NTNU
Kunnskap for en bedre verden

Department of Mathematics

TMA4220 - Finite Element Method

Project 2: Improved Code

Author:
Anne Asklund
Jasper Steinberg
Paulius Cebatarauskas

18th August 2024

1 Introduction

In this project we have chosen task 4, namely to improve the code from Project 1. We start by time-optimizing our existing code to allow for h-refinement. Then we evaluate our errors in energy-norm. Afterwards, we aim towards machine precision as we implement $p = 2$ elements for p-refinement. Finally we test our new method on the L-shape domain.

2 Time optimization

Our codebase from the first project was created with the goal of just giving us a correct approximation of the exact solution to a Poisson problem. Thus, no emphasis was placed on the time optimization of the code and hence our code was inefficient and major modifications had to be made to be able to explore behaviour for finer meshes. Our hypothesis from the beginning was that explicit for-loops in Python are extremely slow (TST 2022). Therefore, since we know that NumPy is very fast compared to using explicit for-loops, our goal was to get rid of as many for-loops as possible and vectorize our code. In addition, we have used the cProfile package in Python to profile our code and find its slowest components. We sorted the profiler output by total accumulated time and then tried to optimize those parts of the code.

In the beginning we saw that quadrature was called many times and used a lot of time. For the code which implemented $p = 1$ elements, we noticed that it was unnecessary to call the quadrature method as our integrand was constant, since the gradient of a linear function is constant. Another thing we discovered was that NumPy's cross product method was very slow for small matrices, so we implemented our own version which was about 20 times faster. This was discovered by accident in a random stackoverflow question. Also, when we first needed the quadratures, we just computed them where they were needed, and implemented only the three point Gauss quadrature, as that was supposed to integrate all of our polynomials exactly. Another mistake, that saved us a lot of time later was to run the GetDisc method only once in solver and then supply its output to the stiffness matrix and load vector methods, in stead of calling the GetDisc method once in each of the three methods. This became a problem for N approaching one million.

Another great improvement came from adopting the use of sparse matrices. Our first use of this was to use a sparse solver, when solving the linear system $AU = F$. This upgrade was only a partial solution, as we ran into memory allocation problems when trying to build the matrix A using NumPy as it had trouble to initialize it using np.zeros((N,N)), for $N > 20\ 000$. Also, in this case our code used a lot of time converting our NumPy array to a sparse matrix and the background method that checked whether the entries in A were zero used majority of the runtime. Thus, we decided to build the stiffness matrix as a sparse matrix, which resulted in great improvements to the code by allowing us to run the code with N of about 100 000 in under a minute.

In the beginning we had triple for-loops in our code, which was highly inefficient. We vectorized this code one for-loop at a time, beginning with the innermost for-loop. This was the easiest implementation. Instead of changing the values in A or F one value at a time you would change multiple values at once. The second for-loop was more challenging to implement as it required transposing and reshaping of the matrix, and outer multiplications to make everything work. The last for-loop required a full vectorization of our code, which meant that we had to create a cross product which computed the areas for each element all at once. This was easy. The harder part was to set up the nodal basis functions and their derivatives, then multiple them together and reshape them. This was especially difficult, since we had to work with three-dimensional arrays which was unfamiliar to us in the beginning. Even worse, was doing this whole procedure for $p = 2$ elements as that resulted in four dimensional arrays, where the fourth dimension is due to the evaluation of functions at the quadrature points. In general, after all for-loops were eradicated, it was hard to find any more things to improve. Also, we were at such a stage where most of the time was spent generating the mesh and actually solving the sparse linear system of equations. In addition, for $N = 2\ 000\ 000$ and $p = 1$ elements our PC-s ran into memory errors again. So at that point we were satisfied.

3 Error analysis

Since our solution should be the best approximation in our solution space when measured in the error is energy norm, in our error analysis we will only consider errors measured in the energy norm. The energy norm $||u||_E$ is defined as the H1-seminorm $|u|_{H1}$ where

$$||u||_E = |u|_{H1}^2 = \int_\Omega \nabla(u) \cdot \nabla(u) dA \tag{1}$$

Note that the energy norm is also equal to

$$||u||_E = \sqrt{a(u,u)} \tag{2}$$

where $a(\cdot,\cdot)$ is the bilinear form.

3.1 From task 1 and 2 in Project 1

In Project 1 task 1 we solved the Poisson equation in 2 dimensions on the disc, with both Dirichlet boundary conditions (3) and mixed boundary conditions (4). These problems are shown below.

$$\begin{cases} \nabla^2 u(x,y) = -f(x,y), & (x,y) \in \Omega, \\ u(x,y) = 0, & (x,y) \in \partial\Omega \end{cases} \tag{3}$$

$$\begin{cases} \nabla^2 u(x,y) = -f(x,y), & (x,y) \in \Omega, \\ u(x,y) = 0, & (x,y) \in \partial\Omega_D, \\ \frac{\partial u(x,y)}{\partial n} = g(x,y), & (x,y) \in \partial\Omega_N \end{cases} \tag{4}$$

Note that $\Omega = \{(x,y) \in \mathbb{R}^2 \ : \ x^2 + y^2 \leq 1\}$ is the unit disc, $\partial\Omega = \{(x,y) \in \mathbb{R}^2 \ : \ x^2 + y^2 = 1, \}$, $\partial\Omega_D = \{(x,y) \in \mathbb{R}^2 \ : \ x^2 + y^2 = 1, \ y < 0\}$, and $\partial\Omega_N = \{(x,y) \in \mathbb{R}^2 \ : \ x^2 + y^2 = 1, \ y > 0\}$. In both of these problems

$$f(x,y) = -8\pi \cos(2\pi(x^2+y^2)) + 16\pi^2(x^2+y^2)\sin(2\pi(x^2+y^2))$$

and for (4)

$$g(x,y) = 4\pi\sqrt{x^2+y^2}\cos(2\pi(x^2+y^2)) + 16\pi^2(x^2+y^2)\sin(2\pi(x^2+y^2)).$$

After a routine (errorestimate) which loops over all elements and implements the energy norm elementwise for the error, the convergence plot for which can be seen in Figure 1. The value of h has been approximated by $h = \frac{1}{\lfloor\sqrt{\frac{N}{\pi}}\rfloor}$, as taken from the mesh generator getdisc. Note that both the Dirichlet and Neumann problems have a convergence rate close to one.

3.2 Comparison to reference solution

To show how to calculate the error with respect to a reference solution we attempt to a problem on the square $[-1,1]^2 \subset \mathbb{R}^2$ with homogeneous Dirichlet boundary conditions. For this domain, we chose the exact solution

$$u(x,y) = (x-1)(x+1)(y-1)(y+1) \tag{5}$$

such that our formulation (3) has $f = -2(x^2+y^2) + 4$. The choices for N were such that $h = \frac{\sqrt{2}}{3}, \frac{\sqrt{2}}{6}, \frac{\sqrt{2}}{12}, \frac{\sqrt{2}}{24}$ and $h_{ref} = \frac{\sqrt{2}}{48}$ The result of this comparison can be seen in Figure 2.

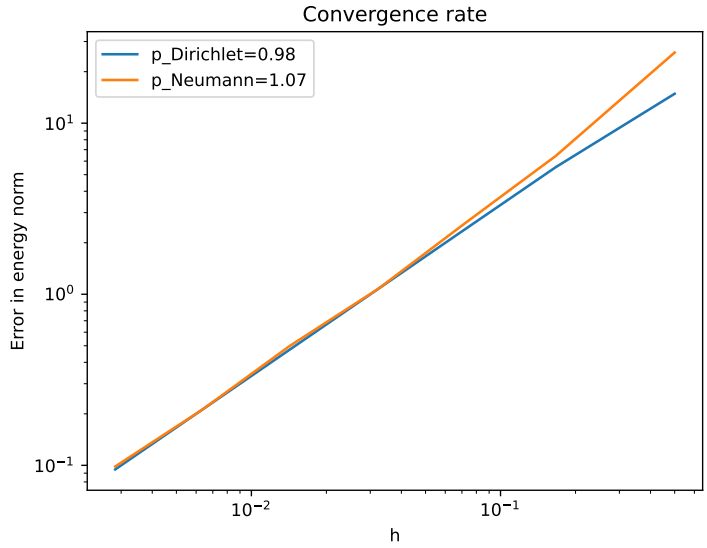

Figure 1: Convergence plot in the energy-norm for our numerical solution to the Dirichlet and Neumann problem.

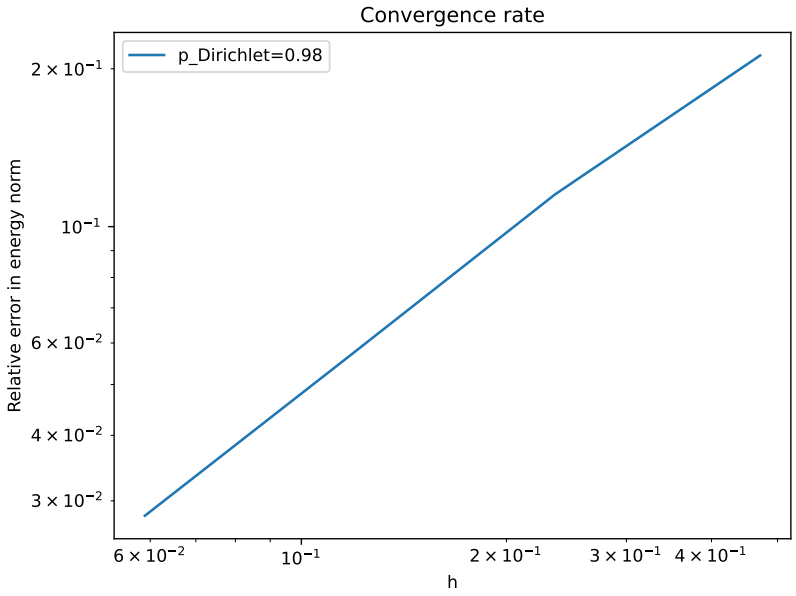

Figure 2: Convergence plot in the energy-norm for our numerical solution to the Dirichlet problem compared to the reference solution.

4 Machine precision

Our previous upgrades to the code, has let us run our code with very small values for h. In order to get even smaller errors, our goal was to implement p refinement, i.e., implement $p = 2$ elements. The whole algorithm stays the same as in the case for $p = 1$ elements. The only difference is that we now will have six nodal basis functions instead of three as was the case for $p = 1$ elements. Our nodal basis functions are

$$\begin{aligned}
N_1(L_1, L_2, L_3) &= L_1(2L_1 - 1) \\
N_2(L_1, L_2, L_3) &= L_2(2L_2 - 1) \\
N_3(L_1, L_2, L_3) &= L_3(2L_3 - 1) \\
N_4(L_1, L_2, L_3) &= 4L_1L_2 \\
N_5(L_1, L_2, L_3) &= 4L_2L_3 \\
N_6(L_1, L_2, L_3) &= 4L_1L_3
\end{aligned}$$

where L_1, L_2, L_3 is our barycentric coordinates (CSU 2022). These coordinates can be expressed using the cartesian coordinates as follows:

$$\begin{aligned}
L_1 &= \frac{1}{2A}\left(\alpha_1 + \beta_1 x + \gamma_1 y\right), \\
L_2 &= \frac{1}{2A}\left(\alpha_2 + \beta_2 x + \gamma_2 y\right), \\
L_3 &= \frac{1}{2A}\left(\alpha_3 + \beta_3 x + \gamma_3 y\right),
\end{aligned}$$

where the constants $\alpha_i, \beta_i, \gamma_i$ for $i \in \{1, 2, 3\}$ are to be decided and A is the area of the triangle spanned by the three points $1, 2, 3$. Given a triangle with vertexes $1, 2, 3$ with their respective coordinates in the cartesian plane given by $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ we can find explicit formulas for $\alpha_i, \beta_i, \gamma_i$. They are given as follows:

$$\begin{aligned}
\alpha_1 &= x_2y_3 - y_2x_3 \qquad \beta_1 = y_2 - y_3 \qquad \gamma_1 = x_3 - x_2 \\
\alpha_2 &= x_3y_1 - y_3x_1 \qquad \beta_2 = y_3 - y_1 \qquad \gamma_2 = x_1 - x_3 \\
\alpha_3 &= x_1y_2 - y_1x_2 \qquad \beta_3 = y_1 - y_2 \qquad \gamma_3 = x_2 - x_1.
\end{aligned}$$

The reason for implementing the nodal basis functions explicitly is that the alternative is to solve a linear system in order to get the coefficients $\alpha_i, \beta_i, \gamma_i$. But there is no point in solving a linear system of equations to get some coefficients if we already know the formula for the coefficients. This is a small system of equations which is solved quickly, but the problem appears when you have a lot of elements in your mesh and you are solving a system of equations for each element. That quickly becomes a lot of system solving when working with values of $N = 2000000$.

The only other change we had to implement from $p = 1$ to $p = 2$ elements, was to implement extra nodes at the midpoint of each simplex edge. We started indexing them from the index of the last vertex from the GetDisc function. Figure 3 shows these nodes for the mesh generated by GetDisc for $N = 5$.

The largest system we solved with this method was the Dirichlet problem from project 1 with $N = 300\ 000$ for the GetDisc mesh-creator. In this case we got a relative error of 0.0001116 in the energy-norm for a mesh-size of about 0.00324, and it took 960.47 seconds to run the system. The maximum deviation between the exact solution and the numerical solution was 0.0000165.

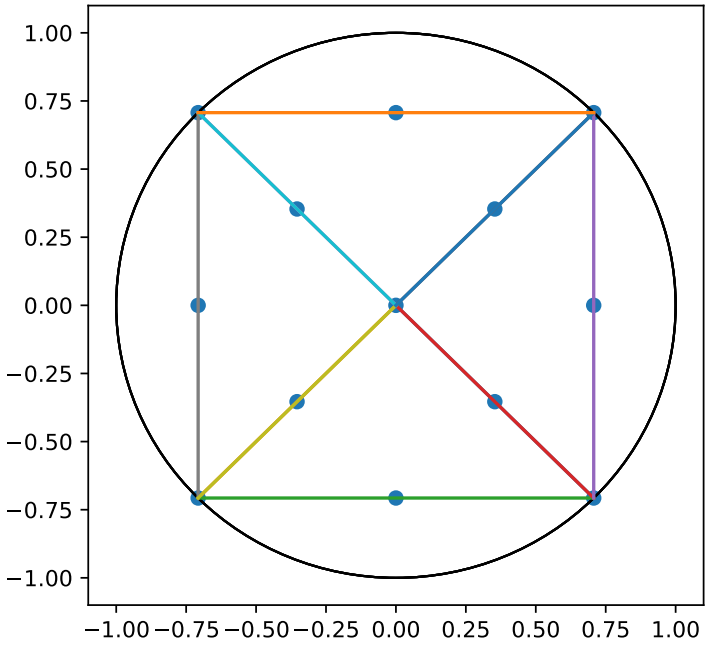

Figure 3: Mesh for $p = 2$ elements on the disc for $N = 5$.

5 The L-shape

For the L-shape we consider the problem

$$\begin{aligned} \Delta u &= 0 \text{ in } \Omega \\ u &= 0 \text{ on } \Gamma_D \\ \frac{\partial u}{\partial n} &= g \text{ on } \Gamma_N \end{aligned}$$

with the known exact solution

$$u(r,\theta) = r^{2/3}\sin\left(\frac{2\theta + 2\pi}{3}\right),\ \theta \in \left[\frac{\pi}{2}, 2\pi\right],\ r^2 = x^2 + y^2. \tag{6}$$

To compute g we first compute some partial derivatives;

$$\begin{aligned} \frac{\partial u}{\partial \theta} &= \frac{2}{3}r^{2/3}\cos\left(\frac{2}{3}(\theta+\pi)\right), \\ \frac{\partial u}{\partial r} &= \frac{2}{3}r^{-1/3}\sin\left(\frac{2}{3}(\theta+\pi)\right), \\ \frac{\partial r}{\partial x} &= \frac{x}{r}, \qquad \frac{\partial r}{\partial y} = \frac{y}{r}, \\ \frac{\partial \theta}{\partial x} &= -\frac{y}{r^2} \quad \text{and} \quad \frac{\partial \theta}{\partial y} = \frac{x}{r^2}. \end{aligned}$$

Using the chain-rule we have;

$$\begin{aligned} \frac{\partial u}{\partial x} &= \frac{\partial u}{\partial r}\frac{\partial r}{\partial x} + \frac{\partial u}{\partial \theta}\frac{\partial \theta}{\partial x} = \frac{2}{3}r^{-4/3}\left[x\sin\left(\frac{2}{3}(\theta+\pi)\right) - y\cos\left(\frac{2}{3}(\theta+\pi)\right)\right] \\ \frac{\partial u}{\partial y} &= \frac{\partial u}{\partial r}\frac{\partial r}{\partial y} + \frac{\partial u}{\partial \theta}\frac{\partial \theta}{\partial y} = \frac{2}{3}r^{-4/3}\left[y\sin\left(\frac{2}{3}(\theta+\pi)\right) + x\cos\left(\frac{2}{3}(\theta+\pi)\right)\right] \end{aligned}$$

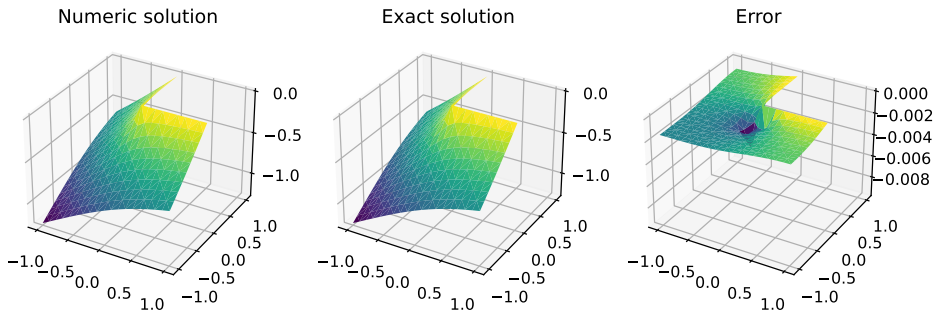

Figure 4: Numerical solution, exact solution and error for the homogeneous Poisson problem on the L-shape domain.

Now let the unit normals be $n_1 = (0, 1)$, $n_2 = (1, 0)$, $n_3 = (0, -1)$, $n_4 = (-1, 0)$. In addition, we know that $y = 1$ along the first Neumann boundary, $x = 1$ along the second Neumann boundary, $y = -1$ along the third Neumann boundary and lastly, $x = -1$ along the fourth Neumann boundary. This leads to the following expressions;

$$g_1(x) = \nabla u \cdot n_1\Big|_{y=1} = \frac{2}{3}(x^2+1)^{-2/3}\left[\sin\left(\frac{2}{3}\left(\arctan\left(x^{-1}\right)+\pi\right)\right) + x\cos\left(\frac{2}{3}\left(\arctan\left(x^{-1}\right)+\pi\right)\right)\right]$$

$$g_2(y) = \nabla u \cdot n_2\Big|_{x=1} = \frac{2}{3}(1+y^2)^{-2/3}\left[\sin\left(\frac{2}{3}\left(\arctan\left(y\right)+\pi\right)\right) - y\cos\left(\frac{2}{3}\left(\arctan\left(y\right)+\pi\right)\right)\right]$$

$$g_3(x) = \nabla u \cdot n_3\Big|_{y=-1} = -\frac{2}{3}(x^2+1)^{-2/3}\left[-\sin\left(\frac{2}{3}\left(\pi-\arctan\left(x^{-1}\right)\right)\right) + x\cos\left(\frac{2}{3}\left(\pi-\arctan\left(x^{-1}\right)\right)\right)\right]$$

$$g_4(y) = \nabla u \cdot n_4\Big|_{x=-1} = -\frac{2}{3}(1+y^2)^{-2/3}\left[-\sin\left(\frac{2}{3}\left(\pi-\arctan\left(y\right)\right)\right) - y\cos\left(\frac{2}{3}\left(\pi-\arctan\left(y\right)\right)\right)\right].$$

Figure 4 compares the numerical and exact solution of this problem. From the figure it is clear that the largest discrepancies between the numerical and exact solution happen in the origin, where the L-shape has a singularity.

Due to the singularity, where the internal angle of the L-shape is $\frac{3}{2}\pi$, we get subpar convergence. This can be seen in Figure 5, as the order of convergence is only 0.76, rather than the expected closer to 2. Thus we can see that the regularity of the domain is important for the convergence of our numerical schemes.

The solution on the finest grid (setting $N = 100$) we ran had a relative error of 0.00635, a mesh-spacing of 0.0047, and it took 166.5 seconds to run. A smaller error could have been found for an even finer mesh. This was not done due to time constraints. Another obvious avenue for reducing the error would be adaptive refinement of the mesh. On the L-shape there is a singularity at origo, where the error is big. One idea could be to refine the mesh near this singularity. In general, one might fight the meshes with the biggest errors and then refine those simplexes until the error in them are below a certain threshold. But, this approach would require a lot more work, and time was a constraint at the end of the day.

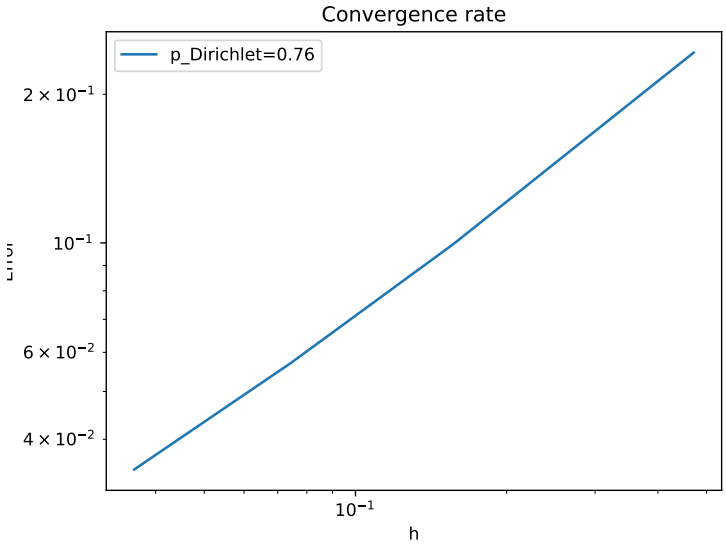

Figure 5: Convergence plot in the energy-norm for our numerical solution to the Poisson problem on the L-shape for $p = 2$ elements.

Bibliography

CSU (2022). *Section 11: HIGHER ORDER TWO DIMENSIONAL SHAPE FUNCTIONS, Higher Order Triangular Elements*. URL: https://academic.csuohio.edu/duffy_s/CVE_512_11.pdf (visited on 19th Nov. 2022).

TST (2022). *Python vs C++ Speed Comparison*. URL: https://www.youtube.com/watch?v=VioxsWYzoJk (visited on 12th Nov. 2022).