Course Project – Part II

NUMN33/FMNN45(F): Numerical Methods for PDEs

Eskil Hansen, Robert Klöfkorn

This is Part II of the overall course project which is based on DUNE and DUNE-FEM in your own Python environment.
In this part, we consider elliptic partial differential equations and investigate various properties. The project consists of both theoretical and practical exercises (tasks).

Please describe you solutions and observation in report form. For the practical exercises it should include

- a description of the problem considered,
- a description of your test cases in such a way so that they are reproducible, and
- a description of your tests, i.e. what is tested and why ...,
- your own interpretation of the results.
- Include important parts of your code as code snippets or append the entire code at the of the report.

Displaying code in Latex documents can, for example, be done like in this template: https://gitlab.maths.lu.se/robertk/thesislatextemplate. Many other examples can be found online.

Grading:

- All "non-living" and "non-generative AI" help is allowed, as long as proper references are given and you are able to explain the results during the oral exam. The use of generative AI is discouraged.
- **NUMN33:**
  - Minimum of 4 points in each section (theoretical/practical) and minimum 12 points in total for grade **G**.
  - Grade **G** and a minimum of 18 points total for a grade **VG**.
- **FMNN45:**
  - 4 points in each section (theoretical/practical) and minimum 12 points in total for a grade **3**.

  - Grade **G** and a minimum 16 points total for a grade **4**.
  - Grade **G** and a minimum 20 points total for a grade **5**.
- **FMNN45F:**
  - 6 points in each section (theoretical/practical) and minimum 16 points in total for grade **G**.

This assignment has **8 tasks** and **24 points** in total.

Theoretical Exercises

Task 1 (6 pts)

Let $\Omega := (-1, 1)^2 \subset \mathbb{R}^2$ and consider the elliptic problem

$$\begin{aligned} -\nabla \cdot (\alpha \nabla u) + \gamma u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned} \tag{1}$$

1. State the weak formulation of (1). (1 pt)
2. Which functions are reasonable to test with? (2 pts)
3. Choose two suitable (non-constant) functions α and γ. Prove that for your choice of functions and for every f in $H_0^1(\Omega)^*$ the weak formulation of (1) has a unique solution u in $H_0^1(\Omega)$. (3 pts)

Task 2 (6 pts)

Consider the FEM discretization of equation (1), and denote the FEM approximation by u_h and the exact solution by u.

1. Prove that $||u_h||_{H^1(\Omega)} \leq C||u||_{H^1(\Omega)}$. (3 pts)
2. Let $\alpha = \gamma = 1$. (3 pts) Prove that u_h is then the best possible approximation of u in the FEM space S_{h0} with respect to the norm $||\cdot||_{H^1(\Omega)}$ i.e.,

$$||u - u_h||_{H^1(\Omega)} = \inf_{\varphi \in S_{h0}} ||u - \varphi||_{H^1(\Omega)}.$$

Practical Exercises

A projection into a finite element space

The following requires the assembly of a finite element matrix (the mass matrix) and a right hand side. We use linear Lagrange shape functions.
Given a linear Finite Element space V_h we are looking for the L^2 projection

$$u_h(x) = \sum_k u_k \varphi_k(x)$$

which is the solution of

$$\int_\Omega u_h \varphi_i = \int_\Omega u \varphi_i, \qquad \forall \varphi_i \in V_h.$$

We assume that on an element E we have

$$\varphi_{g_E(k)}(x) = \hat{\varphi}_k(F_E^{-1}(x))$$

for $k = 0, 1, 2$ and where g_E denotes the local to global dof mapper and F_E is the reference mapping (see Lecture 4 and 6).
So we need to compute

$$M_{kl}^E := \int_{\hat{E}} |DF| \hat{\varphi}_k \hat{\varphi}_l \ , \qquad b_l^E := \int_E u \varphi_l \ ,$$

and distribute these into a global matrix and right hand side, respectively.

Task 3 (3 pts)

Write a class `LinearLagrangeSpace` that provides 3 methods:

```
class LinearLagrangeSpace:
    def __init__(self,view):
        self.view = view
        # TODO: Create a mapper for vertex indices
        self.mapper = see Lecture notes
        self.localDofs = 3
        self.points = numpy.array( [ [0,0],[1,0],[0,1] ] )
    def evaluateLocal(self, x):
        # TODO: Return a numpy array with the evaluations
        # of the 3 basis functions in local point x
        # return numpy.array( [] )
        pass
    def gradientLocal(self, x):
        # TODO: Return a numpy array with the evaluations
        # of the gradients of the 3 basis functions in local point x
        # return numpy.array( dbary )
        pass
```

The right hand side and matrix assembly

We need to iterate over the grid, construct the local right hand side and the local system matrix. After finishing the quadrature loop we store the resulting local matrix in a structure provided by the Python package `scipy`. There are many different storage structures available - we use the so called 'coordinate' (COO) matrix format which requires us construct three vectors, one to store the column indices, one for the row indices, and one for the values. The convention is that entries appearing multiple times are summed up - exactly as we need it. So after computing the local matrix and right hand side vector M^E we store the values M^E_{kl} into the values vector $v_{start+3l+k} = M^E_{kl}$ and the associated global indices $c_{start+3l+k} = g_E(k)$ and $r_{start+3l+k} = g_E(l)$.

Task 4 (4 pts)

Implement a function that assembles the system matrix and the forcing term. For this we will use the `LinearLagrangeSpace`. Use the attribute mapper from the `LinearLagrangeSpace` for index mapping. Remember that the gradients of the basis functions obtained from the `LinearLagrangeSpace` need to be converted to physical space using the `jacobianInverseTransposed` of the geometry.

```
def assemble(space,force):
    # storage for right hand side
    rhs = numpy.zeros(len(space.mapper))

    # storage for local matrix
    localEntries = space.localDofs
    localMatrix = numpy.zeros([localEntries,localEntries])

    # data structure for global matrix using coordinate (COO) format
    globalEntries = localEntries**2 * space.view.size(0)
    value = numpy.zeros(globalEntries)
    rowIndex, colIndex = numpy.zeros(globalEntries,int),
        numpy.zeros(globalEntries,int)

    # TODO: implement assembly of matrix and forcing term
    ...

    # convert data structure to compressed row storage (csr)
    matrix = scipy.sparse.coo_matrix((value, (rowIndex, colIndex)),
                         shape=(len(space.mapper),len(space.mapper))).tocsr()
    return rhs,matrix
```

The main part of the code

Construct the grid and a grid function for the right hand side, compute the projection and plot on a sequence of global grid refinements.

```
# First construct the grid
domain = cartesianDomain([0, 0], [1, 1], [10, 10])
view   = aluConformGrid(domain)
```

```python
# then the grid function to project
@gridFunction(view, name="u_ex", order=3)
def u(p):
    x,y = p
    return numpy.cos(2*numpy.pi*x)*numpy.cos(2*numpy.pi*y)
u.plot(level=3)

# and then do the projection on a series of globally refined grids
for ref in range(3):
    space  = LinearLagrangeSpace(view)
    print("number of elements:",view.size(0),"number of dofs:",len(space.mapper))

    rhs,matrix = assemble(space, u)
    dofs = scipy.sparse.linalg.spsolve(matrix,rhs)
    @gridFunction(view, name="u_h", order=1)
    def uh(e,x):
        indices = space.mapper(e)
        phiVals = space.evaluateLocal(x)
        localDofs = dofs[indices]
        return numpy.dot(localDofs, phiVals)
    uh.plot(level=1)
    view.hierarchicalGrid.globalRefine(2)
```

Task 5 (3 pts)

Compute some error of the projection, i.e., in maximum difference between u_h and u at the center of each element, or some L^2 type error over the domain, e.g.,

$$\sqrt{\int_\Omega |u - u_h|^2}.$$

What is the Experimental order of convergence (EOC)?

This is a simple procedure to test if a numerical scheme works. Assume, for example, that one can prove that the error e_h on a grid with grid spacing h satisfies

$$e_h \approx Ch^p$$

for some constant $C > 0$, then

$$\log \frac{e_h}{e_H} \approx p \log \frac{h}{H}$$

which can be used to get a good idea about the convergence rate p using the errors computed on two different levels with grid spacing h and H of a hierarchical grid.

Task 6 (2 pts)

Compute the H^1 norm of the error $||e_h||_{H^1(\Omega)}$ and again look at EOCs. Recall how the local basis functions are defined and use the chain rule, the required method on the element's geometry is `jacobianInverseTransposed(x)` ...

Extra Task 7 (2 bonus pts in case you want an extra challenge. Not mandatory!)

Implement an interpolation (e.g. as a method on the `LinearLagrangeSpace` class) and compare the errors of the interpolation with the errors/EOCs you computed for the projection.

Extra Task 8 (2 bonus pts in case you want an extra challenge. Not mandatory!)

Add a class with quadratic finite elements and look at the errors/EOCs.

Lycka till!