TU Berlin — Department of Mathematics winter term 2015/2016
Dr. Kersten Schmidt, Dirk Klindworth

NUMERICS OF PARTIAL DIFFERENTIAL EQUATIONS

Series 7

Important note: On the course homepage you find the Python file `FEM.py`. This file includes two functions:

- `gaussTriangle` — a function needed for the numerical quadrature in triangles in exercise 1c, and
- `plot` — a function for plotting your finite element solution using Matplotlib's `plot_trisurf`, which is needed in exercises 3 and 4.

Please add all your Python functions that you write in exercises 1, 2 and 4a to this file, and import it as module into the scripts that you write in exercises 3 and 4!

1. Element stiffness matrix, element mass matrix and element load vector for linear finite elements

a) Add a function `elemStiffness` to your Python module `FEM.py` that returns the element stiffness matrix for constant material coefficient $a_K = 1$. The input parameters are the coordinates of the three nodes.

```
#
# elemStiffness(p)
#
# computes the element stiffness matrix related to the bilinear
# form
#   a_K(u,v) = int_K grad u . grad v dx
# for linear FEM on triangles.
#
# input:
# p - 3x2-matrix of the coordinates of the triangle nodes
#
# output:
# AK - element stiffness matrix
#
```

b) Add a function `elemMass` to your Python module `FEM.py` that returns the element mass matrix for constant material coefficient $c_K = 1$. The input parameters are the coordinates of the three nodes.

```
#
# elemMass(p)
#
# computes the element mass matrix related to the bilinear form
#   m_K(u,v) = int_K u v dx
# for linear FEM on triangles.
#
# input:
```

See next page!

```
# p - 3x2-matrix of the coordinates of the triangle nodes
#
# output:
# MK - element mass matrix
#
```

c) Using Gauß quadrature for triangles write a function `elemLoad` that returns an approximation to the element load vector related to the linear form $\ell_K(v) = \int_K f\, v\, \mathrm{d}\mathbf{x}$. The coordinates of the triangle vertices, the number of quadrature points, and the function f should be given as input parameters. Add this function to your Python module `FEM.py`

```
#
# elemLoad(p, n, f)
#
# returns the element load vector related to linear form
#   l_K(v) = int_K f v dx
# for linear FEM on triangles.
#
# input:
# p - 3x2 matrix of the coordinates of the triangle nodes
# n - order of the numerical quadrature (1 <= n <= 5)
# f - source term function
#
# output:
# fK - element load vector (3x1 array)
#
```

The input parameter `f` should either be a standard Python function, e.g.

```
def f(x,y):
  return x*y
```

or a Python's `lambda` function, e.g.

```
f = lambda x,y: x*y
```

Formulas for numerical quadratures on triangles are defined on the triangle $\widetilde{K}$ with the nodes $(-1,-1)$, $(1,-1)$, $(-1,1)$, see, e.g. page 141 in P. Solin, "Partial Differential Equations and the Finite Element Method", John Wiley & Sons, 2006. On the other hand, the element shape functions are defined on the reference triangle $\widehat{K}$ with nodes $(0,0)$, $(1,0)$, $(0,1)$. Transform the integrals over $\widehat{K}$ to integrals over $\widetilde{K}$.

For the Gauß quadrature rules on triangles use the function `gaussTriangle` that provides the quadrature points and the corresponding weights for all orders up to five. You find the function `gaussTriangle` in the file `FEM.py`, that is available on the course webpage.

Check your code (at least) with $f = 1$. Note that $\sum_{j=0}^{2} (\mathbf{f})_j$ for $f = 1$ is equal to the area of the triangle, where $(\mathbf{f})_j$, $j = 0, 1, 2$, are the three components of the element load vector.

See next page!

2. Assembling of stiffness matrix, mass matrix and load vector

a) Add a function `stiffness` to your Python module `FEM.py` that returns the stiffness matrix in sparse format.

```
#
# stiffness(p, t)
#
# returns the stiffness matrix related to the bilinear form
#   int_Omega grad u . grad v dx
# for linear FEM on triangles.
#
# input:
# p - Nx2 matrix with coordinates of the nodes
# t - Mx3 matrix with indices of nodes of the triangles
#
# output:
# Stiff - NxN stiffness matrix in scipy's sparse lil format
#
```

Use the function `elemStiffness` and the relation of local numbering of shape functions and global numbering of basis functions given the matrix `t`.

b) Add a function `mass` to your Python module `FEM.py` that returns the mass matrix in sparse format.

```
#
# mass(p, t)
#
# returns the mass matrix related to the bilinear form
#   int_Omega u v dx
# for linear FEM on triangles.
#
# input:
# p - Nx2 matrix with coordinates of the nodes
# t - Mx3 matrix with indices of nodes of the triangles
#
# output:
# Mass - NxN mass matrix in scipy's sparse lil format
#
```

Use the function `elemMass` and the relation of local numbering of shape functions and global numbering of basis functions given the matrix `t`.

c) Add a function `load` to your Python module `FEM.py` that returns the load vector.

```
#
# load(p, t, n, f)
#
# returns the load vector related to the linear form
#   int_Omega f v dx
# for linear FEM on triangles.
#
# input:
```

See next page!

```
# p - Nx2 matrix with coordinates of the nodes
# t - Mx3 matrix with indices of nodes of the triangles
# n - order of the numerical quadrature (1 <= n <= 5)
# f - source term function
#
# output:
# Load - Nx1 load vector as numpy-array
#
```

Use the function `elemLoad` and the relation of local numbering of shape functions and global numbering of basis functions given the matrix `t`.

See next page!

3. Linear FE solver for BVPs with homogeneous Neumann boundary conditions

Let $\Omega \subset \mathbb{R}^2$ be an open, bounded Lipschitz domain. Given a function $f \in L^2(\Omega)$, we are interested in the computation of a linear finite element approximation to the solution of

$$-\Delta u + u = f \quad \text{in } \Omega, \tag{1a}$$

$$\mathbf{grad}\, u \cdot \mathbf{n} = 0 \quad \text{on } \partial\Omega. \tag{1b}$$

a) Write the main code for solving (1) including

- definition of the source term f as Python function or Python's `lambda` function,
- mesh generation for the square $\Omega =]0,1[^2$ in dependence of the maximal mesh width h_0,
- computation of the stiffness and mass matrix,
- computation of the load vector in dependence of the source term f and the order n of the numerical integration,
- conversion of the sparse matrices from Scipy's `lil_matrix` format to some other format that is more appropriate for solving the system, e.g. use Scipy's `csr` format,
- solution of the linear system using Scipy's `spsolve`, and
- graphical representation with Matplotlib's `plot_trisurf` (for this you may use the function `plot` in the file `FEM.py`).

b) Let $h_0 = 0.1$ be the maximal mesh width and $n = 3$ the order of the numerical quadrature. Choose $f(x, y)$ such that $u(x, y) = \cos(2\pi x)\cos(2\pi y)$ is the analytical solution of (1). Compute and plot your finite element approximation u_n of u and its discretization error $e_n = u - u_n$.

See next page!

4. Linear FE solver for BVPs with Dirichlet boundary conditions

Let $\Omega \subset \mathbb{R}^2$ be an open, bounded Lipschitz domain. Given the functions $f \in L^2(\Omega)$ and $g \in H^{1/2}(\partial\Omega)$, we are interested in the solution of

$$-\Delta u + u = f \qquad \text{in } \Omega =]0,1[^2, \tag{2a}$$

$$u = g \qquad \text{on } \partial\Omega. \tag{2b}$$

a) Using your the Numpy array `be` of indices of nodes that lie on boundary edges, add a function `interiorNodes` to your Python module `FEM.py` that returns all interior nodes of a triangulation, i. e. nodes that do not lie on the boundary.

```
#
# interiorNodes(p, t, be)
#
# returns the interior nodes as indices into p.
#
# input:
# p  - Nx2 array with coordinates of the nodes
# t  - Mx3 array with indices of nodes of the triangles
# be - Bx2 array with indices of nodes on boundary edges
#
# output:
# IN - Ix1 array of nodes as indices into p that do not lie on
#      the boundary
#
```

b) Write the main code for solving (2) with homogeneous Dirichlet boundary conditions, i. e. with $g \equiv 0$, including

- definition of the source term f as Python function or Python's `lambda` function,
- mesh generation for the square $\Omega =]0,1[^2$ in dependence of the maximal mesh width h_0,
- computation of the stiffness and mass matrix,
- computation of the load vector in dependence of the source term f and the order n of the numerical integration,
- solution of the linear system using Scipy's `spsolve`, and
- graphical representation of the solution with Matplotlib's `plot_trisurf`.

In order to compute the system matrix and load vector, assemble them while ignoring the essential boundary condition (2b), i. e. assemble the matrices and vectors that correspond to (2a) with homogeneous Neumann boundary conditions. Respect the homogeneous Dirichlet boundary condition by initializing your solution vector (of size `Nx1`) with zeros, solving the reduced system obtained when considering only interior nodes, and writing the solution of the reduced system into the initialized solution vector.

c) Let $h_0 = 0.1$ be the maximal mesh width and $n = 3$ the order of the numerical quadrature. Choose $f(x, y)$ such that $u(x, y) = \sin(\pi x)\sin(\pi y)$ is the analytical solution of (2). Compute and plot your FE approximation u_n of u and its discretization error $e_n = u - u_n$.

d) Using the Dirichlet lift approach, write the main code for solving (2) with inhomogeneous Dirichlet boundary conditions, i. e. with $g \not\equiv 0$, including

- definition of the source term f and the Dirichlet data g as Python function or Python's `lambda` function,

See next page!

- mesh generation for the square $\Omega =]0, 1[^2$ in dependence of the maximal mesh width h_0,
- computation of the stiffness and mass matrix,
- computation of the load vector in dependence of the source term f, the order n of the numerical integration and the Dirichlet data g,
- solution of the linear system using Scipy's `spsolve`, and
- graphical representation of the solution with Matplotlib's `plot_trisurf`.

Hint: Choose the approximate Dirichlet lift $u_g \in S^1(\Omega, \mathcal{M})$, that is zero on all interior nodes of $\mathcal{M}$ and whose values on the boundary nodes of $\mathcal{M}$ correspond to the values of g. Then the vectors related to the linear forms $\int_\Omega \nabla u_g \cdot \nabla v \, \mathrm{d}\boldsymbol{\xi}$ and $\int_\Omega u_g v \, \mathrm{d}\boldsymbol{\xi}$ can be obtained with the help of the stiffness and mass matrix, respectively.

e) Let $h_0 = 0.1$ be the maximal mesh width and $n = 3$ the order of the numerical quadrature. Choose $f(\mathbf{x}) = 0$ and $g(\mathbf{x}) = x_1 + x_2$ in (2), and compute and plot your FE approximation u_n of u.

To be handed in by: December 8th, 2015, 10:15 a.m.

Send your Python module `FEM.py`, the `.msh`-files of all used Gmsh meshes (if any), and the Python scripts `series7_3.py`, `series7_4c.py` and `series7_4e.py` that do all steps explained in exercises 3, 4c and 4e without any need of interaction when executed from the command line of the operating system, to `klindworth@math.tu-berlin.de`!
The code must be well documented using prologue and inline comments! The plots must have titles and labels!

There will be no tutorial class on December 3rd. Instead there will be a voluntary but recommended consultation in MA 366 starting at 2:30 p.m. Bring your own laptop! In this consultation you can continue developing your code while Dirk Klindworth will be available for questions.

This exercise series will be discussed in the tutorial class on December 10th, 2015, 2:15 p.m. in A 052.

Coordinator: Dirk Klindworth, MA 365, 030/314-25192, `klindworth@math.tu-berlin.de`
Website: `http://www.tu-berlin.de/?NumPDE`