Numerical Techniques for Fluid Dynamics

MATH5453M FEM Numerical Exercises 3, 2024

Aly Ilyas
mmai@leeds.ac.uk

Due date: December 2024

Sources: Internet.

Problem Statement

Consider the Poisson system:

$$-\nabla^2 u = f \quad \text{on } (x,y) \in [0,1]^2 \tag{1a}$$
$$f(x,y) = 2\pi^2 \sin(\pi x)\cos(\pi y) \tag{1b}$$
$$u(0,y) = u(1,y) = 0 \tag{1c}$$
$$\frac{\partial u}{\partial y}(x,y)\Big|_{y=0} = \frac{\partial u}{\partial y}(x,y)\Big|_{y=1} = 0 \tag{1d}$$

where $u(x,y)$ is the variable or unknown, $f(x,y)$ is a given function, and the system satisfies Dirichlet and Neumann boundary conditions. The exact solution is:

$$u_e(x,y) = \sin(\pi x)\cos(\pi y).$$

Proof of the exact solution:
Let $u_e(x,y) = \sin(\pi x)\cos(\pi y)$. We aim to verify that $u_e(x,y)$ satisfies the given Poisson equation:

$$-\nabla^2 u = f, \quad \text{where } f(x,y) = 2\pi^2 \sin(\pi x)\cos(\pi y). \tag{1}$$

First, compute the partial derivatives of $u_e(x,y)$. The partial derivative with respect to x is given by:

$$\frac{\partial u_e}{\partial x} = \pi \cos(\pi x)\cos(\pi y), \tag{2}$$

and the second derivative with respect to x is:

$$\frac{\partial^2 u_e}{\partial x^2} = -\pi^2 \sin(\pi x)\cos(\pi y). \tag{3}$$

Similarly, compute the partial derivative with respect to y:

$$\frac{\partial u_e}{\partial y} = -\pi \sin(\pi x)\sin(\pi y), \tag{4}$$

and the second derivative with respect to y is:

$$\frac{\partial^2 u_e}{\partial y^2} = -\pi^2 \sin(\pi x)\cos(\pi y). \tag{5}$$

Next, calculate the Laplacian $\nabla^2 u_e$, which is the sum of the second derivatives with respect to x and y from (3) and (5):

$$\nabla^2 u_e = \frac{\partial^2 u_e}{\partial x^2} + \frac{\partial^2 u_e}{\partial y^2}. \tag{6}$$

Substituting the computed second derivatives, we have:

$$\nabla^2 u_e = -\pi^2 \sin(\pi x)\cos(\pi y) - \pi^2 \sin(\pi x)\cos(\pi y). \tag{7}$$

Simplifying, this becomes:

$$\nabla^2 u_e = -2\pi^2 \sin(\pi x)\cos(\pi y). \tag{8}$$

Now, substitute this result into the Poisson equation:

$$-\nabla^2 u_e = -(-2\pi^2 \sin(\pi x)\cos(\pi y)) = 2\pi^2 \sin(\pi x)\cos(\pi y). \tag{9}$$

Thus, we find that:

$$-\nabla^2 u_e = f(x, y), \tag{10}$$

where $f(x, y) = 2\pi^2 \sin(\pi x)\cos(\pi y)$, as given in the problem. This confirms that $u_e(x, y) = \sin(\pi x)\cos(\pi y)$ is indeed the exact solution of the Poisson equation.

Step 1

First, we need to consider the Poisson equation given in the problem. To derive the weak formulation, we first multiply equation (1a) by an arbitrary test function $w(x, y)$ and integrate over the domain $[0, 1]^2$. This yields:

$$\int_0^1 \int_0^1 \left(-\nabla^2 u - f\right) w \, dx \, dy = 0. \tag{11}$$

Next, we apply the divergence theorem to transform the term involving $-\nabla^2 u$. Using the identity $-\nabla^2 uw = \nabla w \cdot \nabla u - \nabla \cdot (w \nabla u)$, we obtain:

$$\int_0^1 \int_0^1 \nabla w \cdot \nabla u \, dx \, dy - \int_\Gamma w \mathbf{n} \cdot \nabla u \, d\Gamma - \int_0^1 \int_0^1 f w \, dx \, dy = 0, \tag{12}$$

where Γ is the boundary of the domain, and $\mathbf{n}$ is the outward unit normal vector.

Now, we consider the boundary integral. The boundary Γ can be divided into the horizontal boundaries (Γ_1) at $y = 0$ and $y = 1$, and the vertical boundaries (Γ_2) at $x = 0$ and $x = 1$ illustrated in Figure 1. On the horizontal boundaries, the outward normal vector is $\mathbf{n} = \pm \mathbf{e}_y$, and $\mathbf{n} \cdot \nabla u = \frac{\partial u}{\partial y}$. From the Neumann boundary condition (1d), $\frac{\partial u}{\partial y} = 0$ on $y = 0$ and $y = 1$. Therefore, the contribution from Γ_1 vanishes:

$$\int_{\Gamma_1} w \, \mathbf{n} \cdot \nabla u \, d\Gamma = 0. \tag{13}$$

On the vertical boundaries, the outward normal vector is $\mathbf{n} = \pm\mathbf{e}_x$, and $\mathbf{n} \cdot \nabla u = \frac{\partial u}{\partial x}$. From the Dirichlet boundary condition (1c), $u = 0$ on $x = 0$ and $x = 1$. To ensure compatibility, we require the test function $w(x, y)$ to satisfy the same boundary conditions as u, namely $w(0, y) = w(1, y) = 0$. Consequently, the contribution from Γ_2 also vanishes:

$$\int_{\Gamma_2} w\, \mathbf{n} \cdot \nabla u \, d\Gamma = 0. \tag{14}$$

With the boundary integral eliminated, the weak formulation (12) of the problem reduces to:

$$\int_0^1 \int_0^1 \nabla w \cdot \nabla u \, dx \, dy = \int_0^1 \int_0^1 f w \, dx \, dy, \quad \forall w \in V, \tag{15}$$

where V is the space of test functions satisfying the homogeneous Dirichlet boundary conditions.

Now, we consider the Ritz-Galerkin principle. Let $w(x, y) = \delta u(x, y)$, where $\delta u(x, y)$ is the variation of $u(x, y)$. Since u satisfies the Dirichlet boundary conditions, the variation δu also satisfies $\delta u(0, y) = \delta u(1, y) = 0$. Substituting $w = \delta u$ into the weak formulation, we obtain:

$$\int_0^1 \int_0^1 \nabla(\delta u) \cdot \nabla u \, dx \, dy = \int_0^1 \int_0^1 f \, \delta u \, dx \, dy. \tag{16}$$

This can be rewritten as:

$$\delta \int_0^1 \int_0^1 \left(\frac{1}{2} |\nabla u|^2 - f u \right) dx \, dy = 0. \tag{17}$$

Thus, the Ritz-Galerkin principle reduces to minimizing the functional:

$$J(u) = \int_0^1 \int_0^1 \left(\frac{1}{2} |\nabla u|^2 - f u \right) dx \, dy. \tag{18}$$

The condition for $J(u)$ to attain its minimum is $\delta J = 0$, which recovers the weak formulation (15):

$$\int_0^1 \int_0^1 \nabla w \cdot \nabla u \, dx \, dy = \int_0^1 \int_0^1 f w \, dx \, dy, \quad \forall w \in V.$$

This demonstrates that the test function $w(x, y)$ is the same as $\delta u(x, y)$.

Step 2

We let $u(x, y) \in V$, where V is the appropriate Sobolev space satisfying the boundary conditions. To approximate u using the finite element method (FEM), we introduce a finite-dimensional subspace $V_h \subset V$, spanned by linearly independent basis functions $\{\phi_j(x, y)\}_{j=1}^n$. The approximate solution $u_h \in V_h$ is expressed as:

$$u_h(x, y) = \sum_{j=1}^{N_n} \hat{u}_j \phi_j(x, y), \tag{19}$$

where $\hat{u}_j$ are unknown coefficients, j denotes the node number (ranging from 1 to N_n), and N_n is the total number of nodes in the finite element mesh. Each $\phi_j(x, y)$ is a global basis function associated with node j.

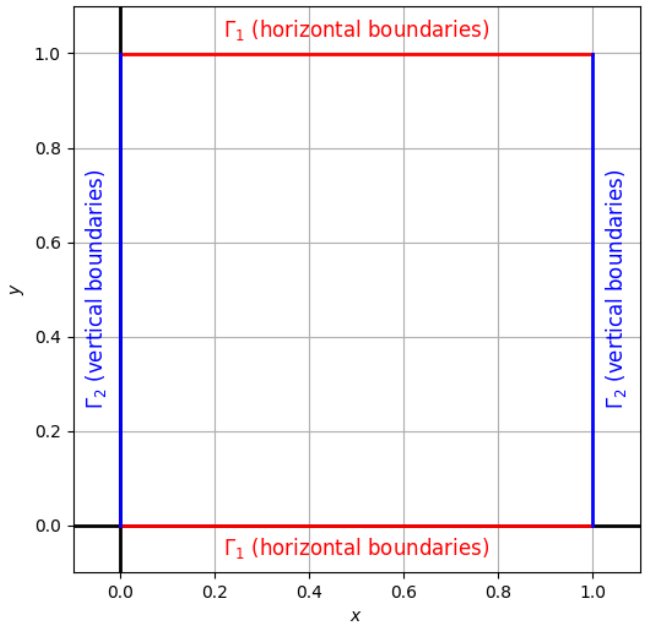


Figure 1: Illustration of the domain $[0,1]^2$ and its boundary Γ, which is composed of both vertical (Γ_2) and horizontal (Γ_1) boundaries. The vertical boundaries correspond to $x = 0$ and $x = 1$, where Dirichlet boundary conditions are applied. The horizontal boundaries correspond to $y = 0$ and $y = 1$, where Neumann boundary conditions are imposed. The diagram clearly shows the structure of the boundary and highlights the contributions of the boundary terms in the weak formulation.

The Ritz-Galerkin principle states that u_h satisfies the weak formulation:

$$\int_0^1 \int_0^1 (\nabla w \cdot \nabla u_h - wf)\, dx\, dy = 0 \quad \forall w \in V_h. \tag{20}$$

Choosing test functions $w = \phi_i$ from the same basis $\{\phi_j\}$, we obtain:

$$\int_0^1 \int_0^1 (\nabla \phi_i \cdot \nabla u_h - \phi_i f)\, dx\, dy = 0, \quad i = 1, \ldots, n. \tag{21}$$

where i is the index of the test function and n is the total number of basis functions. Substitute the expansion $u_h = \sum_{j=1}^{N_n} \hat{u}_j \phi_j$ into the equation:

$$\int_0^1 \int_0^1 \left(\nabla \phi_i \cdot \nabla \left(\sum_{j=1}^{N_n} \hat{u}_j \phi_j \right) - \phi_i f \right) dx\, dy = 0, \quad i = 1, \ldots, n. \tag{22}$$

Simplifying, using the linearity of the integral and gradient:

$$\sum_{j=1}^{N_n} \hat{u}_j \int_0^1 \int_0^1 \nabla \phi_i \cdot \nabla \phi_j \, dx\, dy = \int_0^1 \int_0^1 \phi_i f \, dx\, dy, \quad i = 1, \ldots, n. \tag{23}$$

Next, we define the stiffness matrix A and load vector b as:

$$A_{ij} = \int_0^1 \int_0^1 \nabla \phi_i \cdot \nabla \phi_j \, dx\, dy, \quad b_i = \int_0^1 \int_0^1 \phi_i f \, dx\, dy. \tag{24}$$

Then the equation becomes:

$$\sum_{j=1}^{N_n} A_{ij}\hat{u}_j = b_i, \quad i = 1, \ldots, n. \tag{25}$$

In matrix-vector form, this is written as:

$$A\hat{\mathbf{u}} = \mathbf{b}, \tag{26}$$

where $\hat{\mathbf{u}} = (\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_n)^T$ is the vector of unknown coefficients.

Step 4

The results from solving the system using Firedrake using **ex3modified.py** demonstrate the impact of mesh refinement (h-refinement) and polynomial degree (p-refinement) on solution accuracy. Mesh refinement involves decreasing the element size (Δx), which systematically improves accuracy for a fixed polynomial degree. For instance, with $p = 1$, the L^2 error decreases from 1.59301×10^{-3} for a 16×16 mesh to 2.50964×10^{-5} for a 128×128 mesh. Similarly, p-refinement, which increases the polynomial degree of the finite element basis functions, enhances the approximation accuracy within each element. For a 32×32 mesh, the L^2 error reduces from 4.00757×10^{-4} with $p = 1$ to 1.14821×10^{-11} with $p = 4$.

In terms of error, several h-p combinations yield equivalent magnitudes of error. For example, with a mesh resolution of 16×16 and $p = 3$, the error magnitude is roughly 10^{-9}, which is comparable to a mesh resolution of 64×64 with $p = 2$, also having an error magnitude of approximately 10^{-9}. Similarly, for a mesh resolution of 32×32 with $p = 3$, the error magnitude is around 10^{-10}, comparable to a mesh resolution of 128×128 with $p = 2$, which also has an error magnitude near 10^{-10}.

This occurs because higher polynomial degrees (p) increase the approximation power of the numerical solution within each element, compensating for coarser meshes (h). Conversely, finer meshes improve accuracy by reducing the size of each element, which can offset the use of lower-degree polynomials. As a result, certain h-p combinations achieve similar error magnitudes due to this trade-off between spatial resolution and polynomial approximation.

Interestingly, certain combinations of h-p refinements yield comparable accuracies, such as $p = 2$ on a 32×32 mesh (6.52773×10^{-8}) and $p = 1$ on a 64×64 mesh (1.00346×10^{-4}). This shows that p-refinement can achieve higher accuracy with fewer degrees of freedom compared to h-refinement. However, achieving optimal results requires balancing computational cost and accuracy by exploring equivalent (h, p) combinations. Additionally, an anomaly is observed for Method 2 with $p = 4$, where the error decreases from 16×16 to 32×32, but unexpectedly increases for finer meshes, contrary to expectations. This warrants further investigation into numerical stability or implementation issues for specific cases.

Mesh Resolution	Polynomial Degree (p)	L^2 Error Method 1	L^2 Error Method 2
16×16	1	1.59301×10^{-3}	1.59301×10^{-3}
16×16	2	1.04273×10^{-6}	1.04273×10^{-6}
16×16	3	3.38214×10^{-9}	3.37877×10^{-9}
16×16	4	3.86391×10^{-11}	2.66663×10^{-11}
32×32	1	4.00757×10^{-4}	4.00757×10^{-4}
32×32	2	6.52773×10^{-8}	6.52773×10^{-8}
32×32	3	1.06439×10^{-10}	1.05857×10^{-10}
32×32	4	1.14821×10^{-11}	4.32984×10^{-13}
64×64	1	1.00346×10^{-4}	1.00346×10^{-4}
64×64	2	4.08158×10^{-9}	4.08156×10^{-9}
64×64	3	5.35352×10^{-12}	3.36122×10^{-12}
64×64	4	4.94167×10^{-12}	4.81697×10^{-13}
128×128	1	2.50964×10^{-5}	2.50964×10^{-5}
128×128	2	2.55454×10^{-10}	2.55356×10^{-10}
128×128	3	2.19370×10^{-12}	2.66598×10^{-12}
128×128	4	2.43813×10^{-12}	1.94554×10^{-12}

Table 1: L^2 Errors for Various Mesh Resolutions and Polynomial Degrees

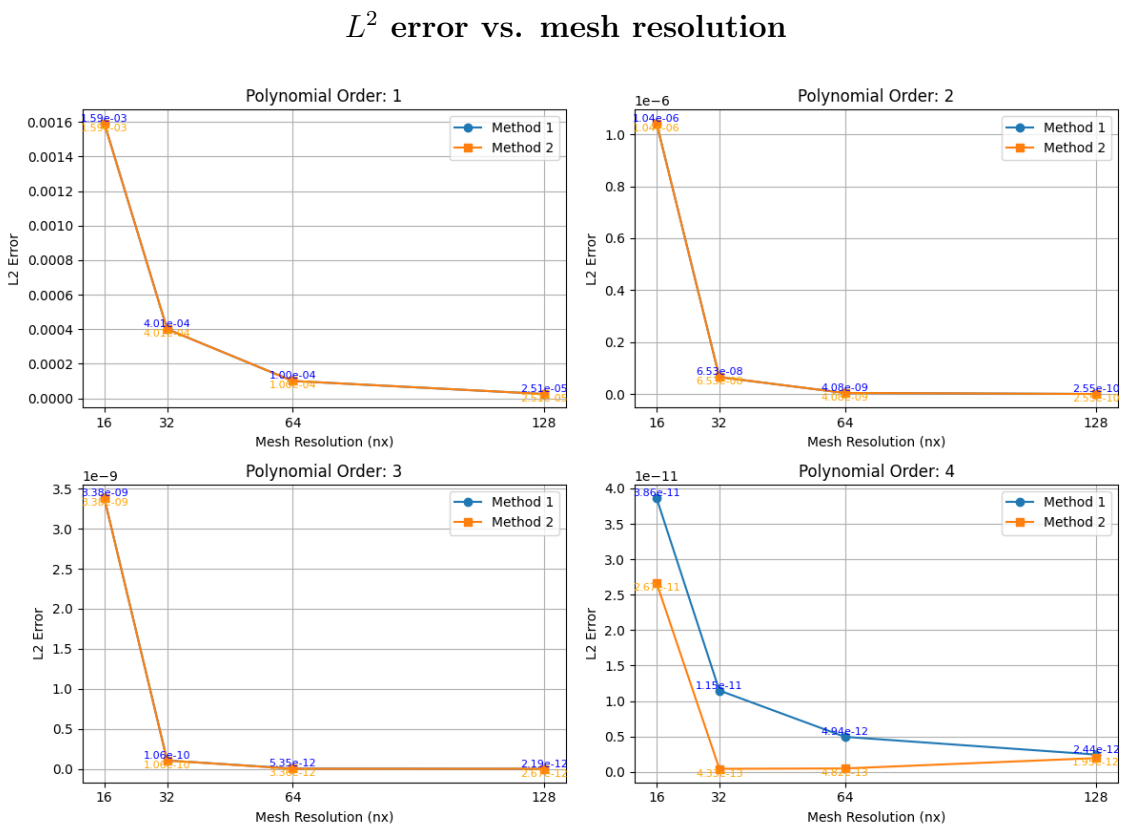


Figure 2: L^2 error vs. mesh resolution for different polynomial degrees (p). Blue: Method 1, Orange: Method 2. Labels indicate error values for each point. For Method 2 with $p = 4$, errors decrease from 16×16 to 32×32 but increase for finer mesh resolutions.

Step 5

In Firedrake, the procedure for solving the Poisson equation is implemented as follows:

a. Defining the Function Space

First, we define a finite element function space V_h using a predefined element (e.g., CG for continuous Galerkin) over a mesh of the unit square:

```
mesh = UnitSquareMesh(nx, nx, quadrilateral=True)
V = FunctionSpace(mesh, 'CG', p)
```

The line

```
mesh = UnitSquareMesh(nx, nx, quadrilateral=True)
```

creates a mesh representing the unit square $[0,1]\times[0,1]$, divided into $nx \times nx$ quadrilateral elements. This grid is used as the computational domain for finite element methods. The `UnitSquareMesh` function is part of Firedrake and takes the number of subdivisions along the x and y axes as arguments, represented by nx. The `quadrilateral=True` argument specifies that each element of the mesh will be a quadrilateral, as opposed to a triangular element.

The line

```
V = FunctionSpace(mesh, 'CG', p)
```

defines the function space V over the created mesh. The function space is a collection of functions that can be used to approximate solutions to the problem. Here, `'CG'` refers to the continuous Galerkin finite element method, which uses piecewise continuous polynomial functions for approximations. The degree of the polynomials is specified by p, which determines the polynomial order for the basis functions in the finite element space. This function space is then used to represent both the trial and test functions in the weak formulation of the problem.

b. Defining the Trial and Test Functions

The trial function u_h and test function w are both defined as functions in V:

```
u = Function(V)  # Trial function
w = TestFunction(V)  # Test function
```

In Firedrake, the **trial function** and **test function** are essential components of the finite element method used to solve partial differential equations. The trial function, denoted as u_h, is the function for which we seek an approximation over the finite element space V. It represents the unknown solution to the problem. The test function, denoted as w, is used to define the weak form of the equation, acting as a weighting function in the variational formulation. Together, the trial and test functions allow us to construct the system of equations that are solved to approximate the solution to the given problem.

c. Weak Formulation

The weak formulation is expressed in terms of variational forms using Firedrake's `inner` and `dx` operators. For the Poisson problem, the weak form is:

```
# Bilinear form (stiffness matrix)
a = inner(grad(u), grad(w)) * dx
# Linear form (load vector)
L = f * w * dx
u_1 = Function(V, name='u_1')
```

In the code, the expressions a and L represent the bilinear form and linear form, respectively, which are key components of the weak formulation of a differential equation in the finite element method in equation (15).

- The term $a = \text{inner}(\nabla u, \nabla w)\, dx$ represents the *bilinear form*, which corresponds to the discretized version of the equation's left-hand side. Here:
  - ∇u and ∇w are the gradients of the trial function u and test function w, respectively.
  - The $\text{inner}(\cdot, \cdot)$ function computes the dot product of the gradients, which measures how the solution varies in the direction of the test function.
  - The dx operator indicates integration over the computational domain. The result of this term is a *stiffness matrix*, which encodes the relationship between the trial and test functions over the mesh.
- The term $L = f \cdot w\, dx$ represents the *linear form*, corresponding to the right-hand side of the weak formulation. Here:
  - f is the source term or forcing function.
  - w is the test function, which weights the source term during integration.
  - The dx operator indicates integration over the domain. The result is a *load vector*, which contains the contributions of the source term f integrated against the test functions.

These forms are assembled into a system of linear equations that are solved to obtain the approximation of the solution u. The function u_1 will store the solution of the variational problem and can be evaluated on the mesh or used in further computations.

d. Boundary condition

The Dirichlet boundary conditions (1c) are applied at both boundaries $x = 0$ and $x = 1$ to ensure that the solution is fixed at zero on these boundaries. In Firedrake, the Dirichlet boundary conditions are defined as:

```
# Dirichlet boundary conditions
# are set to zero at both x=0 and x=1 boundaries.
bc_x0 = DirichletBC(V, Constant(0), 1)
bc_x1 = DirichletBC(V, Constant(0), 2)
```

where:

- V is the function space, containing the finite element functions.
- Constant(0) specifies that the solution is set to zero at the boundaries.
- The numbers 1 and 2 represent the left ($x = 0$) and right ($x = 1$) boundaries, respectively.

These boundary conditions ensure that the solution is fixed at zero at both $x = 0$ and $x = 1$.

e. Solving the System for Method 1

The system of equations $A\hat{u} = b$ is assembled and solved using Firedrake's solver:

```
solve(a == L, u_1,
    solver_parameters={'ksp_type': 'cg', 'pc_type': 'none'},
    bcs=[bc_x0, bc_x1])
```

In the following line, the solve function solves the weak form of the Poisson equation $a(u, v) = L(v)$ for the unknown function u_1. The solver will then compute the solution u_1 that satisfies the weak form of the Poisson equation, with the given boundary conditions and solver settings.

f. Solving the System for Method 2

```
u_2 = Function(V, name='u_2')
Ju = (0.5 * inner(grad(u_2), grad(u_2)) - u_2 * f) * dx
F = derivative(Ju, u_2, du=v)
solve(F == 0, u_2, bcs=[bc_x0, bc_x1])
```

This code solves a nonlinear variational problem in Firedrake. Here's the explanation:

- A function u_2 is created in the finite element space V using:

  $$u\_2 = \texttt{Function}(V, \text{name} =' u_2').$$

  This function will store the solution.

- The energy functional $J(u\_2)$ from equation (18) is defined as:

  $$J(u\_2) = \int_\Omega \left(\frac{1}{2} |\nabla u\_2|^2 - u\_2 f \right) dx.$$

- The residual F is computed as the first variation of $J(u_2)$:

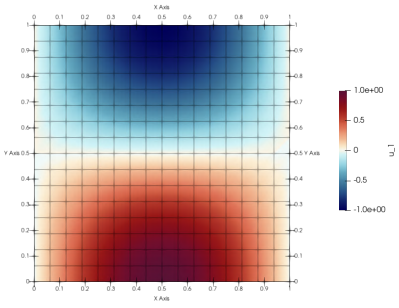
  $$F = \frac{\delta J(u\_2)}{\delta u\_2}(v),$$

  where v is the test function.

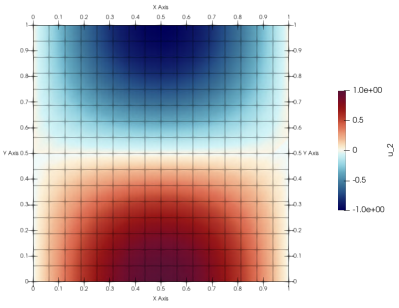
- The problem $F = 0$ is solved for u_2, with Dirichlet boundary conditions $u\_2 = 0$ at $x = 0$ and $x = 1$:

  $$\texttt{solve}(F == 0, u\_2, \text{bcs} = [\text{bc\_x0}, \text{bc\_x1}]).$$

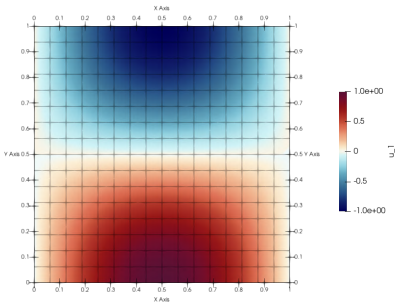
Method 1 Results **Method 2 Results**



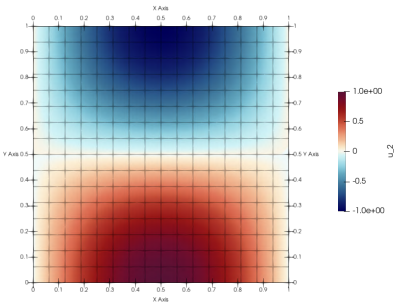
(a) $p = 1, 16 \times 16$



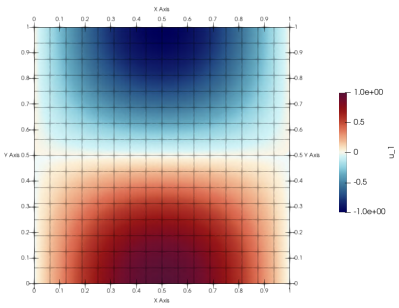
(b) $p = 1, 16 \times 16$



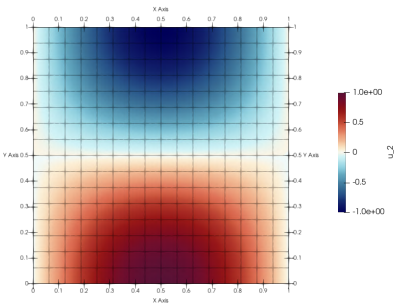
(c) $p = 2, 16 \times 16$



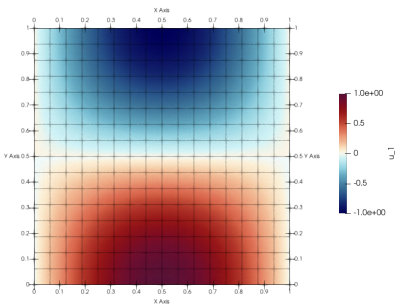
(d) $p = 2, 16 \times 16$



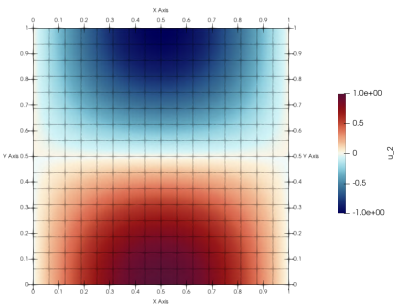
(e) $p = 3, 16 \times 16$



(f) $p = 3, 16 \times 16$



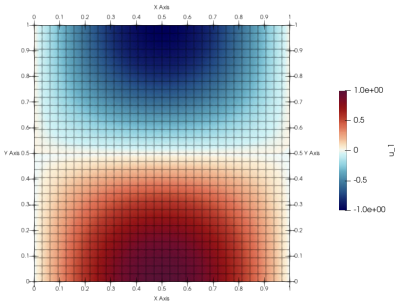
(g) $p = 4, 16 \times 16$



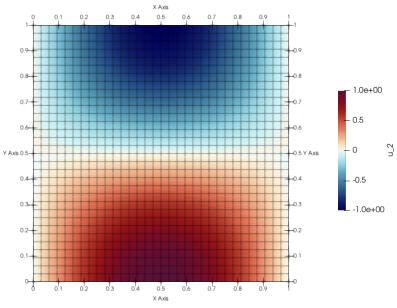
(h) $p = 4, 16 \times 16$

Figure 3: Comparison of numerical result for Method 1 and Method 2 for mesh resolutions of 16×16 and different polynomial orders ($p = 1$ to $p = 4$).

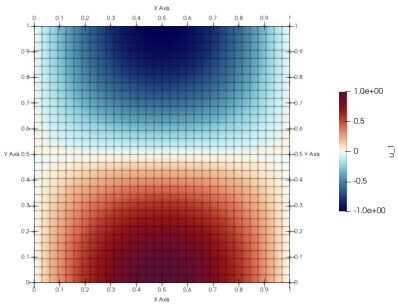
Method 1 Results **Method 2 Results**



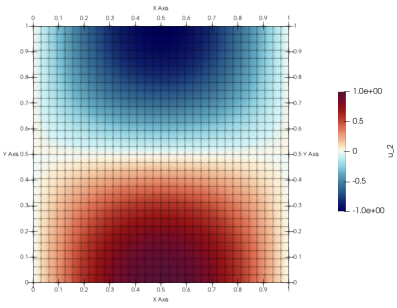
(a) $p = 1, 32 \times 32$



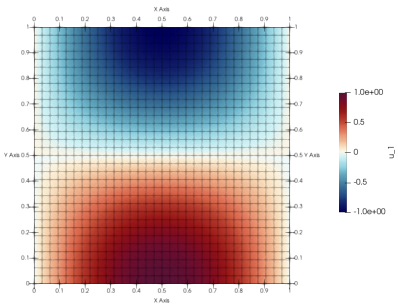
(b) $p = 1, 32 \times 32$



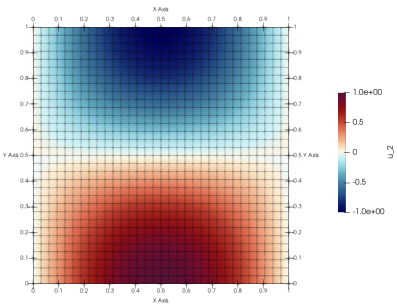
(c) $p = 2, 32 \times 32$



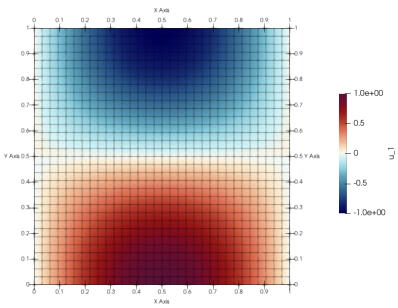
(d) $p = 2, 32 \times 32$



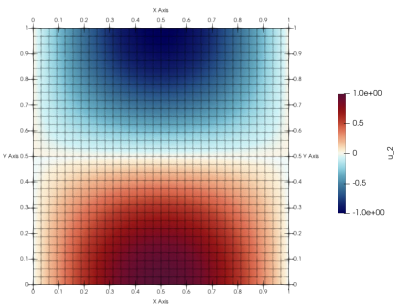
(e) $p = 3, 32 \times 32$



(f) $p = 3, 32 \times 32$



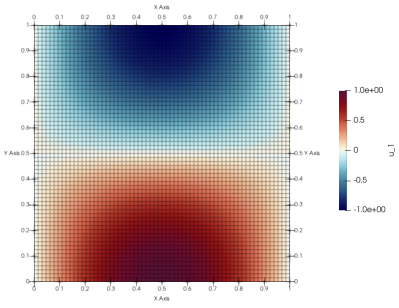
(g) $p = 4, 32 \times 32$



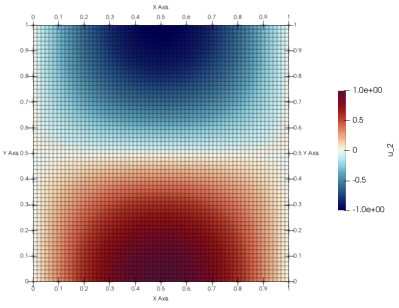
(h) $p = 4, 32 \times 32$

Figure 4: Comparison of numerical result for Method 1 and Method 2 for mesh resolutions of 32×32 and different polynomial orders ($p = 1$ to $p = 4$).

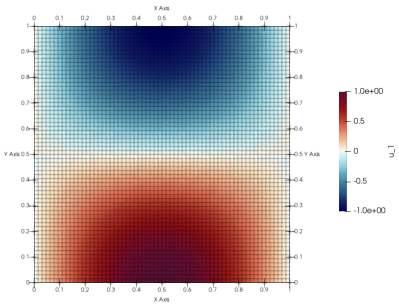
Method 1 Results **Method 2 Results**



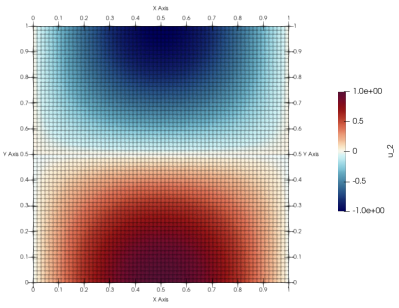
(a) $p = 1, 64 \times 64$



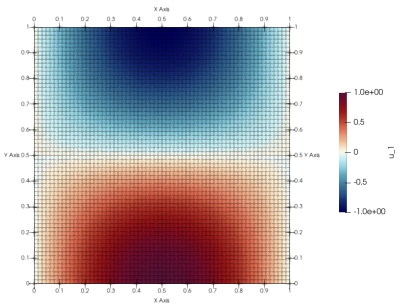
(b) $p = 1, 64 \times 64$



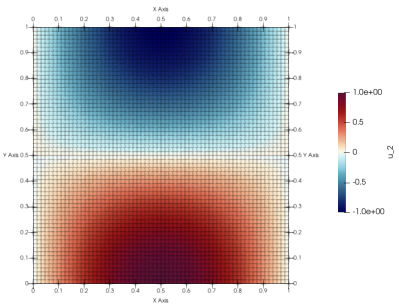
(c) $p = 2, 64 \times 64$



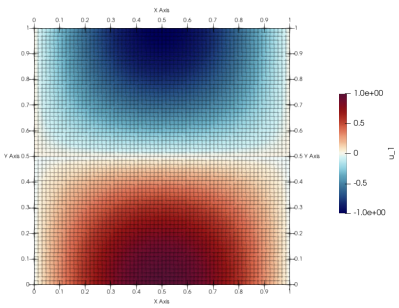
(d) $p = 2, 64 \times 64$



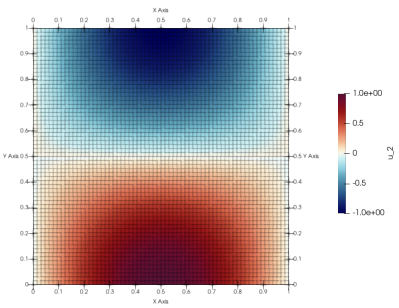
(e) $p = 3, 64 \times 64$



(f) $p = 3, 64 \times 64$



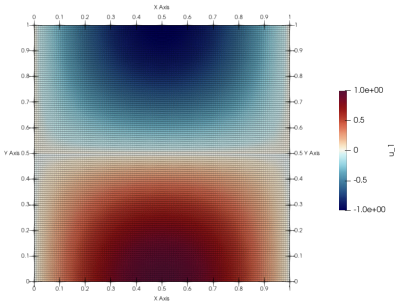
(g) $p = 4, 64 \times 64$



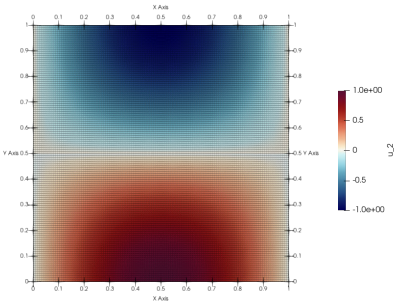
(h) $p = 4, 64 \times 64$

Figure 5: Comparison of numerical result for Method 1 and Method 2 for mesh resolutions of 64×64 and different polynomial orders ($p = 1$ to $p = 4$).

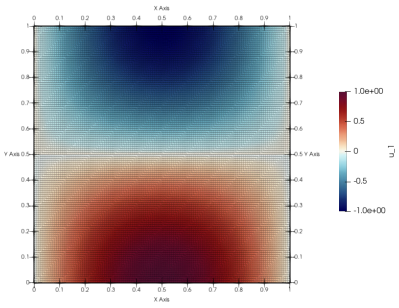
Method 1 Results **Method 2 Results**



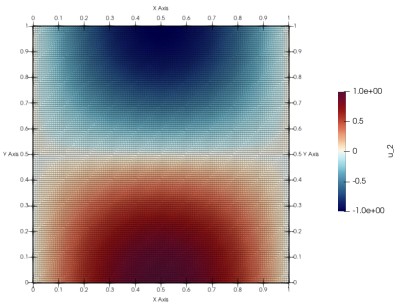
(a) $p = 1, 128 \times 128$



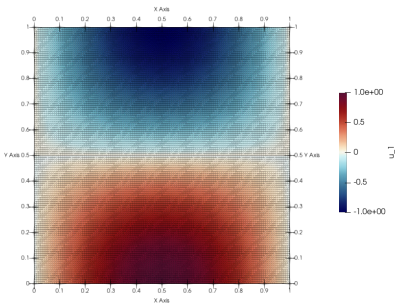
(b) $p = 1, 128 \times 128$



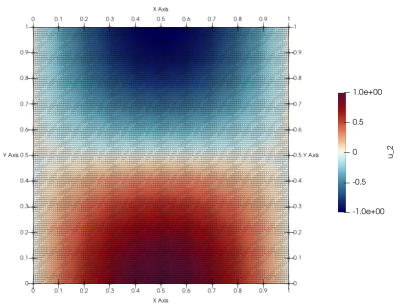
(c) $p = 2, 128 \times 128$



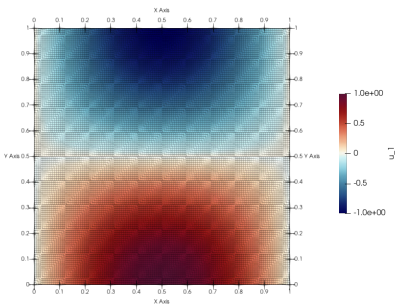
(d) $p = 2, 128 \times 128$



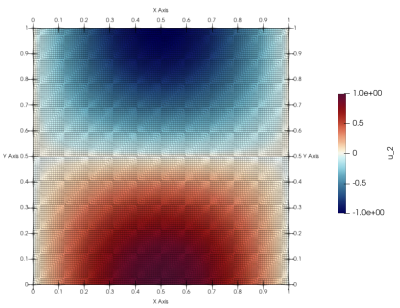
(e) $p = 3, 128 \times 128$



(f) $p = 3, 128 \times 128$



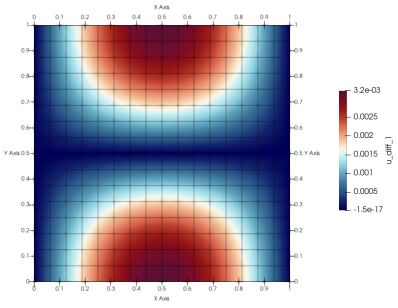
(g) $p = 4, 128 \times 128$



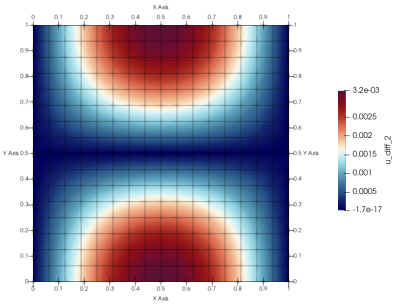
(h) $p = 4, 128 \times 128$

Figure 6: Comparison of numerical result for Method 1 and Method 2 for mesh resolutions of 128×128 and different polynomial orders ($p = 1$ to $p = 4$).

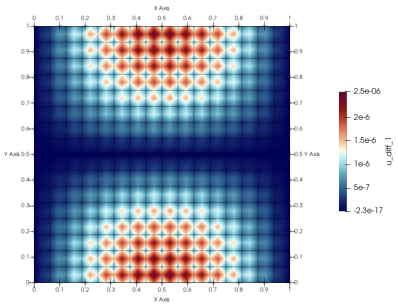
Method 1 Errors **Method 2 Errors**



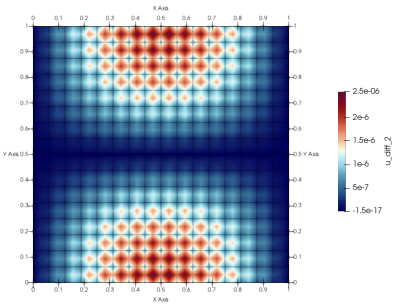
(a) $p = 1, 16 \times 16$



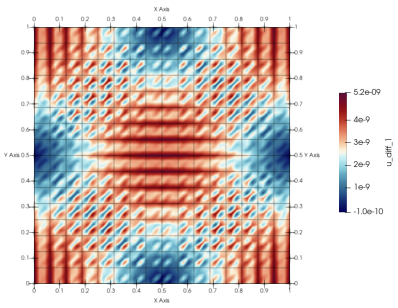
(b) $p = 1, 16 \times 16$



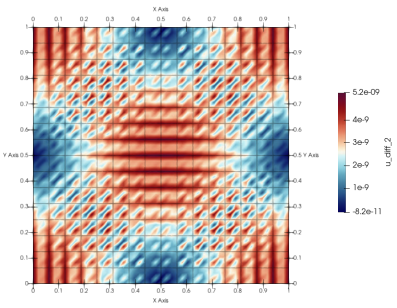
(c) $p = 2, 16 \times 16$



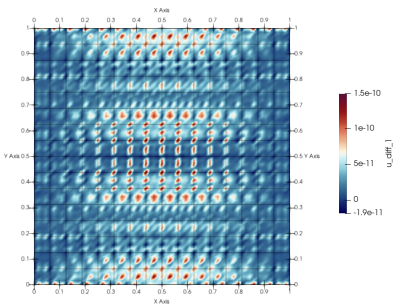
(d) $p = 2, 16 \times 16$



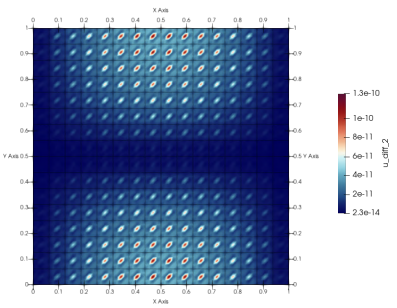
(e) $p = 3, 16 \times 16$



(f) $p = 3, 16 \times 16$



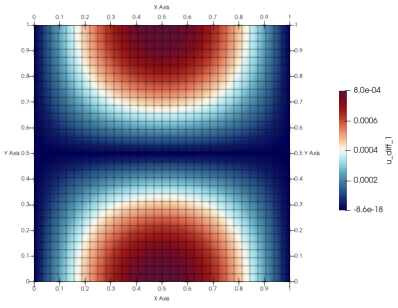
(g) $p = 4, 16 \times 16$



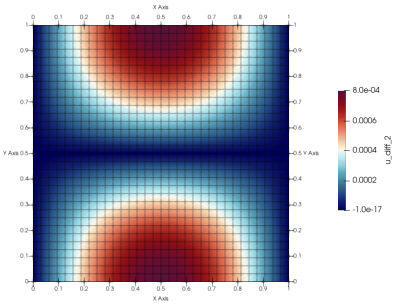
(h) $p = 4, 16 \times 16$

Figure 7: Comparison of numerical error for Method 1 and Method 2 for mesh resolutions of 16×16 and different polynomial orders ($p = 1$ to $p = 4$).

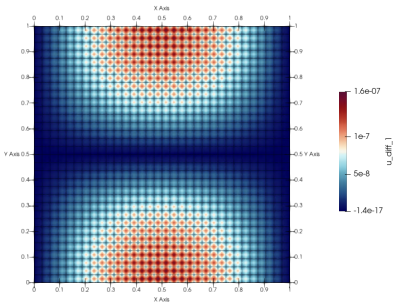
Method 1 Errors **Method 2 Errors**



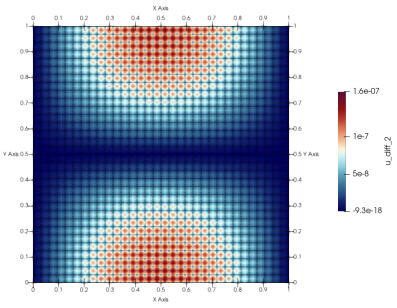
(a) $p = 1, 32 \times 32$



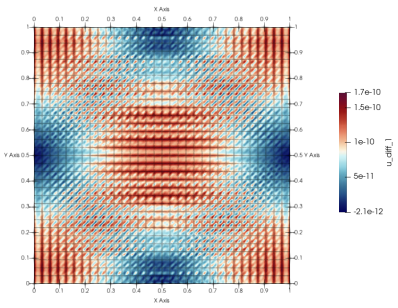
(b) $p = 1, 32 \times 32$



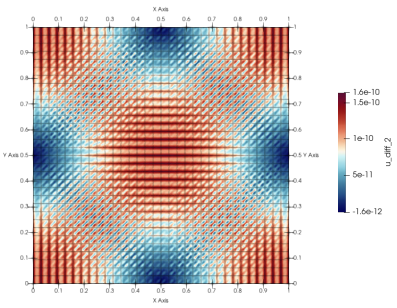
(c) $p = 2, 32 \times 32$



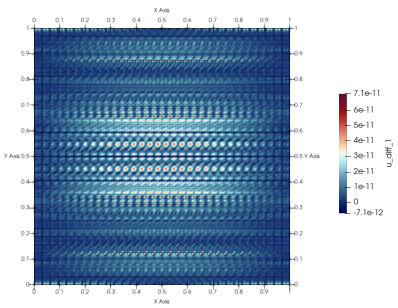
(d) $p = 2, 32 \times 32$



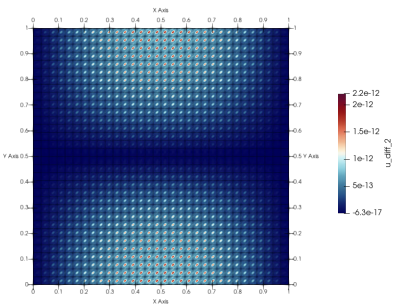
(e) $p = 3, 32 \times 32$



(f) $p = 3, 32 \times 32$



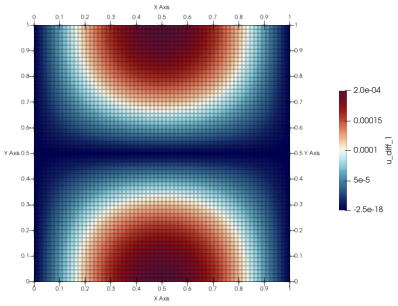
(g) $p = 4, 32 \times 32$



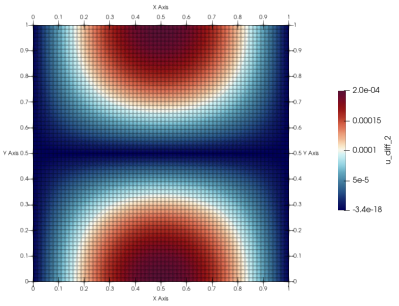
(h) $p = 4, 32 \times 32$

Figure 8: Comparison of numerical error for Method 1 and Method 2 for mesh resolutions of 32×32 and different polynomial orders ($p = 1$ to $p = 4$).

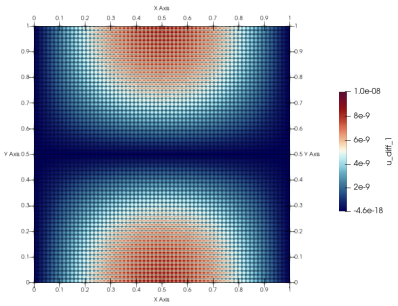
Method 1 Errors **Method 2 Errors**



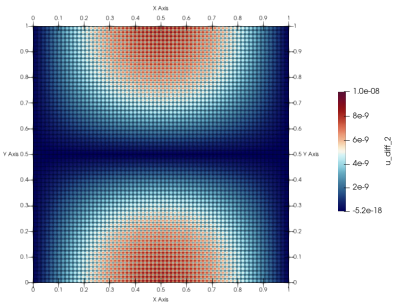
(a) $p = 1, 64 \times 64$



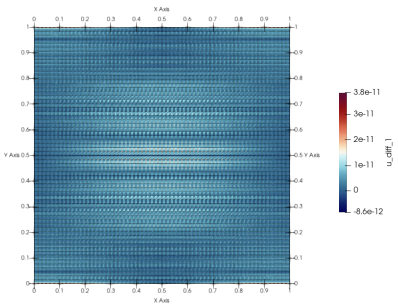
(b) $p = 1, 64 \times 64$



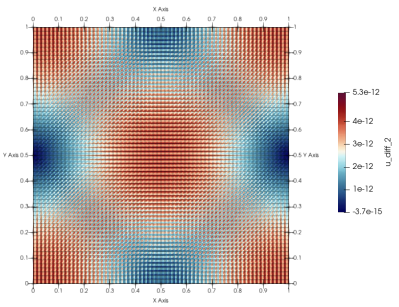
(c) $p = 2, 64 \times 64$



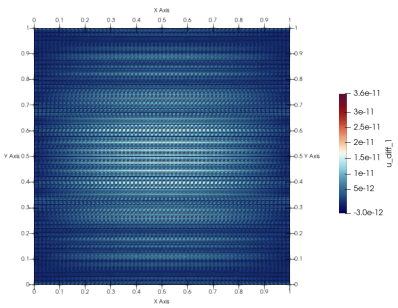
(d) $p = 2, 64 \times 64$



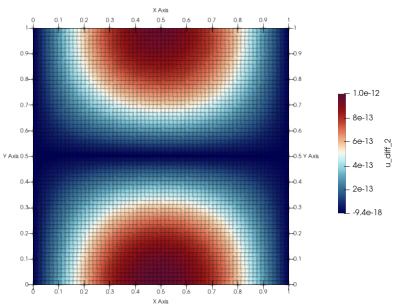
(e) $p = 3, 64 \times 64$



(f) $p = 3, 64 \times 64$



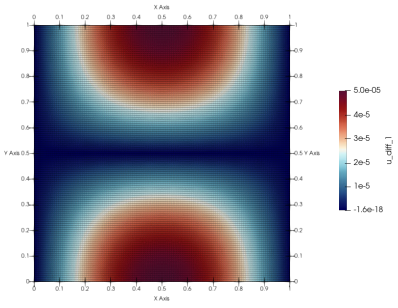
(g) $p = 4, 64 \times 64$



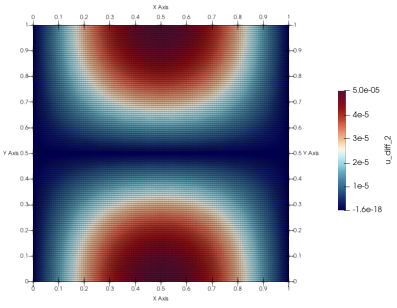
(h) $p = 4, 64 \times 64$

Figure 9: Comparison of numerical error for Method 1 and Method 2 for mesh resolutions of 64×64 and different polynomial orders ($p = 1$ to $p = 4$).

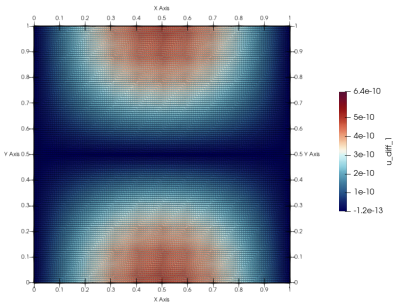
Method 1 Errors **Method 2 Errors**



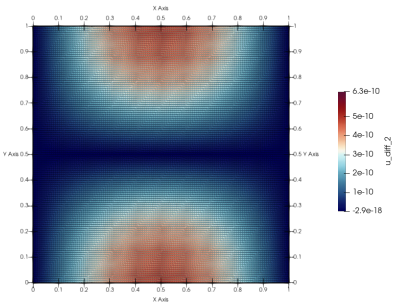
(a) $p = 1, 128 \times 128$



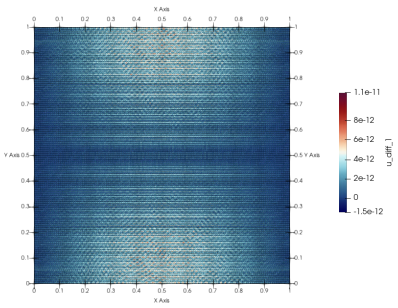
(b) $p = 1, 128 \times 128$



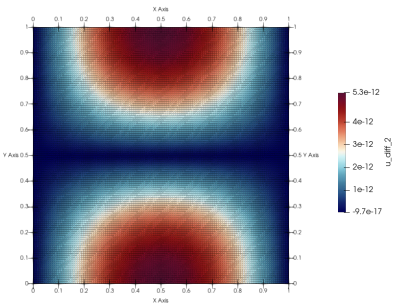
(c) $p = 2, 128 \times 128$



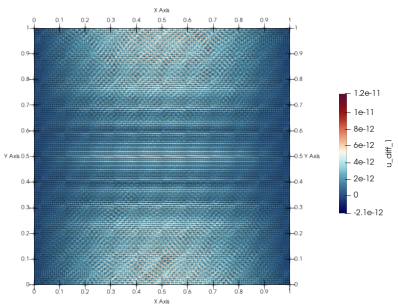
(d) $p = 2, 128 \times 128$



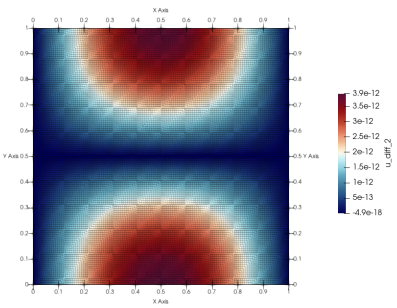
(e) $p = 3, 128 \times 128$



(f) $p = 3, 128 \times 128$



(g) $p = 4, 128 \times 128$



(h) $p = 4, 128 \times 128$

Figure 10: Comparison of numerical errror for Method 1 and Method 2 for mesh resolutions of 128×128 and different polynomial orders ($p = 1$ to $p = 4$).