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

NUMERICS OF PARTIAL DIFFERENTIAL EQUATIONS

Series 9

1. Integration with barycentric coordinates

Prove that for any non-generate triangle K and $\beta_1, \beta_2, \beta_3 \in \mathbb{N}_0$ it holds

$$\int_K \lambda_1^{\beta_1} \lambda_2^{\beta_2} \lambda_3^{\beta_3} \,\mathrm{d}\mathbf{x} = 2|K| \cdot \frac{\beta_1!\, \beta_2!\, \beta_3!}{(\beta_1 + \beta_2 + \beta_3 + 2)!} \,. \tag{1}$$

Hint: Transform the integral onto the reference triangle and use a transformation which leads to an expression with Euler's beta function

$$B(\alpha, \beta) := \int_0^1 t^{\alpha-1}(1-t)^{\beta-1} \,\mathrm{d}t \,, \quad 0 < \alpha, \beta < \infty \,.$$

Then, use the formula $\Gamma(\alpha + \beta)\, B(\alpha, \beta) = \Gamma(\alpha)\Gamma(\beta)$*, with* Γ *being the Gamma function and* $\Gamma(n) = (n-1)!$ *for* $n \in \mathbb{N}$.

2. Edge indices for quadratic finite elements

In addition to the nodal basis functions of linear finite elements, quadratic finite elements also have basis functions associated to the edges. Therefore, we need to assign indices to all edges of the mesh and relate these edge indices with the node indices, e.g. between nodes n_1 and n_2 there is edge e.

For this, we can employ, for example, a sparse $N \times N$–matrix, where N is the number of nodes, that contains the indices of the edges that connect the nodes. Using the example above, the entry in row n_1 and column n_2 is e. Since for quadratic finite elements the orientation of the edges does not matter, we can simply create an upper or lower triangular matrix.

To create such an array, either

- write a function `edgeIndex` and add it to your Python module `meshes.py`, that accepts the arrays `p` and `t` as input parameters and that returns the desired array, or
- directly extend your structured mesh generator `grid_square` in your Python module `meshes.py`, such that it also returns the desired array.

See next page!

3. Quadratic finite element solver

For quadratic finite elements on triangles we use the shape functions

$$\begin{aligned} N_0(\boldsymbol{\lambda}) &= \lambda_1, & N_1(\boldsymbol{\lambda}) &= \lambda_2, & N_2(\boldsymbol{\lambda}) &= \lambda_3, \\ N_3(\boldsymbol{\lambda}) &= \lambda_2\lambda_3, & N_4(\boldsymbol{\lambda}) &= \lambda_1\lambda_3, & N_5(\boldsymbol{\lambda}) &= \lambda_1\lambda_2, \end{aligned}$$

where $\lambda_j, j = 1, 2, 3$ are the barycentric coordinates related to the nodes of the triangle with coordinates $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$ in counter-clockwise order.

Note, that for the triangle the barycentric coordinates are given by

$$\lambda_j(\mathbf{x}) = \frac{1}{2|K|}(\mathbf{x} - \mathbf{p}_{j+1}) \cdot (\mathbf{p}_{j-1} - \mathbf{p}_{j+1})^\perp$$

with index meant modulus 3 and where $\mathbf{v}^\perp = (-v_2, v_1)^\top$. The gradient of the barycentric coordinates are the (constant) vectors

$$\mathbf{grad}\,\lambda_j(\mathbf{x}) = \frac{1}{2|K|}\left(\mathbf{p}_{j-1} - \mathbf{p}_{j+1}\right)^\perp,$$

and we can express the matrix $\mathbf{G}_K$ with entries $(\mathbf{G}_K)_{ij} = \mathbf{grad}\,\lambda_j \cdot \mathbf{grad}\,\lambda_i$ by

$$\mathbf{G}_K = \frac{1}{4|K|^2}\mathbf{D}_K^\top\mathbf{D}_K,$$

where $\mathbf{D}_K$ is the matrix with coordinate differences.

a) Write the 6×6 stiffness matrix $\mathbf{A}_K$ related to the bilinear form

$$\mathsf{a}_K(u, v) = \int_K \mathbf{grad}\,u \cdot \mathbf{grad}\,v\,\mathrm{d}\mathbf{x}$$

in terms of the matrix entries $\mathbf{G}_K$.
Hint: Use Eq. (1) *for the entries related to higher order polynomials.*

b) Add a function `elemStiffnessP2` to your Python module `FEM.py` that returns the element stiffness matrix for quadratic finite elements on triangles with constant material coefficient $a_K = 1$. The input parameters are the coordinates of the three nodes.

```
#
# elemStiffnessP2(p)
#
# computes the element stiffness matrix related to the bilinear
# form
#   a_K(u,v) = int_K grad u . grad v dx
# for quadratic FEM on triangles.
#
# input:
# p - 3x2-matrix of the coordinates of the triangle nodes
#
# output:
# AK - element stiffness matrix
#
```

See next page!

c) Add a function `elemMassP2` to your Python module `FEM.py` that returns the element mass matrix for quadratic finite elements on triangles with constant material coefficient $c_K = 1$. The input parameters are the coordinates of the three nodes.

```
#
# elemMassP2(p)
#
# computes the element mass matrix related to the bilinear form
#   m_K(u,v) = int_K u v dx
# for quadratic FEM on triangles.
#
# input:
# p - 3x2-matrix of the coordinates of the triangle nodes
#
# output:
# MK - element mass matrix
#
```

d) Using Gauß quadrature for triangles add a function `elemLoadP2` to your Python module `FEM.py`, that returns an approximation to the element load vector related to the linear form $\ell_K(v) = \int_K f\, v\, \mathrm{d}\mathbf{x}$ for quadratic finite elements on triangles. The coordinates of the triangle vertices, the order of the quadrature rule, and the function f should be given as input parameters.

```
#
# elemLoadP2(p, n, f)
#
# returns the element load vector related to linear form
#   l_K(v) = int_K f v dx
# for quadratic 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)
#
```

e) (optional) Improve your function `elemLoadP2` by evaluating the integrals using the Duffy trick and the Gauß-Legendre quadrature rule (tensor product on square $[-1, 1]^2$). For this, use Numpy's `leggauss` function `numpy.polynomial.legendre.leggauss`.

f) Add the functions `stiffnessP2`, `massP2` and `loadP2` to your Python module `FEM.py`, that return the stiffness matrix, the mass matrix and the load vector for quadratic finite elements on triangular meshes. Besides the arrays `p` and `t` with the coordinates of the vertices and the indices of the triangles, the functions should also take the array computed in Exercise 2 as input parameter. To obtain a global numbering of basis functions, begin with those basis functions identified to nodes and then those identified to edges.

g) Solve the BVP in Exercise 3b) in Series 7. Plot your solution using the function `plot` of your Python module `FEM.py` while ignoring the degrees of freedom associated to the edges, i.e. only pass the first N components of your solution vector to the function `plot`, where N is the number of vertices in your mesh.

See next page!

h) Do a convergence study like in Exercise 3 in Series 8 and compare your results with the results of your linear finite element solver. For this, plot the two results in one figure using double-logarithmic scaling. What are the observed convergence rates?

To be handed in by: January 18th, 2016, 9:00 a.m.
Your solutions of Exercise 1 have to be handed in in paper form or as PDF file sent to Anastasia Thöns (`anastasia.thoens@math.tu-berlin.de`)!
Send your Python modules `FEM.py` and `meshes.py`, the `.msh`-files of all used Gmsh meshes (if any), and the Python scripts `series9_3g.py` and `series9_3h.py`, that do all steps explained in Exercises 3g) and 3h) without any need of interaction when executed from the command line of the operating system, to Anastasia Thöns (`anastasia.thoens@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 January 14th. 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 Anastasia Thöns will be available for questions.

This exercise series will be discussed in the tutorial class on January 21st, 2016, 2:15 p.m. in A 052.

Coordinator: Anastasia Thöns, MA 365, 030/314-79369, `anastasia.thoens@math.tu-berlin.de`
Website: `http://www.tu-berlin.de/?NumPDE`