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 - 10 - 13

Tuesday, 13 May 2014 - Tuesday, 03 June 2014 - Tuesday, 24 June 2014

Time: $10^{\underline{15}}$ – $11^{\underline{45}}$, Room: S2 / 120.

Programming project

In this project we will develop step by step a C++ code realizing a 2D Finite Element Method. In our first meeting on May 13 you should present P01 - P06. Then you have time until the end of the semester (June 24) to finish the project. You can download the file `tut07-10-13.zip` containing some auxiliary classes on the website of the lecture/tutorial. Please make sure that your code is well-structured and well-documented and deliver it to `gangl@numa.uni-linz.ac.at` by June 24, 2014. The project will be graded and will count as 1/3 of the total grade of the tutorial.

Reference element

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 $m,\ n \in \{2,3\}$, it is recommended 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]`

P01 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`=ξ and `i`=α.

P02 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`=ξ to real coordinates `x`=$x_\delta(\xi)$. The method `getJacobian` should return the Jacobi matrix J of the transformation.

P03 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}.

P04 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.,

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

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.

P05 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);
```

P06 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.

P07 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 P04 and P06.

P08 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 P05.

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

$$\int_\Omega \nabla u(x)\cdot\nabla v(x) + u(x)\,v(x)\,dx \;=\; \int_\Omega f(x)\,v(x)\,dx \qquad \forall v \in H^1(\Omega), \tag{3.14}$$

with $f(x_1,x_2) = (5\pi^2 + \frac{1}{4})\cos(2\pi\,x_1)\cos(4\pi\,x_2)$. Give the classical formulation of formulation (3.14)!

P09 Implement a Jacobi preconditioner:

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

P10 Assemble the finite element system $K\,u = b$ for (3.14) 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 $h_0/h = 2, 4, 8, 16$ 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).

Incorporating boundary conditions

Consider the Neumann boundary value problem

$$\begin{aligned} -\Delta u(x) + u &= f(x) \qquad \text{for } x \in \Omega := (0,1)^2, \\ \frac{\partial u}{\partial n}(x) &= g(x) \qquad \text{for } x \in \Gamma_N := \partial\Omega\,. \end{aligned}$$

The associated variational formulation is to find $u \in V_0 := H^1(\Omega)$ such that

$$\int_\Omega \nabla u(x) \cdot \nabla v(x) + u(x)\, v(x)\, dx \;=\; \int_\Omega f(x)\, v(x)\, dx + \int_{\Gamma_N} g(x)\, v(x)\, ds \qquad \forall v \in V_0\,. \tag{3.15}$$

P11 Let $e \subset \Gamma_N$ be an element edge on the Neumann boundary with the two endpoints $x^{(e,1)}$ and $x^{(e,2)}$ and set $h_e := |x^{(e,2)} - x^{(e,1)}|$. Let us denote the two functions on the reference edge by $p^{(1)}(\xi) = 1 - \xi$ and $p^{(2)}(\xi) = \xi$.

Write a function

```
void calcNeumannElVec (const Point2D& p0, const Point2D& p1,
                       ScalarField g, Vec<2>& elVec);
```

to approximate

$$g_e^{(\alpha)} \;:=\; \int_e g(x)\, p^{(e,\alpha)}(x)\, ds \;\approx\; \frac{h_e}{2}\left(g(x^{(e,1)})\, p^{(\alpha)}(0) + g(x^{(e,2)})\, p^{(\alpha)}(1)\right)$$

as above by the trapezoidal rule; `elVec`$\approx (g_e^{(1)}, g_e^{(2)})$, `p0`$=x^{(e,1)}$, `p1`$=x^{(e,2)}$, and `g`$=g$.

P12 Write a function

```
void addNeumannLoadVector (const Mesh& mesh, ScalarField g, Vector& b);
```

which *adds* the contribution corresponding to $\int_{\Gamma_N} g(x)\, v(x)\, ds$ to an (already existing) load vector `b`.

Hint: Loop over all segments of the mesh and for those marked as Neumann (use `bcSegments[i] == BC_NEUMANN`) call `calcNeumannElVec`.

P13 Solve the finite element system corresponding to (3.15) with $f(x_1, x_2) = -2.5 + x_1$ and $g(x_1, x_2) = 0.5$ for a suitably refined mesh (see exercise P10) and visualize the solution.

Consider the Dirichlet boundary value problem

$$\begin{aligned} -\Delta u(x) &= f(x) \qquad \text{for } x \in \Omega := (0,1)^2, \\ u(x) &= g \qquad \text{for } x \in \Gamma_D := \partial\Omega. \end{aligned}$$

The associated variational formulation is to find $u \in V_g := \{u \in H^1(\Omega) : u|_\Gamma = g\}$ such that

$$\int_\Omega \nabla u(x) \cdot \nabla v(x)\, dx \;=\; \int_\Omega f(x)\, v(x)\, dx \qquad \forall v \in V_g\,. \tag{3.16}$$

P14 Write a function

```
void incorporateHomogeneousDirichletBC (const Mesh& mesh,
                                        SparseMatrix& K, Vector& b);
```

that incorporates the homogeneous Dirichlet boundary conditions ($g = 0$) into the system matrix `K` and the load vector `b`.

Hint: Loop over all segments of the mesh and search for those marked as Dirichlet (use `bcSegments[i] == BC_DIRICHLET`). For each such vertex with index i it sets all entries in row i and column i of K to zero and $K_{i,i} = 1$, $b_i = 0$.

P15 Solve the finite element system corresponding to (3.16) with $f(x_1, x_2) = 20\pi^2 \sin(2\pi x_1) \sin(4\pi x_2)$ for a suitably refined mesh (see exercise P10) and visualize the solution.

P16 Write a function

```
void incorporateInhomogeneousDirichletBC (const Mesh& mesh,
                                const Vector& ug, SparseMatrix& K, Vector& b);
```

that incorporates the inhomogenenous Dirichlet boundary conditions `ug` into the system matrix `K` and the load vector `b`. Here `ug` is a vector of the same size as `b` carrying the prescribed Dirichlet values (other values are ignored).

Hint: Ensure that the entries in `ug`, that do not correspond to Dirichlet values are set to zero. The modification of the load vector `b` can be done by

$$\mathtt{b}[i] = \begin{cases} \mathtt{ug}[i], & i \text{ corresponds to Dirichlet node} \\ b[i] - (\mathtt{K} * \mathtt{ug})[i], & \text{else} \end{cases}$$

After that, in order to modify `K`, proceed as in Exercise P14.

P17 Solve the finite element system corresponding to (3.16) with $f(x_1, x_2) = 20\pi^2 \sin(2\pi x_1) \sin(4\pi x_2)$ and $g(x_1, x_2)$ given by

$$g(x_1, x_2) = \begin{cases} 0, & x_2 = 1 \vee x_1 = 1 \\ (1 - x_1), & x_2 = 0 \\ (1 - x_2), & x_1 = 0 \end{cases}$$

for a suitably refined mesh (see exercise P10) and visualize the solution.

Let's consider Robin boundary conditions of the type

$$\frac{\partial u}{\partial N} := \lambda \frac{\partial u}{\partial n} = \kappa(u_0 - u) = g_3 - \kappa u.$$

for given λ, κ and u_0 and the normal derivative n.

P18 Let $e \subset \Gamma_R$ be element edges on the Robin boundary with the two endpoints $x^{(e,1)}$ and $x^{(e,2)}$. Let the reference edge be $\Delta = (0, 1)$ with the corresponding nodal basis functions $p^{(0)}(\xi) = 1 - \xi$ and $p^{(1)}(\xi) = \xi$. Write a function

```
void calcRobinElMat (const Vec<2>& x0, const Vec<2>& x1,
                                ScalarField kappa, Mat<2, 2>& elMat);
```

that computes the element Robin matrix K

$$K^e_{\alpha\beta} = \int_e \kappa(x) p^{(e,\alpha)}(x) p^{(e,\beta)}(x) dx = \int_\Delta \kappa(x_e(\xi)) p^{(\alpha)}(\xi) p^{(\beta)}(\xi) \det(J_e) d\xi$$

using the quadrature rule on $\Delta = (0, 1)$ given by

$$\int_\Delta g(\xi) d\xi \approx \frac{1}{6} \left[g(0) + 4g(0.5) + g(1) \right].$$

Show that this quadrature rule is exact for $g \in P_3$.

Hint: In order to get $x_e(\xi)$, implement a class modelling the affine linear transformation for edges , i.e. in 1D (compare P02, P03 and NumPDE-Tutorial).

P19 Write a function

```
void incorporateRobinBC (const Mesh& mesh, ScalarField kappa,
                         ScalarField u0, SparseMatrix& K, Vector& b);
```

that incorporates the Robin boundary conditions into the system matrix `K` and the load vector `b`.

Hint: Loop over all segments of the mesh and search for those marked as Robin (use `bcSegments[i] == BC_ROBIN`) and reuse the function from the previous Exercise P18 to add the local contributions to the stiffness matrix.

Hint: For the contribution corresponding to $\int_{\Gamma_R} g_3(x)\, v(x)\, ds$, proceed as for the Neumann Boundary (see P11).

L_2-error and H^1-error

P20 Write a function

```
double calcElErrorL2 (const Point2D& p0, const Point2D& p1,
                      const Point2D& p2, ScalarField exact,
                      double v0, double v1, double v2);
```

that approximates the element L^2-error $\|v - v_h\|_{L^2(\delta_r)}$, where `exact`$=v$ and $v_h(x_{\delta_r}(\xi)) = \sum_{\alpha \in A} v^{(r,\alpha)}\, p^{(\alpha)}(\xi)$ with `v0`$=v^{(r,1)}$ etc.

Hint: Use the quadrature rule from Exercise P05 to approximate

$$\|v - v_h\|^2_{L^2(\delta_r)} \;=\; \int_{\delta_r} |v(x) - v_h(x)|^2\, dx \;=\; \int_\Delta |v(x_{\delta_r}(\xi)) - v_h(x_{\delta_r}(\xi))|^2\, |\det J_{\delta_r}|\, d\xi$$

P21 Write a function

```
double calcElErrorH1 (const Point2D& p0, const Point2D& p1,
                      const Point2D& p2,
                      ScalarField Dx1exact, ScalarField Dx2exact,
                      double v0, double v1, double v2);
```

that approximates the element H^1-error $|Dv - \nabla v_h|_{L^2(\delta_r)}$, where $Dv = \nabla v = (\frac{\partial v}{\partial x_1}, \frac{\partial v}{\partial x_2})^T$, with `Dx1exact`=$\frac{\partial v}{\partial x_1}$, `Dx2exact`=$\frac{\partial v}{\partial x_2}$ and $v_h(x_{\delta_r}(\xi)) = \sum_{\alpha \in A} v^{(r,\alpha)}\, p^{(\alpha)}(\xi)$ with `v0`=$v^{(r,1)}$ etc.

Hint: Use the quadrature rule from Exercise P05 to approximate

$$|v - v_h|^2_{H^1(\delta_r)} \;=\; \int_{\delta_r} |Dv(x) - \nabla_x v_h(x)|^2 \, dx \;=\; \int_{\Delta} |Dv(x_{\delta_r}(\xi)) - J_r^{-T} \nabla_\xi v_h(x_{\delta_r}(\xi))|^2 \, |\det J_{\delta_r}| \, d\xi$$

P22 Write a function

```
double calcErrorL2 (const Mesh& mesh, ScalarField exact,
                    const Vector& solution);
```

that approximates the global L^2-error $\|v - v_h\|_{L^2(\Omega)}$, where `exact`=$v$ and `solution`=v_h.

Hint: use `calcElErrorL2` in a loop over all elements.

Show that $u(x_1, x_2) = \frac{1}{4} \cos(2\pi\, x_1) \cos(4\pi\, x_2)$ is the unique solution of (3.14) (see Tutorial 08, Exercise P10). Compute $\|u - u_h\|_{L^2(\Omega)}$ for each finite element solution u_h from Exercise P10 for the different meshes.

P23 Write a function

```
double calcErrorH1 (const Mesh& mesh, ScalarField exact,
                    ScalarField Dx1exact, ScalarField Dx2exact,
                    const Vector& solution);
```

that approximates the global H^1-error $\|v - v_h\|_{H^1(\Omega)}$, where `exact`=$v$, `Dx1exact`=$\frac{\partial v}{\partial x_1}$, `Dx2exact`=$\frac{\partial v}{\partial x_2}$ and `solution`=v_h.

Hint: use `calcElErrorL2` and `calcElErrorH1` in a loop over all elements.

Compute $\|u - u_h\|_{H^1(\Omega)}$ for each finite element solution u_h from Exercise P10 for the different meshes.

The CHIP-Problem

Recall the CHIP-Problem from the lecture (T08a, T08b, T09)!

P24 Prepare the initial mesh for the CHIP problem as proposed on T09 in your mesh-format, taking care of the appropriate boundary conditions.

Hint: If possible use symmetric reduction.

P25 Modify your functions from P04, P06 and P07, such that you can assemble the stiffness matrix `K` according to the bilinear form

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

where $\lambda(x)$ and $a(x)$ are given coefficient functions.

P26 Solve the finite element system corresponding to the CHIP problem on T08a with the parameter setting of T08b for the initial mesh of P24. Solve the same system for uniformly refined meshes with $h_0/h = 2, 4, 8, 16$ and visualize the solution.

Hint: For incorporating the BC, use the following order: First natural BC, then essential BC.

A posteriori error estimates

P27* Implement the residual error estimator for the CHIP-problem as derived in Section 3.6.2 of the lecture (see exercise in Tutorial 12).

P28* Compute the residual error for the CHIP-problem for uniformly refined meshes with $h_0/h = 2, 4, 8, 16$ and visualize the error on each element!