T U T O R I A L

“Numerical Methods for the Solution of Elliptic Partial Differential Equations”

to the lecture

“Numerics of Elliptic Problems”

Tutorial 07 Tuesday, 16 May 2017, Time: $10^{\underline{15}}$ – $11^{\underline{45}}$, Room: HT 177F.

Programming

Reference element

In this and the next tutorials we consider Courant’s finite element. The reference triangle is given by

$$\Delta = \{\xi \in \mathbb{R}^2 : \xi_1 \geq 0,\ \xi_2 \geq 0,\ \xi_1 + \xi_2 \leq 1\},$$

with vertices $\xi^{(0)} = (0,0)$, $\xi^{(1)} = (1,0)$, and $\xi^{(2)} = (0,1)$, the space of shape functions is P_1, and the nodal variables are the evaluations at the three vertices. Recall that the nodal shape functions are given by

$$\begin{aligned} p^{(0)}(\xi) &= 1 - \xi_1 - \xi_2, \\ p^{(1)}(\xi) &= \xi_1, \\ p^{(2)}(\xi) &= \xi_2. \end{aligned}$$

To model *small* vectors from $\mathbb{R}^n$ and $n \times m$ matrices, where $m, n \in \{2, 3\}$, I recommend to use `vec.hh` and `mat.hh` (see also the demo `matvecdemo.cc`). There 0-based indices are used throughout, for example:

$\xi \in \mathbb{R}^2 \leftrightarrow$ `Vec<2> xi` $\qquad$ $\xi_1 \leftrightarrow$ `xi[0]`

$\xi_2 \leftrightarrow$ `xi[1]`

30 Write two functions

```
double calcShape (int i, const Vec<2>& xi);
Vec<2> calcDShape (int i, const Vec<2>& xi);
```

that compute the *value* $p^{(\alpha)}(\xi)$ and the *gradient* $\nabla_\xi p^{(\alpha)}(\xi)$ of a nodal shape function, respectively, where `xi`$=\xi$ and `i`$=\alpha$.

31 Complete and implement the following class modelling the affine linear transformation x_δ from Δ to an *arbitrary* non-degenerate triangle δ:

$$x = x_\delta(\xi) = x_0 + J\,\xi,$$

where x_0 is the image of $(0,0)$.

```
class ElTrans {
public:
  ElTrans(const Vec<2>& x0, const Vec<2>& x1, const Vec<2>& x2);
  void transform (const Vec<2>& xi, Vec<2>& x);
  void getJacobian (Mat<2, 2>& J);
  ...
};
```

Above, `x0`, `x1`, `x2` are the three vertices of δ. The method `transform` should transform reference coordinates `xi`$=\xi$ to real coordinates `x`$=x_\delta(\xi)$. The method `getJacobian` should return the Jacobi matrix J of the transformation.

32 Add two more methods to `class ElTrans`:

```
  double jacobiDet ();
  void getInvJacobian (Mat<2, 2>& invJ);
```

The first should return the Jacobi determinant $\det J$ (check if the determinant is positive, why?), the second one should return `invJ`$=J^{-1}$.

33 Write a function

```
  void calcLaplaceElMat (const Vec<2>& x0, const Vec<2>& x1,
                         const Vec<2>& x2, Mat<3, 3>& elMat);
```

that computes the element stiffness matrix `elMat`$=K_r$ associated to an element δ_r (given by the three vertices `x0`, `x1`, and `x2`), i. e.

$$(K_r)_{\alpha\beta} \;=\; \int_{\delta_r} \nabla_x p^{(r,\alpha)}(x)\cdot\nabla_x p^{(r,\beta)}(x)\,dx \;=\; \int_\Delta \left(J_r^{-T}\nabla_\xi p^{(\alpha)}(\xi)\right)\cdot\left(J_r^{-T}\nabla_\xi p^{(\beta)}(\xi)\right)\,\det(J_r)\,d\xi.$$

Hint: Consider only the above formula on the reference element. Use `calcDShape` to get $\nabla_\xi p^{(\alpha)}(\xi)$, and `ElTrans` to get $\det J$ and J_r^{-1}. Note finally that J_r^{-T} and $\nabla_\xi p^{(\alpha)}$ are constant on Δ.

34 Write a function

```
  void calcSourceElVec (const Vec<2>& x0, const Vec<2>& x1,
      const Vec<2>& x2, ScalarField f, Vec<3>& elVec);
```

that approximates the element load vector f_r given by

$$(f_r)_\alpha \;=\; \int_{\delta_r} f(x)\,p^{(r,\alpha)}(x)\,dx \;=\; \int_\Delta f(x_{\delta_r}(\xi))\,p^{(\alpha)}(\xi)\,\det(J_r)\,d\xi,$$

using the following quadrature rule on Δ:

$$\int_\Delta g(\xi)\,d\xi \;\approx\; \frac{1}{6}\Big[g(\tfrac{1}{6},\tfrac{1}{6}) + g(\tfrac{4}{6},\tfrac{1}{6}) + g(\tfrac{1}{6},\tfrac{4}{6})\Big].$$

Show that this quadrature rule is exact for $g \in P_2$.

Hint: Use `ElTrans` to get $x_{\delta_r}(\xi)$. Note that ξ must *loop* over the three integration points.

Hint: To model the *type* of a scalar function depending on a vector in $\mathbb{R}^2$ use

```
typedef double (*ScalarField)(const Vec<2>& x);
```

35 Write a function

```
void calcMassElMat (const Vec<2>& x0, const Vec<2>& x1,
                    const Vec<2>& x2, Mat<3, 3>& elMat);
```

that computes the element mass matrix M_r given by

$$(M_r)_{\alpha\beta} \;=\; \int_{\delta_r} p^{(r,\alpha)}(x)\, p^{(r,\beta)}(x)\, dx$$

Hint: Transform to the reference element as done in the previous two exercises.

Test all your functions, i. e. apply them to concrete parameters and output the results! At minimum use $f(x, y) = 1$ and test $\delta_r = \Delta$ as well as the triangle with the vertices $(1, 1)$, $(1.5, 1)$, and $(1.25, 1.5)$.

Assembling

Download the files

- `vector.hh` – a vector class (for vectors of dynamic length)
- `sparsematrix.hh`, `sparsematrix.cc` – a sparse matrix class
- `mesh.hh`, and `mesh.cc` – a 2D triangular mesh

from the tutorial website.
There are also two demos:

- `smdemo.cc` – showing how to work with the sparse matrix and
- `meshdemo.cc` – showing how to work with the mesh.

Go through these demos and understand what is happening there.

36 Write a function

```
void assembleStiffnessMatrix (const Mesh& mesh, SparseMatrix& K);
```

that assembles the stiffness matrix `K` according to the bilinear form

$$a(u, v) \;=\; \int_{\Omega} \nabla u(x) \cdot \nabla v(x) + u(x)\, v(x)\, dx$$

for `mesh` being the triangulation of Ω.
Hint: Reuse the functions from the previous section, in particular exercises 33 and 35.

37 Write a function

```
void assembleLoadVector (const Mesh& mesh, ScalarField f, Vector& b);
```

that assembles the load vector `b` according to the functional

$$\langle F,\, v\rangle \;=\; \int_{\Omega} f(x)\, v(x)\, dx$$

for `mesh` being the triangulation of Ω.
Hint: Reuse the function from exercise 34.

All routines should be tested for the two meshes created in `meshdemo.cc`

Solving

As a concrete example we consider the problem to find $u \in H^1(\Omega)$ such that

$$\int_\Omega \nabla u(x) \cdot \nabla v(x) + u(x)\, v(x)\, dx \;=\; \int_\Omega f(x)\, v(x)\, dx \qquad \forall v \in H^1(\Omega), \tag{3.22}$$

with $f(x_1, x_2) = (5\pi^2 + \frac{1}{4})\, \cos(2\pi\, x_1)\, \cos(4\pi\, x_2)$.

38 Implement a Jacobi preconditioner:

```
class JacobiPreconditioner
{
public:
  JacobiPreconditioner (const SparseMatrix& K);
  void solve (const Vector& r, Vector& z);
};
```

39 Assemble the finite element system $K\, u = b$ for (3.22) for the initial mesh from `meshdemo.cc` and solve it using conjugate gradients `cg.hh` with your Jacobi preconditioner. Solve the same system for the uniformly refined meshes with $h/h_0 = 2, 4, 8, 16$ where h_0 is the mesh size of the initial mesh.

You can visualize solutions calling `mesh.matlabOutput ("output.m", u);` from your program, and then loading the file into `matlab` (provided you have the PDE Toolbox).