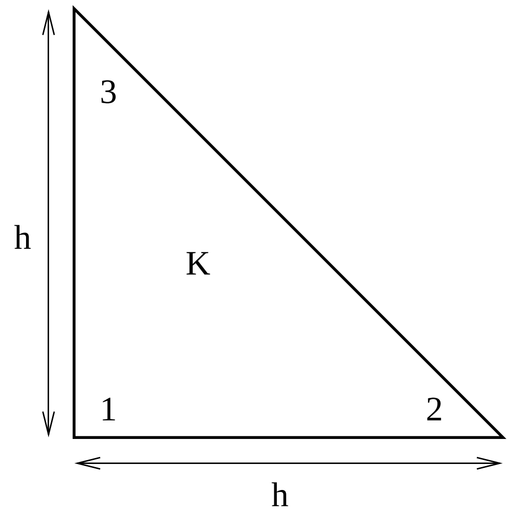


Figure 3.1: Reference element for 2D linear finite elements ($h = 1$).

Problem 3.1 2D Linear Finite Elements (Core problem)

We consider the problem

$$-\operatorname{div}(D(\boldsymbol{x})\,\mathbf{grad}\,u(\boldsymbol{x})) = f(\boldsymbol{x}) \quad \text{in } \Omega \subset \mathbb{R}^2 \tag{3.1.1}$$

$$u(\boldsymbol{x}) = g(\boldsymbol{x}) \quad \text{on } \partial\Omega \tag{3.1.2}$$

where D is uniformly positive and bounded in Ω, g is a continuous function of $\partial\Omega$ and $f \in L^2(\Omega)$.

We solve (3.1.1)-(3.1.2) by means of Galerkin discretization based on piecewise linear finite elements on triangular meshes of Ω.

(3.1a) Write the variational formulation for (3.1.1)-(3.1.2), specifying the bilinear form and the linear form.

(3.1b) Show that the solution to the variational formulation in subproblem (3.1a) exists and is unique when $g = 0$.

(3.1c) Implement the function

```
shap = shap_LFE(x)
```

which computes the the value of the three local shape functions $\lambda_i(\boldsymbol{x})$, $i = 1, 2, 3$, on the reference element depicted in Fig.3.1 at the points x (a $N \times 2$ matrix, where each rows contains the coordinates of a point), and returns the values in the $N \times 3$ matrix `shap` (each row corresponding to the evaluation of the basis functions in a point).

(3.1d) Implement the function

```
shap = grad_shap_LFE(x)
```

which returns the values of the derivatives of local shape functions $\lambda_i(\boldsymbol{x})$, $i = 1, 2, 3$. The input argument `x` follows the same convention as in shap_LFE(x), while the output `shap` is a $N \times 6$ matrix containing the gradients of the shape functions evaluated at the N points (the first two columns contain the gradient of λ_1, and so on).

(3.1e) Implement the routine STIMA_Heat_LFE to compute the element (stiffness) matrices. The function header is

```
Aloc = STIMA_Heat_LFE(Vertices, QuadRule, FHandle)
```

Here, Vertices is a 3×2-vector providing the coordinates of the element vertices, `QuadRule.w` is a vector with quadrature weights and `QuadRule.x` is a vector with quadrature points relative to $\hat{K}$. The function should return a 3×3 matrix Aloc containing the element stiffness matrix. FHandle is a handle to the function D.

HINT: Use `grad_shap_LFE` to compute the gradients of the shape functions.

(3.1f) Implement the routine `LOAD_LFE` to compute the *element vector*. The function header is

```
Lloc = LOAD_LFE(Vertices, QuadRule, FHandle)
```

and follows the same convention as `STIMA_Heat_LFE`.

(3.1g) Implement a function

```
A = assemMat_LFE(Mesh,EHandle,varargin)
```

that assembles the Galerkin matrix A given the mesh structure `Mesh` and a routine `EHandle` to assemble the element matrix. In your implementation make a call `EHandle(Vertices, varargin{:})`, where `Vertices` are the coordinates of an element K_i. Here for `EHandle = STIMA_Heat_LFE` the variable argument list `varargin` should carry the parameters `QuadRule, FHandle`.

HINT: Use the sparse format to store the matrix A.

(3.1h) Implement a function

```
L = assemLoad_LFE(Mesh,QuadRule,FHandle)
```

to compute the right-hand side vector L given the mesh structure `Mesh`, the quadrature rule via `QuadRule` and a handle to the function f via `FHandle`.

HINT: Proceed as for `assemMat_LFE` and use `LOAD_LFE`.

(3.1i) Implement a routine

```
[U,FreeDofs] = assemDir_LFE(Mesh,BdFlag,GHandle)
```

which accepts in input the mesh, the flag `BdFlag` associated to the Dirichlet boundary and the function handle `GHandle` to the boundary data $g(\boldsymbol{x})$. As output, this function should return the degrees of freedom which are not on the Dirichlet boundary, and initialize the solution vector `U` incoporating the Dirichlet boundary conditions for the entries of `U` associated to nodes on the Dirichlet boundary.
In this problem the convention is that the boundary flag is -1 if the edge is on the Dirichlet boundary.

(3.1j) Implement a function

```
plot_LFE(U,Mesh)
```

to plot the FE solution given the vector of coefficients `U` and the structure `Mesh` containing the field `Mesh.Coordinates`.

(3.1k) Implement a function

```
err = L2Err_LFE(Mesh,u,QuadRule,FHandle)
```

that computes the L^2-error of the FEM function given by the coefficient vector `u` and the mesh `Coordinates` to the exact solution given as the function handle `FHandle`.

HINT: Proceed computing the local contributions element-wise and then summing them up to get the total error.

(3.1l) Implement a function

```
err = H1SErr_LFE(Mesh,u,QuadRule,FHandle)
```

that computes the H^1-seminorm error of the FEM function given by the coefficient vector `u` and the mesh `Coordinates` to the exact solution gradient given as the function handle `FHandle`.

HINT: Proceed element-wise as in subproblem subproblem (3.1k).

(3.1m) Implement a function

```
[N,l2,h1s] = main_LFE(Mesh)
```

to compute the FE solution `U` to (3.1.1) with coefficient $D(x) = 1$ and exact solution $u_{\text{ex}} = \cos(2\pi x)\cos(2\pi y)$ on the square $\Omega = (0,1)^2$. The function should return the number `N` of *degrees of freedom* (in this case the nodes which are *not* on the boundary) and the L^2-norm and H^1-seminorm errors.
Inside the function, use the routine implemented in task (3.1j) to plot the solution.
As quadrature rule, use the sixth-order quadrature rule `P7O6()` given in the handout.

(3.1n) Run the routine implementated in task (3.1m) to produce a plot of the solution. For the mesh, load the mesh `Square5.mat` given in the handout.

(3.1o) Consider again the case $u_{\text{ex}} = \cos(2\pi x)\cos(2\pi y)$ and $D(x) = 1$ on the unit square.
Implement a script called `cvg_LFE` to perform the convergence study for the error in the L^2-norm and H^1-seminorm.
Produce loglog plots of the errors versus the number of degrees of freedom.
Use the meshes contained in the file `Square.zip` given in the handout.
Which rates of convergence do you observe?

HINT: You may use the function `main_LFE` implemented in task (3.1m).

We are now going to solve (3.1.1)-(3.1.2) on the L-shaped domain $\Omega = (-1,1)^2 \setminus ((0,1) \times (-1,0))$, as depicted in Fig.3.2.

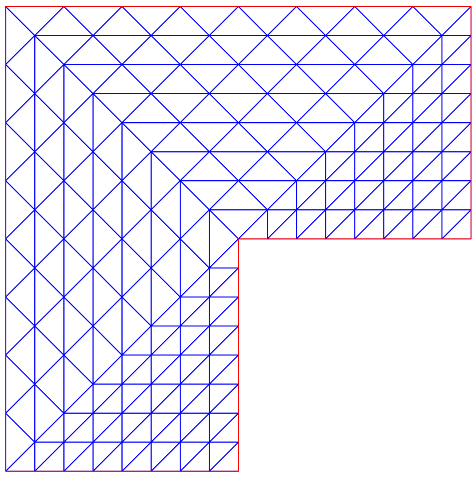


Figure 3.2: Domain for subproblems (3.1p)-(3.1r)

We consider the case that the exact solution is, in polar coordinates, $u = r^{\frac{2}{3}}\sin(\frac{2}{3}\varphi)$, for $(r,\varphi) \in [0,1) \times [0,2\pi)$. The right-handside is then $f = 0$ and the boundary data $g = u_{|\partial\Omega}$.

(3.1p) Implement a function

```
uex = uex_LShap_L2(x)
```

to compute the exact solution $u = r^{\frac{2}{3}}\sin(\frac{2}{3}\varphi)$ given the $N \times 2$ vector of point coordinates `x`, and store the values in the column vector `uex`.

(3.1q) Implement a function

```
uex = uex_LShapH1S(x)
```

to compute the gradient of the exact solution $u = r^{\frac{2}{3}}\sin(\frac{2}{3}\varphi)$ given the $N \times 2$ vector of point coordinates `x`, and store the values in the $N \times 2$ vector `uex`.

(3.1r) Modify the routine `main_LFE` implemented in subproblem (3.1m) and the script `cvg_LFE` implemented in subproblem (3.1o) to perform the convergence study for the L-shaped domain. Use the meshes contained in the zip file `Lshape.zip` given in the handout. Which rates of convergence to you observe?
Give a motivation for your results.

Listing 3.1: Testcalls for Problem 3.1

```
Mesh = load(['Square' num2str(1) '.mat']);
DHandle = @(x) 1;
FHandle = @(x) 8*pi^2*cos(2*pi.*x(:,1)).*cos(2*pi.*x(:,2));
Uex = @(x) cos(2*pi.*x(:,1)).*cos(2*pi.*x(:,2));

```

```
fprintf('\n\n##shap_LFE:')
shap_LFE([0.3 0.6])

fprintf('\n\n##grad_shap_LFE:')
grad_shap_LFE([0.4 0.4])

fprintf('\n\n##STIMA_Heat_LFE:')
STIMA_Heat_LFE([0 0; 1 1/4; 1/8 1],P706(),DHandle)

fprintf('\n\n##LOAD_LFE:')
LOAD_LFE([0 0; 1 1/4; 1/8 1],P706(),FHandle)

fprintf('\n\n##assemMat_LFE:')
A = assemMat_LFE(Mesh, @STIMA_Heat_LFE, P706(),DHandle);
A = full(A);
A(1:6,1:6)
A(20:25,20:25)

fprintf('\n\n##assemLoad_LFE:')
L = assemLoad_LFE(Mesh,P706(),FHandle);
L(1:3)

fprintf('\n\n##assemDir_LFE:')
[U,FreeDofs]=assemDir_LFE(Mesh,-1,Uex);
FreeDofs
```

Listing 3.2: Output for Testcalls for Problem 3.1

```
>> test_call

##shap_LFE:
ans =

    0.1000    0.3000    0.6000

##grad_shap_LFE:
ans =

    -1    -1     1     0     0     1

##STIMA_Heat_LFE:
ans =

    0.6855   -0.3306   -0.3548
   -0.3306    0.5242   -0.1935
   -0.3548   -0.1935    0.5484

##LOAD_LFE:
ans =

    1.2638
    2.3698
    2.5917

##assemMat_LFE:
ans =

     1     0     0     0     0     0
     0     1     0     0     0     0
     0     0     1     0     0     0
     0     0     0     1     0     0
     0     0     0     0     2     0
     0     0     0     0     0     2

ans =

     4    -1    -1     0     0     0
    -1     4     0     0     0     0
    -1     0     4     0     0     0
     0     0     0     4     0    -1
     0     0     0     0     4    -1
     0     0     0    -1    -1     4

##assemLoad_LFE:
ans =

    0.6366
    0.9995
    0.6366

##assemDir_LFE:
FreeDofs =

     7    12    18    20    21    22    23    24    25
```

Problem 3.2 2D Quadratic Finite Elements

We consider the problem

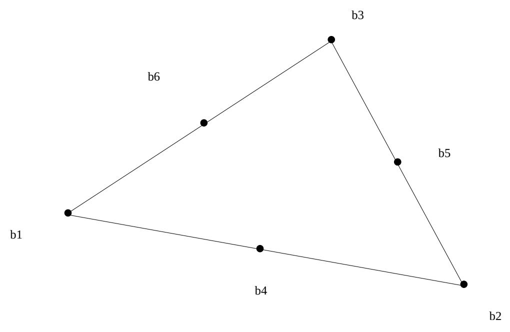
$$-\operatorname{div}(D(\boldsymbol{x})\,\mathbf{grad}\,u(\boldsymbol{x})) = f(\boldsymbol{x}) \quad \text{in } \Omega \subset \mathbb{R}^2 \tag{3.2.1}$$

$$u(\boldsymbol{x}) = g(\boldsymbol{x}) \quad \text{on } \partial\Omega \tag{3.2.2}$$

where D is uniformly positive and bounded in Ω, g is a continuous function of $\partial\Omega$ and $f \in L^2(\Omega)$.

We solve (3.2.1)-(3.2.2) by means of Galerkin discretization based on *piecewise quadratic* finite elements on triangular meshes of Ω.

For quadratic finite elements with affine element mapping a specific choice of element shape functions is given by



$$b_1(\mathbf{x}) := -\lambda_1(\mathbf{x})(1-2\lambda_1(\mathbf{x})),$$
$$b_2(\mathbf{x}) := -\lambda_2(\mathbf{x})(1-2\lambda_2(\mathbf{x})),$$
$$b_3(\mathbf{x}) := -\lambda_3(\mathbf{x})(1-2\lambda_3(\mathbf{x})),$$
$$b_4(\mathbf{x}) := 4\lambda_1(\mathbf{x})\lambda_2(\mathbf{x}),$$
$$b_5(\mathbf{x}) := 4\lambda_2(\mathbf{x})\lambda_3(\mathbf{x}),$$
$$b_6(\mathbf{x}) := 4\lambda_3(\mathbf{x})\lambda_1(\mathbf{x}).$$

(3.2a) Implement the function

```
shap = shap_QFE(x)
```

which computes the the value of the six local shape functions $\lambda_i(\boldsymbol{x})$, $i = 1, \dots, 6$, on the reference element at the points x (a $N \times 2$ matrix, where each rows contains the coordinates of a point), and returns the values in the $N \times 6$ matrix `shap` (each row corresponding to the evaluation of the basis functions in a point).

(3.2b) Implement the function

```
shap = grad_shap_QFE(x)
```

to compute the gradients of the shape functions, following the same convention as in `grad_shap_LFE`.

(3.2c) Implement the routine STIMA_Heat_QFE to compute the element (stiffness) matrices. The function header is

```
Aloc = STIMA_Heat_QFE(Vertices, QuadRule, FHandle)
```

and the conventions are the same as in `STIMA_Heat_LFE`.

(3.2d) Implement the routine `LOAD_QFE`

```
Lloc = LOAD_QFE(Vertices, QuadRule, FHandle)
```

to compute the *element vector*, which follows the same convention as `LOAD_LFE`.

We now consider the assembly part.
The global order of basis function for quadratic finite elements is the following:

- first the basis functions associated to the vertices are stored; the basis function associated to the vertex with index i is $b_N^i(\boldsymbol{x})$;
- then, the basis functions associated to the midpoints, i.e. to the edges, are considered: the basis function associated to the edge i is $b_N^{N_V+i}(\boldsymbol{x})$, with N_V the number of vertices.

(3.2e) Implement a function

```
A = assemMat_QFE(Mesh,EHandle,varargin)
```

that assembles the Galerkin matrix A and follows the same convention as `assem_Mat_LFE`.

HINT: Use the sparse format to store the matrix A.
The field `Mesh.Vert2Edge` may be useful.

(3.2f) Implement a function

```
L = assemLoad_QFE(Mesh,QuadRule,FHandle)
```

to compute the right-hand side vector L, following the same conventions as in `assemLoad_LFE`.

(3.2g) Implement a routine

```
[U,FreeDofs] = assemDir_QFE(Mesh,BdFlag,GHandle)
```

following the same principles as `assemDir_LFE`.

(3.2h) Implement a function

```
err = L2Err_QFE(Mesh,u,QuadRule,FHandle)
```

that computes the L^2-error of the FEM function given by the coefficient vector `u` and the mesh `Mesh` to the exact solution given as the function handle `FHandle`.

(3.2i) Implement a function

```
err = H1SErr_QFE(Mesh,u,QuadRule,FHandle)
```

that computes the H^1-seminorm error of the FEM function given by the coefficient vector `u` and the mesh `Mesh` to the exact solution gradient given as the function handle `FHandle`.

(3.2j) Implement a function

```
[N,l2,h1s] = main_QFE(Mesh)
```

to compute the FE solution `U` to (3.2.1) with coefficient $D(x) = 1$ and exact solution $u_{\text{ex}} = \cos(2\pi x)\cos(2\pi y)$ on the square $\Omega = (0,1)^2$. The routine follows the same conventions as `main_LFE`, but this time you don't need to plot the solution.
Again, as quadrature rule, use the sixth-order quadrature rule `P7O6()` given in the handout.

(3.2k) Consider again the case $u_{\text{ex}} = \cos(2\pi x)\cos(2\pi y)$ and $D(x) = 1$ on the unit square. Implement a script called `cvg_QFE` to perform the convergence study for the error in the L^2-norm and H^1-seminorm.
Produce loglog plots of the errors versus the number of degrees of freedom.
Use the meshes contained in the file `Square.zip` given in the handout.
Which rates of convergence do you observe?

HINT: Use the function `main_QFE` implemented in task (3.2j).

Now we consider again (3.2.1)-(3.2.2) on the L-shaped domain $\Omega = (-1,1)^2 \setminus ((0,1) \times (-10))$. We take again the case that the exact solution is, in polar coordinates, $u = r^{\frac{2}{3}} \sin(\frac{2}{3}\varphi)$, for $(r,\varphi) \in [0,1) \times [0,2\pi)$.

(3.2l) Modify the routine `main_QFE` implemented in subproblem (3.2j) and the script `cvg_QFE` implemented in subproblem (3.2k) to perform the convergence study for the L-shaped domain. Use the meshes contained in the zip file `Lshape.zip` given in the handout. Which rates of convergence to you observe?
Compare your results with the case of Linear Finite Elements and give a motivation for the behavior that you observe.

HINT: You may use the routines `uex_LShap_L2(x)` and `uex_LShapH1S(x)` implemented for the previous problem for the computation of the exact solution and its gradient.

Listing 3.3: Testcalls for Problem 3.2

```matlab
Mesh = load(['Square' num2str(1) '.mat']);
DHandle = @(x) 1;
FHandle = @(x) 8*pi^2*cos(2*pi.*x(:,1)).*cos(2*pi.*x(:,2));
Uex = @(x) cos(2*pi.*x(:,1)).*cos(2*pi.*x(:,2));

fprintf('\n\n##shap_QFE:')
shap_QFE([0.3 0.6])

fprintf('\n\n##grad_shap_QFE:')
grad_shap_QFE([0.4 0.8])

fprintf('\n\n##STIMA_Heat_QFE:')
STIMA_Heat_QFE([0 0; 1 0; 0 1],P706(),DHandle)

fprintf('\n\n##LOAD_QFE:')
LOAD_QFE([0 0; 1 0; 0 1],P706(),FHandle)

fprintf('\n\n##assemMat_QFE:')
A = assemMat_QFE(Mesh, @STIMA_Heat_QFE, P706(),DHandle);
A = full(A);
A(1:3,1:3)
A(26:28,26:28)

fprintf('\n\n##assemLoad_QFE:')
L = assemLoad_QFE(Mesh,P706(),FHandle);
L(1:3)
```

```
L(26:28)

fprintf('\n\n##assemDir_QFE:')
[U,FreeDofs]=assemDir_QFE(Mesh,-1,Uex);
FreeDofs
```

Listing 3.4: Output for Testcalls for Problem 3.2

```
test_call

##shap_QFE:
ans =

  -0.0800   -0.1200    0.1200    0.1200    0.7200    0.2400

##grad_shap_QFE:
ans =

   1.8000    1.8000    0.6000         0         0    2.2000
      -2.4000   -1.6000    3.2000    1.6000   -3.2000   -4.0000

##STIMA_Heat_QFE:
ans =

   1.0000    0.1667    0.1667   -0.6667   -0.0000   -0.6667
   0.1667    0.5000         0   -0.6667   -0.0000    0.0000
   0.1667         0    0.5000    0.0000   -0.0000   -0.6667
  -0.6667   -0.6667    0.0000    2.6667   -1.3333   -0.0000
  -0.0000   -0.0000   -0.0000   -1.3333    2.6667   -1.3333
  -0.6667    0.0000   -0.6667   -0.0000   -1.3333    2.6667

##LOAD_QFE:
ans =

   1.0920
   0.1993
   0.1993
  -1.7408
   5.2648
  -1.7408

##assemMat_QFE:
ans =

   1.0000         0         0
        0    1.0000         0
        0         0    1.0000

ans =

   2.6667   -0.0000         0
  -0.0000    2.6667         0
        0         0    5.3333

##assemLoad_QFE:
ans =

   0.0590
   0.1889
   0.0590

ans =

   0.5776
   0.5776
   0.7577

##assemDir_QFE:
FreeDofs =

  Columns 1 through 20

     7    12    18    20    21    22    23    24    25    28    34
          36    39    41    42    44    45    46    47    48

  Columns 21 through 40

    49    50    53    54    57    58    59    60    61    62    63
          64    65    66    67    68    69    70    71    72

  Columns 41 through 49

    73    74    75    76    77    78    79    80    81
```

Published on March 31.

To be submitted on April 14.

Last modified on April 11, 2014