1 Background

The PDE being solved is Laplace's equation in two dimensions,

$$-\nabla^2 \boldsymbol{u} = \boldsymbol{0}, \tag{1}$$

with homogeneous Neumann conditions on the boundary $\partial\Omega$,

$$\nabla \cdot \hat{\boldsymbol{n}} = \boldsymbol{0}. \tag{2}$$

Training meshes are created by first generating 45-90 random points, then triangulating the mesh such that sharp interior angles are avoided. These meshes are then used to discretize the PDE using piecewise-linear (triangular) finite elements.

2 Training Algorithm

The training routine is formally described in Alg. 1. In summary, each matrix in a training batch has its last row replaced with 1's (to force it to be solvable w/ minimal norm solution), and then combined into a large block diagonal system. This block-diagonal system is converted directly into a graph with fine-fine connections removed, then passed into the interpolation weight graph network.

The graph network will return a new list of edges of the same cardinality as that which was passed in, meaning we have interpolation weights between coarse-coarse and fine-coarse nodes. Fine-fine weights can be trivially added in by setting their respective entries to 1. This new list of edges is then transformed back into a matrix, and the columns corresponding to fine nodes are removed to get the correct $N_f \times N_c$ interpolation matrix.

This interpolation matrix is then passed to the AMG loss function, described in Alg. 2, which runs several AMG iterations. This loss function will randomly generate several starting guess vectors, $\boldsymbol{x}_i$, then iterate directly on these to solve $\boldsymbol{Ax} = \boldsymbol{0}$. A Jacobi relaxation scheme is used with weight $\omega = \frac{2}{3}$. After each multigrid iteration, the error (norm of each $\boldsymbol{x}_i$) is recorded. After a sufficient number of iterations, the convergence factor for each vector is (approximately) computed using the geometric mean and the largest convergence factor is returned. In practice, if $\boldsymbol{x}_i$ are randomly generated for each invocation of the loss function, then good training convergence can be achieved.

Algorithm 1 Main training loop

```
procedure TrainingLoop
    for each training batch do
        Let $\boldsymbol{A}_{I_1} \dots \boldsymbol{A}_{I_n}$ be the $n$ matrices in the current batch. Also $\boldsymbol{c}_{I_1} \dots \boldsymbol{c}_{I_n}$ to be the
    coarse-fine partitionings for each $\boldsymbol{A}$.

        for $\boldsymbol{A}_i$ in the batch do
            Set the last row of $\boldsymbol{A}_i$ to consist of all 1's
        end for

        $\boldsymbol{A}_{\text{combined}} := \begin{bmatrix} \boldsymbol{A}_{I_1} & & & \\ & \boldsymbol{A}_{I_2} & & \\ & & \ddots & \\ & & & \boldsymbol{A}_{I_n} \end{bmatrix}$
        $\boldsymbol{c}_{\text{combined}} := \begin{bmatrix} \boldsymbol{c}_{I_1}^T & \boldsymbol{c}_{I_2}^T & \dots & \boldsymbol{c}_{I_n}^T \end{bmatrix}^T$

        $\boldsymbol{v}$, $\boldsymbol{e}$ := MatrixToGraph($\boldsymbol{A}_{\text{combined}}$, $\boldsymbol{c}_{\text{combined}}$)
        $\boldsymbol{e}_P$ := GraphNet($\boldsymbol{v}$, $\boldsymbol{e}$)          ▷ Get prolongation edge weights from graphnet

        $\boldsymbol{P}_{\text{full}}$ := GraphToMatrix($\boldsymbol{v}$, $\boldsymbol{e}_P$)   ▷ Get the full $n_F \times n_F$ interpolation matrix
        for coarse node $i$ in $\boldsymbol{c}_{\text{combined}}$ do
            $(\boldsymbol{P}_{\text{full}})_{ii} \leftarrow 1$                  ▷ Set all coarse-coarse connections to be 1
        end for
        Create $\boldsymbol{P}$ by removing all columns from $\boldsymbol{P}_{\text{full}}$ that correspond to fine nodes

        $l$ := AMGLoss($\boldsymbol{P}$, $\boldsymbol{A}_{\text{combined}}$, 10)
        Optimizer($l$)
    end for
end procedure
```

Algorithm 2 Training loss function

```
procedure JACOBI($\boldsymbol{A}$, $\boldsymbol{X}$, $\omega$, $\nu$)
    $\boldsymbol{D}$ := the diagonal matrix of $\boldsymbol{A}$
    for $i = 1 \dots \nu$ do
        $\boldsymbol{X} \leftarrow \boldsymbol{X} - \omega \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{X}$
    end for
    return $\boldsymbol{X}$
end procedure
procedure AMGLOSS($\boldsymbol{P}$, $\boldsymbol{A}$, $\ell$)
    $\boldsymbol{A}_H := \boldsymbol{P}^T\boldsymbol{A}\boldsymbol{P}$
    Define $\boldsymbol{X} \in \mathbb{R}^{n_F \times T}$ to be a matrix whose $T$ columns are randomly generated from a
normal distribution. Each column is normalized with the 2-norm.
    Define $\boldsymbol{E} \in \mathbb{R}^{\ell \times T}$ to hold the error (norm of $\boldsymbol{x}_i$) at each iteration.
    for $i = 1 \dots \ell$ do
        $\boldsymbol{X} \leftarrow$ JACOBI($\boldsymbol{A}$, $\boldsymbol{X}$, $\omega = \frac{2}{3}$, $\nu = 2$)          ▷ Pre-relaxation
        $\boldsymbol{E_H} := -\boldsymbol{A_H}^{-1}\left(\boldsymbol{P}^T\boldsymbol{A}\boldsymbol{X}\right)$          ▷ Coarse-grid correction
        $\boldsymbol{X} \leftarrow \boldsymbol{X} + \boldsymbol{P}\boldsymbol{E_H}$          ▷ Interpolate to fine grid
        $\boldsymbol{X} \leftarrow$ JACOBI($\boldsymbol{A}$, $\boldsymbol{X}$, $\omega = \frac{2}{3}$, $\nu = 2$)          ▷ Post-relaxation
        $\boldsymbol{E}_{ij} \leftarrow \|\boldsymbol{x}_j\|$          ▷ Store the norm of each test vector
    end for
    $a := \min\left\{\left\lfloor \frac{T}{2} \right\rfloor, 6\right\}$
    return $\max\left((\boldsymbol{e}_T / \boldsymbol{e}_{T-a})^{1/(a-1)}\right)$          ▷ Return approx. convergence factor
end procedure
```

Algorithm 3 Auxiliary matrix to graph routine

```
procedure MATRIXTOGRAPH($\boldsymbol{A}, \boldsymbol{c}$)
    $\boldsymbol{v} = \{\}$          ▷ Create an empty ordered list of vertices
    $\boldsymbol{e} = \{\}$          ▷ Create an empty ordered list of edges
    for $i = 1 \dots n$ do
        for $j = 1 \dots n$ do
            if $\boldsymbol{A}_{ij} \neq 0$ then
                Add node $ij$ to $\boldsymbol{v}$
                if nodes $i$ and $j$ are both not fine then
                    Add edge $ij$ to $\boldsymbol{e}$ with weight $|\boldsymbol{A}_{ij}|$
                end if
            end if
        end for
    end for
    return $\boldsymbol{v}$, $\boldsymbol{e}$
end procedure
```