Math 597

Project #3

Damyn Chipman

Introduction

The final set of equations we will look at in 1D are elliptic type equations. Elliptic equations arise in a variety of physics applications and require a slightly different approach when solving them than hyperbolic equations.

Primarily, we will need to take a good look at the boundary conditions imposed on elliptic problems as they dicate how the solution behaves. We'll look at how to apply both a Dirichlet and Neumann boundary condition to our elliptic solver.

We'll go through the derivation of the local and global matrices we will construct, as well as the code and algorithms required to generate them. We'll also do our standard convergence analysis.

As always, the code for this project is housed on GitHub on the HydroForest repository.

Problem Statement

We will consider the following boundary value problem (BVP):

$$PDE: \ \frac{d^2q(x)}{dx^2} + q(x) = f(x), \qquad \forall x \in \Omega = [-1, 1] \tag{1}$$

$$BC: \ q(x)|_{x=-1} = g = 0 \tag{2}$$

$$BC: \ \frac{dq(x)}{dx}|_{x=1} = h = -\pi \tag{3}$$

where $f(x) = (1 - \pi^2)\sin(\pi x)$ and the exact solution is known to be $q(x) = \sin(\pi x)$.

This equation is known as Helmholtz equation.

From PDE to Element Linear Systems

Let's use our standard Galerkin approach to form a system of linear equations on the reference element.

We start by expressing q and f in terms of an expansion of basis functions:

$$q(x) \approx q_N^{(e)} = \sum_{j=0}^{N} \psi_j(x) q_j^{(e)} \tag{4}$$

$$f(x) \approx f_N^{(e)} = f(q_N^{(e)}) \tag{5}$$

The CG and DG approaches diverge from the start for this case, so let's start with the CG approach.

Continuous Galerkin (CG) Method

We multiply our PDE by a test function that is the same as our basis function (the Galerkin principle) and integrate over an individual element:

$$\int_{\Omega_e} \psi_i(x) \frac{d^2 q_N^{(e)}(x)}{dx^2} d\Omega_e + \int_{\Omega_e} \psi_i(x) q_N^{(e)}(x) d\Omega_e = \int_{\Omega_e} \psi_i(x) f_N^{(e)} d\Omega_e \tag{6}$$

Using the product rule, we expand the first term to give us:

$$\int_{\Omega_e} \Big(\frac{d}{dx} \big[\psi_i(x) \frac{dq_N^{(e)}(x)}{dx} \big] - \frac{d\psi_i(x)}{dx} \frac{dq_N^{(e)}(x)}{dx} \Big) d\Omega_e + \int_{\Omega_e} \psi_i(x) q_N^{(e)}(x) d\Omega_e = \int_{\Omega_e}$$

Now using the Fundamental Theorem of Calculus, we get:

$$\Big[\psi_i(x) \frac{dq_N^{(e)}(x)}{dx} \Big]_{\Gamma_e} - \int_{\Omega_e} \frac{d\psi_i(x)}{dx} \frac{dq_N^{(e)}(x)}{dx} d\Omega_e + \int_{\Omega_e} \psi_i(x) q_N^{(e)}(x) d\Omega_e = \int_{\Omega_e} \psi_i(x$$

Now plug in our approximation and expansion:

$$\left[\psi_i(x)\frac{dq_N^{(e)}(x)}{dx}\right]_{\Gamma_e} - \sum_{j=0}^{N}\int_{\Omega_e}\frac{d\psi_i(x)}{dx}\frac{d\psi_j(x)}{dx}d\Omega_e q_j^{(e)} + \sum_{j=0}^{N}\int_{\Omega_e}\psi_i(x)\psi_j(x)d\Omega_e q_j^{(e)}$$

Next we transform to the reference element:

$$\left[\psi_i(x)\frac{dq_N^{(e)}(x)}{dx}\right]_{\Gamma_e} - \sum_{j=0}^{N}\int_{\hat{\Omega}}\frac{d\psi_i(\xi)}{d\xi}\frac{d\psi_j(\xi)}{d\xi}\frac{d\xi}{dx}d\hat{\Omega}q_j^{(e)} + \sum_{j=0}^{N}\int_{\hat{\Omega}}\psi_i(\xi)\psi_j(\xi)\frac{dx}{d\xi}d\hat{\Omega}$$

Which will result in the following local element system:

$$B_i^{(e)} - L_{ij}^{(e)}q_j^{(e)} + M_{ij}^{(e)}q_j^{(e)} = M_{ij}^{(e)}f_j^{(e)} \tag{11}$$

$$\Rightarrow (-L_{ij}^{(e)} + M_{ij}^{(e)})q_j^{(e)} = M_{ij}^{(e)}f_j^{(e)} - B_i^{(e)} \tag{12}$$

$$\Rightarrow H_{ij}^{(e)}q_j^{(e)} = M_{ij}^{(e)}f_j^{(e)} - B_i^{(e)} \tag{13}$$

Discontinuous Galerkin (DG) Method

For the DG approach, we will be using an auxilary variable to convert the second order ODE into a system of first order ODEs. We does this as follows:

Let

$$Q(x) = \frac{dq(x)}{dx} \tag{14}$$

which gives us the following system to solve:

$$\begin{cases} \frac{dq(x)}{dx} = Q(x) \\ \frac{dQ(x)}{dx} + q(x) = f(x) \end{cases} \tag{15}$$

Now use our Galerkin approach by multiply both equations by a test function and integrating over an element:

$$\begin{cases} \int_{\Omega_e} \psi_i(x) \frac{dq_N^{(e)}(x)}{dx} d\Omega_e = \int_{\Omega_e} \psi_i(x) Q_N^{(e)}(x) d\Omega_e \\ \int_{\Omega_e} \psi_i(x) \frac{dQ_N^{(e)}(x)}{dx} d\Omega_e + \int_{\Omega_e} \psi_i(x) q_N^{(e)}(x) d\Omega_e = \int_{\Omega} \psi_i(x) f_N^{(e)}(x) d\Omega_e \end{cases} \tag{16}$$

Next, we use the product rule and the FTC to transfer a derivative off of q and Q onto the test functions:

$$\begin{cases} \left[\psi_i(x) q_N^{(*,e)}(x)\right]\Big|_{\Gamma_e} - \int_{\Omega_e} \frac{d\psi_i(x)}{dx} q_N^{(e)}(x) d\Omega_e = \int_{\Omega_e} \psi_i(x) Q_N^{(e)}(x) d\Omega_e \\ \left[\psi_i(x) Q_N^{(*,e)}(x)\right]\Big|_{\Gamma_e} - \int_{\Omega_e} \frac{d\psi_i(x)}{dx} Q_N^{(e)}(x) d\Omega_e + \int_{\Omega_e} \psi_i(x) q_N^{(e)}(x) d\Omega_e = \int_{\Omega_e} \psi_i(x)
\end{cases}$$

Now plug in our expansion...

$$\begin{cases} \left[\psi_i(x) q_N^{(*,e)}(x)\right]\Big|_{\Gamma_e} - \sum_{j=0}^{N} \int_{\Omega_e} \frac{d\psi_i(x)}{dx} \psi_j(x) d\Omega_e q_j^{(e)} = \sum_{j=0}^{N} \int_{\Omega_e} \psi_i(x) \psi_j(x) d\Omega_e \\ \left[\psi_i(x) Q_N^{(*,e)}(x)\right]\Big|_{\Gamma_e} - \sum_{j=0}^{N} \int_{\Omega_e} \frac{d\psi_i(x)}{dx} \psi_j(x) d\Omega_e Q_j^{(e)} + \sum_{j=0}^{N} \int_{\Omega_e} \psi_i(x) \psi_j(x) d\Omega
\end{cases}$$

And transform to the reference element:

$$\begin{cases} \left[\psi_i(x) q_N^{(*,e)}(x)\right]\Big|_{\Gamma_e} - \sum_{j=0}^{N} \int_{\hat{\Omega}} \frac{d\psi_i(\xi)}{d\xi} \psi_j(\xi) d\hat{\Omega} q_j^{(e)} = \sum_{j=0}^{N} \int_{\hat{\Omega}} \psi_i(\xi) \psi_j(\xi) \frac{dx}{d\xi} d\hat{\Omega} Q \\ \left[\psi_i(x) Q_N^{(*,e)}(x)\right]\Big|_{\Gamma_e} - \sum_{j=0}^{N} \int_{\hat{\Omega}} \frac{d\psi_i(\xi)}{d\xi} \psi_j(\xi) d\hat{\Omega} Q_j^{(e)} + \sum_{j=0}^{N} \int_{\hat{\Omega}} \psi_i(\xi) \psi_j(\xi) \frac{dx}{d\xi} d\hat{\Omega}
\end{cases}$$

Leading us to the following system:

$$\begin{cases} F_{ij}^{(e)} q_j^{(*,e)} - \tilde{D}_{ij}^{(e)} q_j^{(e)} = M_{ij}^{(e)} Q_j^{(e)} \\ F_{ij}^{(e)} Q_j^{(*,e)} - \tilde{D}_{ij}^{(e)} Q_j^{(e)} + M_{ij}^{(e)} q_j^{(e)} = M_{ij}^{(e)} f_j^{(e)} \end{cases} \tag{20}$$

We'll look at how to solve this system as well as how to handle F_{ij} and $q_j^{(*,e)}$ and $Q_j^{(*,e)}$ after we form a global system.

From Element Linear System to Global Linear System

Just like how we used a local to global mapping matrix ID in project 2, we will use the same mapping here. This also means that the direct stiffness summation (DSS) operation is fairly similar. The issue will be in how we apply the boundary conditions, but we will look at that as part of the global system.

CG Global System

Using the same DSS operations as before (except for the Laplacian matrix, where we use an inverse metric term instead), we get the following global system:

$$(-L_{IJ} + M_{IJ})q_J = M_{IJ}f_J - B_I \tag{21}$$
$$\Rightarrow q_J = (M_{IJ} - L_{IJ})^{-1}M_{IJ}f_J - (M_{IJ} - L_{IJ})^{-1}B_I \tag{22}$$

DG Global System

We use the same DSS operations for the DG approach as well. The flux matrix, however, is slightly different. In the hyperbolic problem in project 2, we used a global flux vector computed using the Rusanov numerical flux. Becuase our elliptic problem is steady, we do not need an upwinding scheme for the numerical flux, so we just use a centered flux:

$$f^{(*,e,k)} = \frac{1}{2}(q^{(e)} + q^{(k)}) \tag{23}$$

where e is the element and k is the element face. So when we DSS the DG element system, we will use an approach that seperates the flux matrix into a boundary vector and a flux matrix times just the state variables q and Q. We'll provide details in the section below on boundary conditions.

Using the DSS operation, we form the following global system and solve for Q_J:

$$\begin{cases} B_I^{(q)} + F_{IJ}^* q_J - \tilde{D}_{IJ} q_J = M_{IJ} Q_J \\ B_I^{(Q)} + F_{IJ}^* Q_J - \tilde{D}_{IJ} Q_J + M_{IJ} = M_{IJ} f_J \end{cases} \tag{24}$$

$$\Rightarrow Q_J = (M_{IJ})^{-1}(B_I^{(q)} + \hat{D}_{IJ}^{(q)} q_J) \tag{25}$$

where $\hat{D}_{IJ} = F_{IJ}^* - \tilde{D}_{IJ}$.

Now plug this into the other equation and solve for q_J:

$$B_I^{(Q)} + \hat{D}_{IJ}^{(Q)}\left((M_{IJ})^{-1}(B_I^{(q)} + \hat{D}_{IJ}^{(q)} q_J)\right) + M_{IJ} q_J = M_{IJ} f_J \tag{26}$$

$$\Rightarrow H_{IJ} q_J = M_{IJ} f_J - B'_I \tag{27}$$

where

$$H_{IJ} = \hat{D}_{IK}^{(Q)} M_{KL}^{-1} \hat{D}_{LJ}^{(q)} + M_{IJ} \tag{28}$$

$$B'_I = B_I^{(Q)} - \hat{D}_{IK}^{(Q)} M_{KJ}^{-1} B_J^{(q)} \tag{29}$$

which does look quite eloquent compared to the actual ODE doesn't it?

Boundary Conditions

We need to look at how to apply the Dirichlet boundary condition at the left edge and the Neumann boundary condition at the right edge. These will be different for both CG and DG.

CG Boundary Conditions

Let's consider the Neumann BC first, as those are simplier in FE methods. Consider the form of the boundary integral:

$$B_i^{(e)} = \left[\psi_i(x)\frac{dq_N^{(e)}(x)}{dx}\right]_{\Gamma_e} \tag{30}$$

This only lives on the boundary of the element, and disappears between elements. Also, at element boundary, all basis functions are either zero or unity, so we only need to worry about the $\frac{dq}{dx}$ piece. Fortunately, we know $\frac{dq}{dx}$ at the Neumann boundary, we simply need to create a vector with zeros everywhere except the global index point corresponding to the Neumann boundary (in this case, it's the last point). So this looks like:

$$B_I = \begin{bmatrix}\dots & 0 & h\end{bmatrix}^T = \begin{bmatrix}\dots & 0 & -\pi\end{bmatrix}^T \tag{31}$$

The Dirichlet BC cannot be inforced through the boundary vector, so we will enforce it in the strong sense. This means we will manipulate the global system to force the value of q at the Dirichelt boundary to be the value we need.

The global system looks like:

$$(-L_{IJ} + M_{IJ})q_J = M_{IJ}f_J - B_I = R_I \tag{32}$$

So we change the first row of $-L_{IJ} + M_{IJ}$ to be zeros except for a one in the first row, first column. And we force the first entry in the RHS vector R_I to be the value of the function at the edge. This looks like for the first row/equation of the global system:

$$\begin{bmatrix} 1 & 0 & 0 & \dots \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} g \\ R_1 \\ R_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} 0 \\ R_1 \\ R_2 \\ \vdots \end{bmatrix} \tag{33}$$

This forces the first equation in the global system to read $q_0 = g = 0$.

DG Boundary Conditions

Apply the DG boundary conditions are a little more tricky. We do so through the flux matrix F_{IJ} and the boundary vector B_I.

When we go from the element linear system

$$\begin{cases} F_{ij}^{(e)} q_j^{(*,e)} - \tilde{D}_{ij}^{(e)} q_j^{(e)} = M_{ij}^{(e)} Q_j^{(e)} \\ F_{ij}^{(e)} Q_j^{(*,e)} - \tilde{D}_{ij}^{(e)} Q_j^{(e)} + M_{ij}^{(e)} q_j^{(e)} = M_{ij}^{(e)} f_j^{(e)} \end{cases} \tag{34}$$

to the global linear system

$$\begin{cases} B_I^{(q)} + F_{IJ}^* q_J - \tilde{D}_{IJ} q_J = M_{IJ} Q_J \\ B_I^{(Q)} + F_{IJ}^* Q_J - \tilde{D}_{IJ} Q_J + M_{IJ} = M_{IJ} f_J \end{cases} \tag{35}$$

we transfer the action of the numerical flux onto the flux vector itself. This allows us to operate with the state vectors q_J and Q_J freely. Forming the flux matrix then requires a new approach. We motivate it by expressing the global system first as the flux matrix from previous projects (i.e., just [-1, 0, ..., 0, 1] on the diagonal per element) times the numerical flux (centered flux in this case):

$$F_{IJ}q_J^{(*,e,k)} = \begin{bmatrix} -1 & & & & & & & & & & & & & & & \\ & \ddots & & & & & & & & & & & & & & \\ & & 1 & & & & & & & & & & & & & \\ & & & -1 & & & & & & & & & & & & \\ & & & & \ddots & & & & & & & & & & & \\ & & & & & 1 & & & & & & & & & & \\ & & & & & & \ddots & & & & & & & & & \\ & & & & & & & \ddots & & & & & & & & \\ & & & & & & & & \ddots & & & & & & & \\ & & & & & & & & & -1 & & & & & & \\ & & & & & & & & & & \ddots & & & & & \\ & & & & & & & & & & & 1 & & & & \\ & & & & & & & & & & & & -1 & & & \\ & & & & & & & & & & & & & & & \\ & & & & & & & & & & & & & & & \\ & & & & & & & & & & & & & & & \end{bmatrix}$$

Now for each element interface, we consider the numerical flux scheme we are using and write the system in terms of the state variables instead of numerical flux variables. For example, at the right edge of the first element, we would get:

$$q^{(*,0,1)} = \frac{1}{2}(q_N + q_{N+1}) \tag{37}$$

Rewriting the system as such yields:

$$F^*_{IJ}q_J = \begin{bmatrix} ? & & & & & & & & & & & & \\ & \ddots & & & & & & & & & & & \\ & & \frac{1}{2} & \frac{1}{2} & & & & & & & & & \\ & & -\frac{1}{2} & -\frac{1}{2} & & & & & & & & & \\ & & & & \ddots & & & & & & & & \\ & & & & & \frac{1}{2} & \frac{1}{2} & & & & & & \\ & & & & & & \ddots & & & & & & \\ & & & & & & & \ddots & & & & & \\ & & & & & & & & \ddots & & & & \\ & & & & & & & & -\frac{1}{2} & -\frac{1}{2} & & & \\ & & & & & & & & & & \ddots & & \\ & & & & & & & & & & & \frac{1}{2} & \frac{1}{2} \\ & & & & & & & & & & & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix}$$

Now for the boundaries. We can form this system for both q and Q. At the left edge, we need to apply Dirichlet BC, so the numerical flux looks like:

$$q^{(*,0,-1)} = \frac{1}{2}(q_{-1} + q_0) = \frac{1}{2}(g + q_0) \tag{39}$$

And for the right edge, we apply Neumann BC, which look like Dirichlet BC for Q:

$$Q^{(*,N_e-1,N_e)} = \frac{1}{2}(Q_{N_e(N+1)-1} + Q_{N_e(N+1)}) = \frac{1}{2}(Q_{N_e(N+1)-1} + h) \tag{40}$$

Now, the last piece is to figure out what to do for q on the right edge (where there is no BC enforced) and Q on the left edge (again, where no BC is enforced). We do this by enforcing the natural BC for q or Q at their respective edges:

(Left edge for Q):

$$Q^{(*,0,-1)} = \frac{1}{2}(Q_{-1} + Q_0) = \frac{1}{2}(Q_0 + Q_0) = Q_0 \tag{41}$$

(Right edge for q):

$$q^{(*,N_e-1,N_e)} = \frac{1}{2}(q_{N_e(N+1)-1} + q_{N_e(N+1)}) = \frac{1}{2}(q_{N_e(N+1)-1} + q_{N_e(N+1)-1}) = q_{N_e(N}$$

This results in the follwing two terms: 1) a numerical flux matrix times the state vector and 2) the boundary vector.

$$F^{*}_{IJ}q_J + B^{(q)}_I = \left[\begin{matrix}
-\frac{1}{2} & & & & & & & & & & \\
 & \ddots & & & & & & & & & \\
 & & \frac{1}{2} & \frac{1}{2} & & & & & & & \\
 & & -\frac{1}{2} & -\frac{1}{2} & & & & & & & \\
 & & & & \ddots & & & & & & \\
 & & & & & \frac{1}{2} & \frac{1}{2} & & & & \\
 & & & & & & \ddots & & & & \\
 & & & & & & & \ddots & & & \\
 & & & & & & & & \ddots & & \\
 & & & & & & & & -\frac{1}{2} & -\frac{1}{2} & \\
 & & & & & & & & & & \ddots
\end{matrix}\right.$$

$$F^*_{IJ}Q_J + B^{(Q)}_I = \begin{bmatrix} -1 & & & & & & & & & & & & & & \\ & \ddots & & & & & & & & & & & & & \\ & & \frac{1}{2} & \frac{1}{2} & & & & & & & & & & & \\ & & -\frac{1}{2} & -\frac{1}{2} & & & & & & & & & & & \\ & & & & \ddots & & & & & & & & & & \\ & & & & & \frac{1}{2} & \frac{1}{2} & & & & & & & & \\ & & & & & & \ddots & & & & & & & & \\ & & & & & & & \ddots & & & & & & & \\ & & & & & & & & \ddots & & & & & & \\ & & & & & & & & & -\frac{1}{2} & -\frac{1}{2} & & & & \\ & & & & & & & & & & & \ddots & & & \\ & & & & & & & & & & & & & & \\ & & & & & & & & & & & & & & \\ & & & & & & & & & & & & & & \\ & & & & & & & & & & & & & & \\ & & & & & & & & & & & & & & \\ & & & & & & & & & & & & & & \\ & & & & & & & & & & & & & & \\ & & & & & & & & & & & & & & \end{bmatrix}$$

The algorithm for forming F^*_{IJ} follows Algorithm 6.4 from the book and the code is provided below:

```
template<typename NumericalType>
class DGGlobalCenteredFluxMatrix : public Matrix<NumericalType> {

public:


DGGlobalCenteredFluxMatrix(std::vector<Element1D<NumericalType>>&
elements, Matrix<int>& IDMatrix, BoundaryConditionType
leftBoundaryType, BoundaryConditionType rightBoundaryType) :
        Matrix<NumericalType>(elements.size()*(IDMatrix.nRows()),
elements.size()*(IDMatrix.nRows()), 0) {

        //
        int N = IDMatrix.nRows()-1;
        for (auto e = 0; e < elements.size(); e++) {
```

```
            int L, R, I, J;

            // Identify left and right elements
            L = e-1;
            R = e+1;
            if (e == 0) {
                if (leftBoundaryType ==
BoundaryConditionType::Periodic) L = elements.size()-1;
                else L = -1;
            }
            else if (e == elements.size()-1) {
                if (rightBoundaryType ==
BoundaryConditionType::Periodic) R = 0;
                else R = -1;
            }

            if (L != -1) {
                I = IDMatrix(0, e);
                J = IDMatrix(N, L);
                this->operator()(I,I) = -0.5;
                this->operator()(I,J) = -0.5;
            }
            else {
                I = 0;
                this->operator()(I,I) = 1.0;
            }

            if (R != -1) {
                I = IDMatrix(N, e);
                J = IDMatrix(0, R);
                this->operator()(I,I) = 0.5;
                this->operator()(I,J) = 0.5;
            }
            else {
                I = this->nRows()-1;
                this->operator()(I,I) = 1.0;
            }
        }

    }

};
```

With the matrices formed above, we can now solve for q_J using the equation we derived as part of the global linear system.

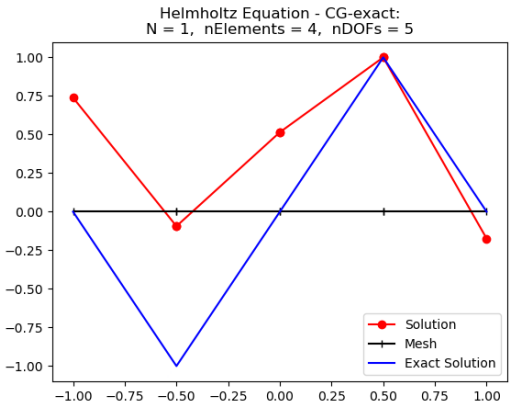
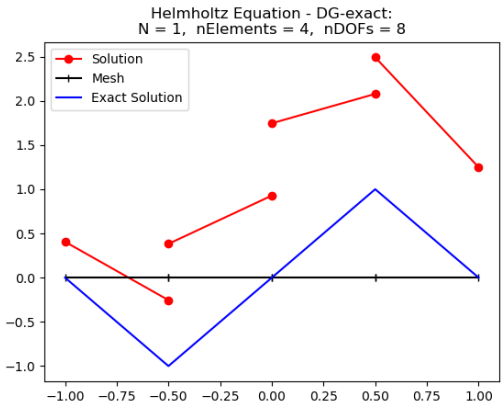
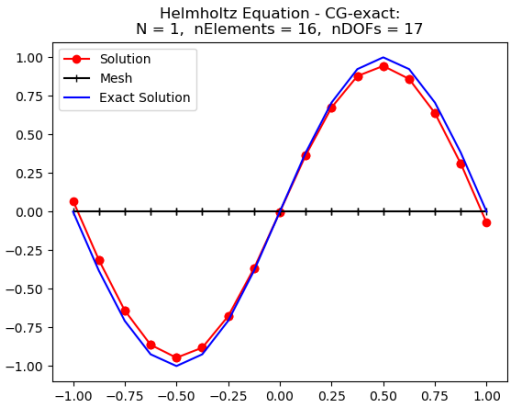
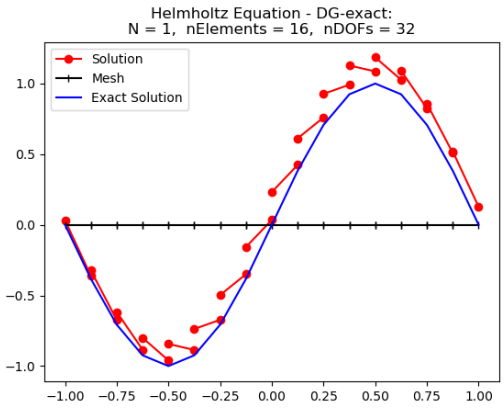
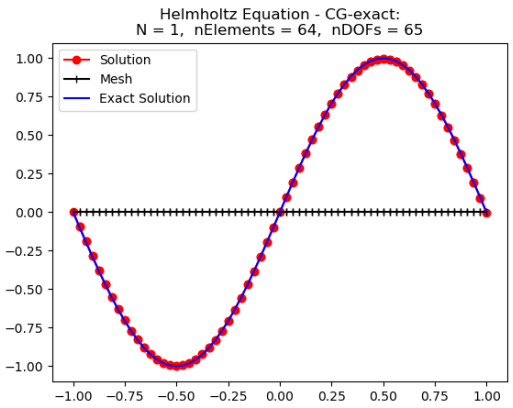
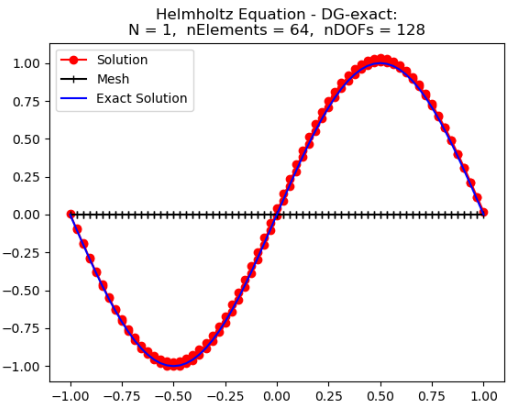
Results

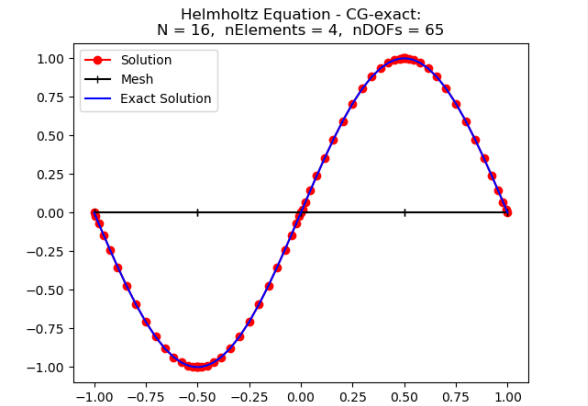
Results

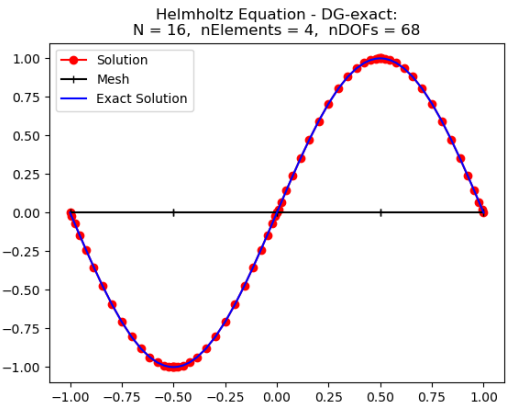
Plots

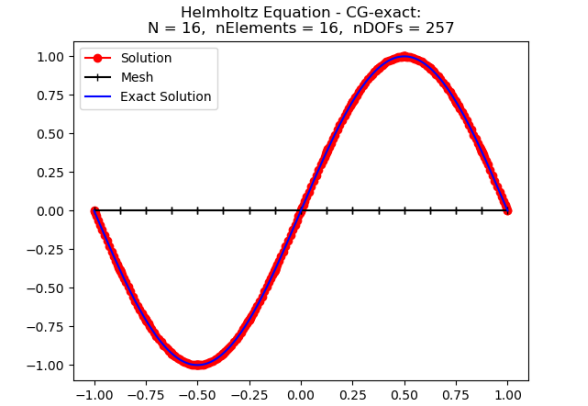
Below are plots of the numerical and exact solution for various orders and number of elements:

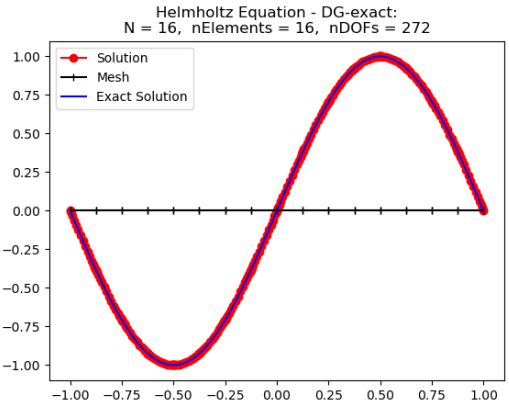
Exact Integration

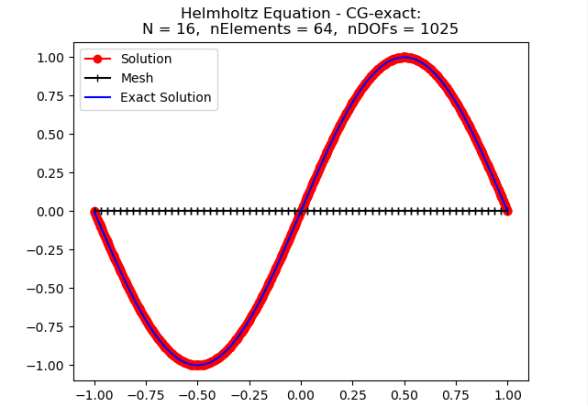
CG	DG
	
	
	

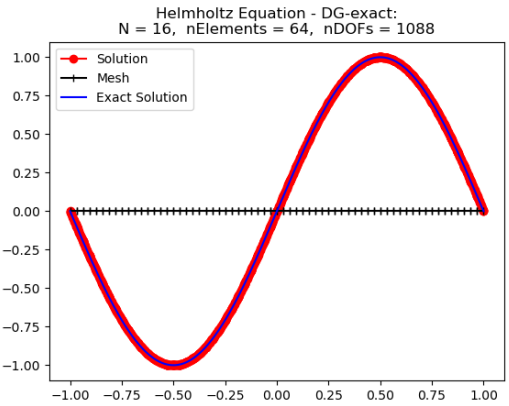






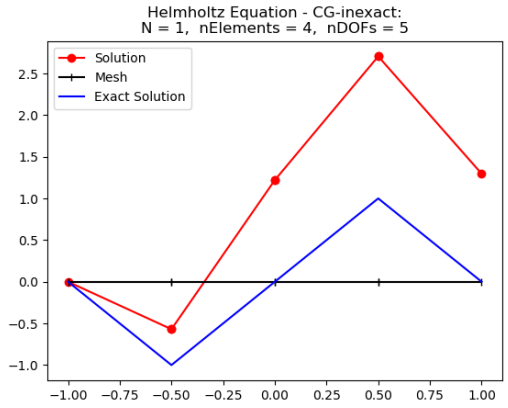


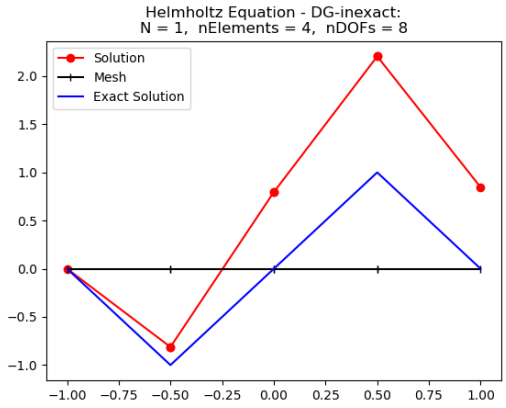


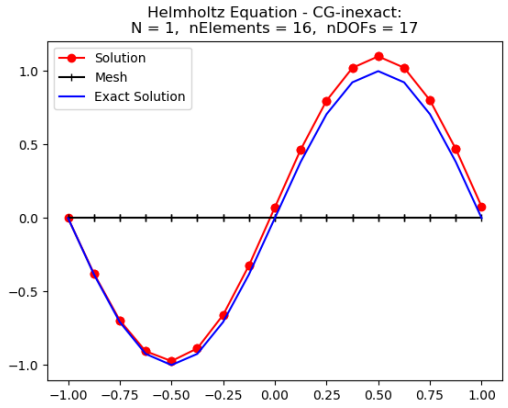


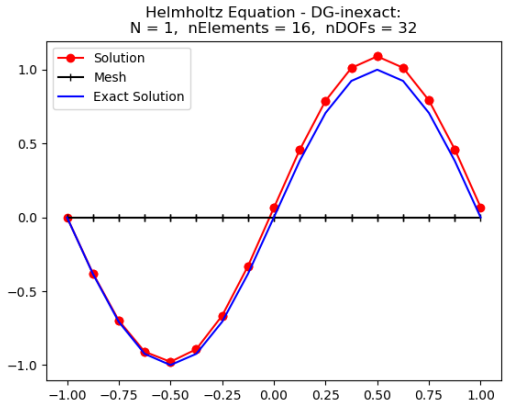
Inexact Integration

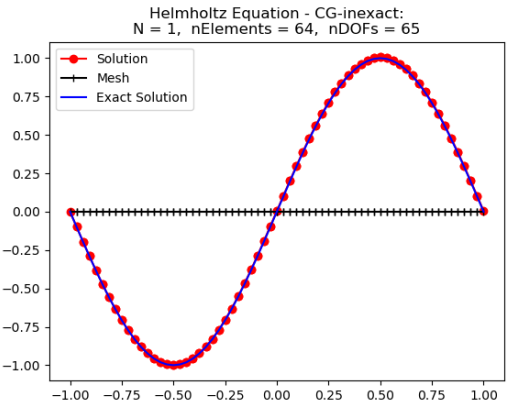
CG	DG

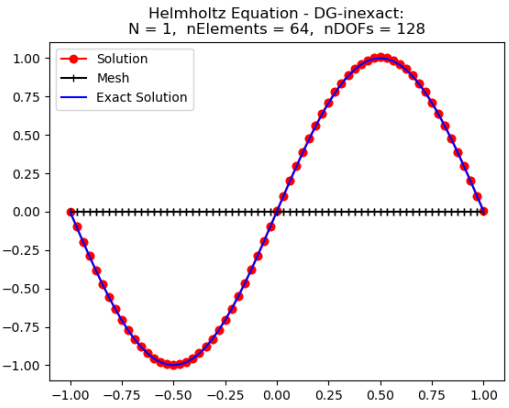

Helmholtz Equation - CG-inexact:
N = 1, nElements = 4, nDOFs = 5
Solution
Mesh
Exact Solution

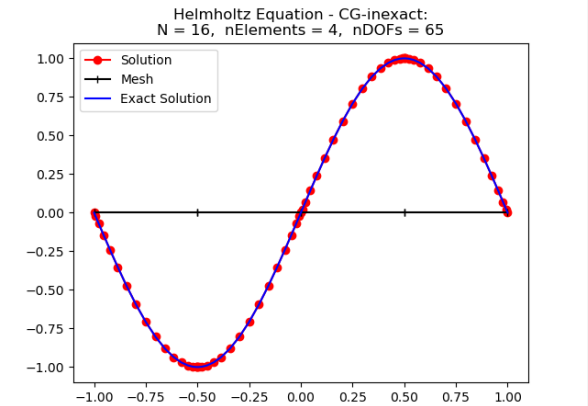

Helmholtz Equation - DG-inexact:
N = 1, nElements = 4, nDOFs = 8
Solution
Mesh
Exact Solution

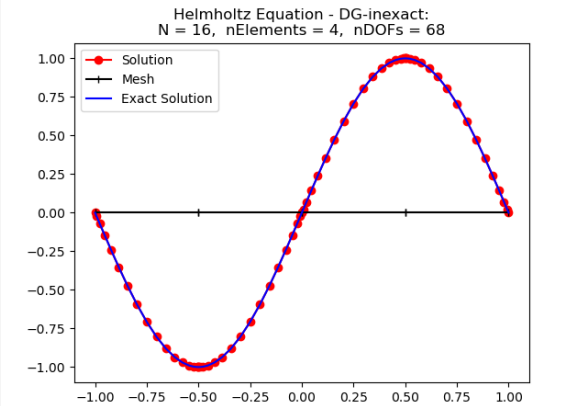

Helmholtz Equation - CG-inexact:
N = 1, nElements = 16, nDOFs = 17
Solution
Mesh
Exact Solution

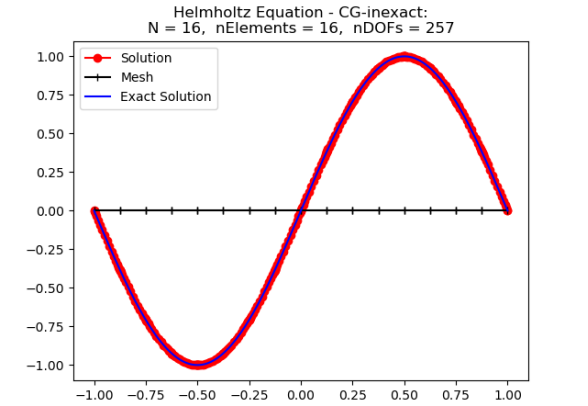

Helmholtz Equation - DG-inexact:
N = 1, nElements = 16, nDOFs = 32
Solution
Mesh
Exact Solution

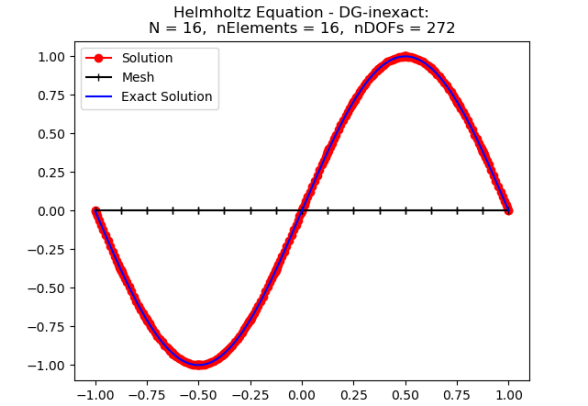

Helmholtz Equation - CG-inexact:
N = 1, nElements = 64, nDOFs = 65
Solution
Mesh
Exact Solution

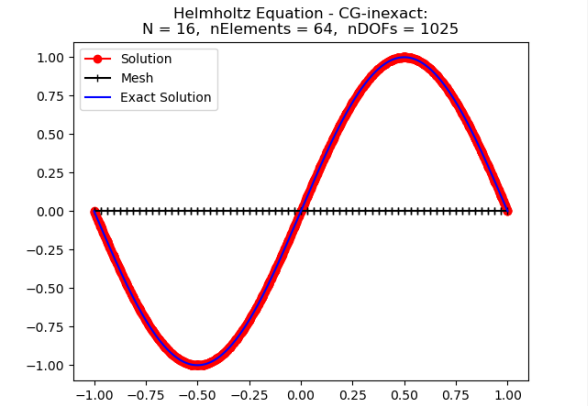

Helmholtz Equation - DG-inexact:
N = 1, nElements = 64, nDOFs = 128
Solution
Mesh
Exact Solution

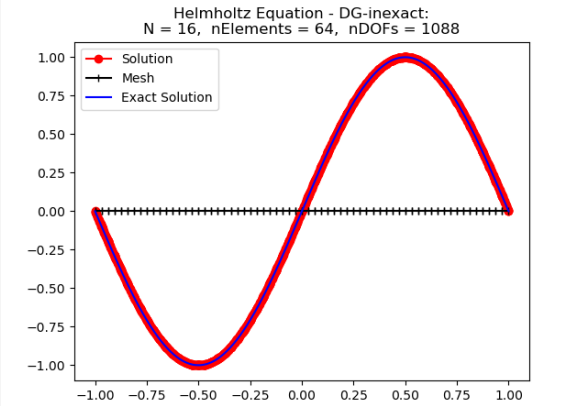

Helmholtz Equation - CG-inexact:
N = 16, nElements = 4, nDOFs = 65
Solution
Mesh
Exact Solution


Helmholtz Equation - DG-inexact:
N = 16, nElements = 4, nDOFs = 68
Solution
Mesh
Exact Solution


Helmholtz Equation - CG-inexact:
N = 16, nElements = 16, nDOFs = 257
Solution
Mesh
Exact Solution


Helmholtz Equation - DG-inexact:
N = 16, nElements = 16, nDOFs = 272
Solution
Mesh
Exact Solution


Helmholtz Equation - CG-inexact:
N = 16, nElements = 64, nDOFs = 1025
Solution
Mesh
Exact Solution


Helmholtz Equation - DG-inexact:
N = 16, nElements = 64, nDOFs = 1088
Solution
Mesh
Exact Solution

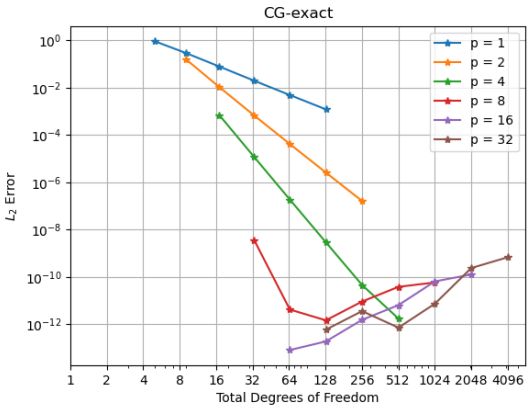
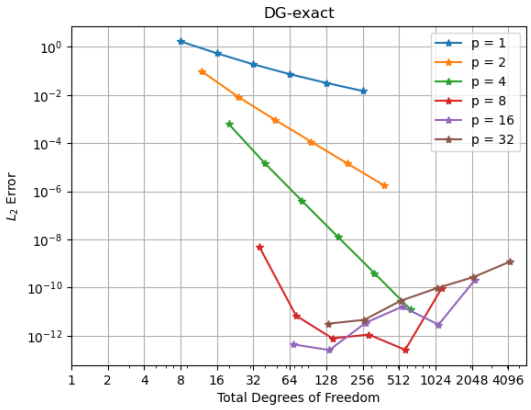
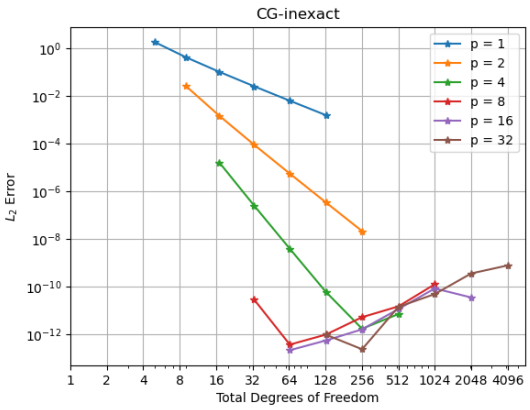
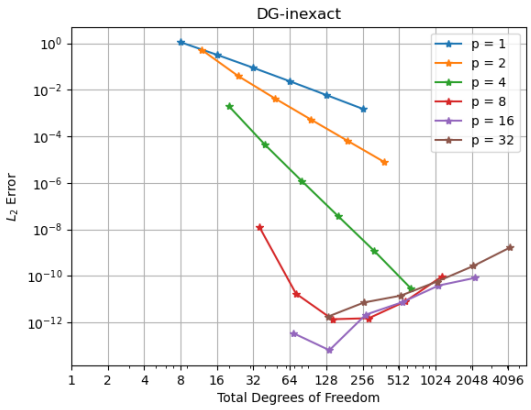
Convergance Study

The `main` program in `examples/math597-P3/main.cpp` runs a convergance study for the following parameter sweeps:

- Scheme : CG, DG
- Integration Method : Exact, Inexact
- Basis Order : 1, 2, 4, 8, 16, 32
- Number of Elements : 4, 8, 16, 32, 64, 128

This is done with MPI on 4 ranks with each rank taking one of the scheme/integration method combos. Running it produces plots of the number of degrees of freedom vs. the L_2 error compared to the exact solution, which is the initial solution advected around the domain once.

The plots are shown below:

CG	DG
	
	

Discussion

CG vs DG

Looking at the convergence plots, we see that the CG cases have a slightly better rate coefficient, though both CG and DG seem to be converging at the same rate. My guess would be this has to do with the numerical flux and the discontinuity across element interfaces. If we look at a plot with lower number of elements and a lower order, the CG plot, while not correct, is closer to the solution than the DG version. And the jumps between elements only increase the error locally. This is of course corrected with higher number of elements of higher orders.

Exact vs Inexact Integration

In terms of error, there is little difference between CG exact vs inexact and DG exact vs inexact. Viewing the plots for the DG solutions, the discontinuities practically disappear.

In terms of speed, when I ran this, the inexact cases finished a good couple seconds ahead of the exact method, which would be useful for saving time in a transient problem.

Numerical Round Off Error

It seems that the round off error is worse than I would have expected. While `double` precision numbers normally have 16 digits of accuracy, we are only able to get to about 12 digits with all methods. And at higher degrees of freedom, round off error drives the error significantly up. I would be interested in looking at how to drive this down, because a high order scheme should be able to give us better numerical precision here.

Boundary Conditions

The boundary conditions have a direct impact on the solution. As Helmholtz equation is a BVP, the main driver is the boundary conditions. When implementing the matrices above, any slight variance to a value on the boundary significantly changed the resulting solution. Thus, accurate boundary conditions must be enforced well!

Conclusion

In this project, we explored Helmholtz equation as a boundary value problem. We implemented a CG and DG scheme using a Laplacian matrix and a system of first order ODEs, respectively. The boundary conditions were enforced through the use of boundary vectors and flux matrices. We showed that we can get expected convergence with a wide range of number of elements and polynomial orders.

References

Giraldo, Francis X. *An Introduction to Element-Based Galerkin Methods on Tensor-Product Bases: Analysis, Algorithms, and Applications*. Vol. 24. Springer Nature, 2020.

In [ ]: