1 Background

Currently, we are attempting to learn an interpolation operator $\boldsymbol{P}$ given a pre-existing aggregation of some system $\boldsymbol{A}$ by first learning $\hat{\boldsymbol{P}}$, then forming $\boldsymbol{P} = \hat{\boldsymbol{P}}\text{Agg}$, for an aggregate assignment matrix Agg. This is analogous to smoothed aggregation AMG, where the $\hat{\boldsymbol{P}}$ is defined as $\hat{\boldsymbol{P}} = \left(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\right)$ for some smoother $\boldsymbol{M}^{-1}$[?].

To test the method and neural network, we are fixing the problem to be a $n = 9$ node 1D Poisson problem discretized with finite differences. The nodes will be aggregated into 3 equal aggregates of 3 nodes each. We will examine using both Dirichlet (2) and Neumann (3) conditions and how the network output differs from the results smoothed aggregation gives.

2 Dirichlet Boundaries

For the dirichlet boundary case, we are using the following $\boldsymbol{A}$ matrix and aggregate information.

$$\boldsymbol{A} = \begin{bmatrix} 2 & -1 & & & & & & & \\ -1 & 2 & -1 & & & & & & \\ & -1 & 2 & -1 & & & & & \\ & & -1 & 2 & -1 & & & & \\ & & & -1 & 2 & -1 & & & \\ & & & & -1 & 2 & -1 & & \\ & & & & & -1 & 2 & -1 & \\ & & & & & & -1 & 2 & -1 \\ & & & & & & & -1 & 2 \end{bmatrix} \tag{1}$$

$$\text{Agg} = \begin{bmatrix} 1 & & \\ 1 & & \\ 1 & & \\ & 1 & \\ & 1 & \\ & 1 & \\ & & 1 \\ & & 1 \\ & & 1 \end{bmatrix} \tag{2}$$

Using a Jacobi smoother with $\omega = \frac{2}{3}$, the following operator is obtained for smoothed aggregation:

$$\boldsymbol{I} - \frac{2}{3}\boldsymbol{D}^{-1}\boldsymbol{A} = \begin{bmatrix} 1/3 & 1/3 & & & & & & & \\ 1/3 & 1/3 & 1/3 & & & & & & \\ & 1/3 & 1/3 & 1/3 & & & & & \\ & & 1/3 & 1/3 & 1/3 & & & & \\ & & & 1/3 & 1/3 & 1/3 & & & \\ & & & & 1/3 & 1/3 & 1/3 & & \\ & & & & & 1/3 & 1/3 & 1/3 & \\ & & & & & & 1/3 & 1/3 & 1/3 \\ & & & & & & & 1/3 & 1/3 \end{bmatrix}, \tag{3}$$

while from the graph network we obtain

$$\hat{\boldsymbol{P}} = \begin{bmatrix} 0.433 & 0.441 & & & & & & & \\ 0.333 & 0.438 & 0.221 & & & & & & \\ & 0.059 & 0.931 & 0.759 & & & & & \\ & & 0.759 & 0.931 & 0.957 & & & & \\ & & & 0.957 & 0.931 & 0.957 & & & \\ & & & & 0.957 & 0.931 & 0.759 & & \\ & & & & & 0.759 & 0.931 & 0.059 & \\ & & & & & & 0.221 & 0.438 & 0.333 \\ & & & & & & & 0.441 & 0.433 \end{bmatrix}. \tag{4}$$

Thus, right multiplying the "smoothers" gives us

$$\boldsymbol{P}_{SA} = \begin{bmatrix} 2/3 & & \\ 1 & & \\ 2/3 & 1/3 & \\ 1/3 & 2/3 & \\ & 1 & \\ & 2/3 & 1/3 \\ & 1/3 & 2/3 \\ & & 1 \\ & & 2/3 \end{bmatrix} \tag{5}$$

and

$$\boldsymbol{P}_{ML} = \begin{bmatrix} 0.8746777 & & \\ 0.99204123 & & \\ 0.98971415 & 0.7593956 & \\ 0.7593956 & 1.8878527 & \\ & 2.8447313 & \\ & 1.8878527 & 0.7593956 \\ & 0.7593956 & 0.9897142 \\ & & 0.99204195 \\ & & 0.8746774 \end{bmatrix}. \tag{6}$$

Note that the columns of $\boldsymbol{P}_{ML}$ have inconsistent scaling when compared to the SA operator; in practice this does not affect convergence as the scaling is undone by the coarse solve (a proof for this is given in A).

The above two interpolation operators give a convergence of 0.37275 for SA and 0.29470 for ML, meaning the graphnet is able to learn a more optimal interpolation than using only Jacobi to smooth the aggregates.

3 Neumann Boundaries

In the Neumann case, we have A defined as

$$\boldsymbol{A} = \begin{bmatrix} 1 & -1 & & & & & & & \\ -1 & 2 & -1 & & & & & & \\ & -1 & 2 & -1 & & & & & \\ & & -1 & 2 & -1 & & & & \\ & & & -1 & 2 & -1 & & & \\ & & & & -1 & 2 & -1 & & \\ & & & & & -1 & 2 & -1 & \\ & & & & & & -1 & 2 & -1 \\ & & & & & & & -1 & 1 \end{bmatrix}, \tag{7}$$

and reuse the aggregate information in (2). Again using a Jacobi smoother with $\omega = \frac{2}{3}$, the following operator is obtained for smoothed aggregation:

$$\boldsymbol{I} - \frac{2}{3}\boldsymbol{D}^{-1}\boldsymbol{A} = \begin{bmatrix} 1/3 & 2/3 & & & & & & & \\ 1/3 & 1/3 & 1/3 & & & & & & \\ & 1/3 & 1/3 & 1/3 & & & & & \\ & & 1/3 & 1/3 & 1/3 & & & & \\ & & & 1/3 & 1/3 & 1/3 & & & \\ & & & & 1/3 & 1/3 & 1/3 & & \\ & & & & & 1/3 & 1/3 & 1/3 & \\ & & & & & & 1/3 & 1/3 & 1/3 \\ & & & & & & & 2/3 & 1/3 \end{bmatrix}. \tag{8}$$

Notice the major difference is the 2/3 in the top-left and bottom-right corners. Meanwhile with ML, we obtain a $\hat{\boldsymbol{P}}$ of

$$\hat{\boldsymbol{P}} = \begin{bmatrix} 4.444 & 4.432 & & & & & & & \\ 4.449 & 4.445 & 4.339 & & & & & & \\ & 4.42 & 3.786 & 3.869 & & & & & \\ & & 3.868 & 3.784 & 2.483 & & & & \\ & & & 2.483 & 3.784 & 2.483 & & & \\ & & & & 2.483 & 3.784 & 3.868 & & \\ & & & & & 3.869 & 3.786 & 4.42 & \\ & & & & & & 4.339 & 4.445 & 4.449 \\ & & & & & & & 4.432 & 4.444 \end{bmatrix}. \tag{9}$$

Forming the smoothers gives

$$\boldsymbol{P}_{SA} = \begin{bmatrix} 1 & & \\ 1 & & \\ 2/3 & 1/3 & \\ 1/3 & 2/3 & \\ & 1 & \\ & 2/3 & 1/3 \\ & 1/3 & 2/3 \\ & & 1 \\ & & 1 \end{bmatrix} \tag{10}$$

and

$$\boldsymbol{P}_{ML} = \begin{bmatrix} 8.876631 & & & \\ 13.232742 & & & \\ 8.206335 & 3.8687856 & & \\ 3.868087 & 6.266823 & & \\ & 8.749655 & & \\ & 6.266823 & 3.868087 & \\ & 3.8687856 & 8.206335 & \\ & & 13.232742 & \\ & & 8.87663 & \end{bmatrix}. \tag{11}$$

Note that with the Jacobi smoother, we appear to be handling the Neumann boundary conditions differently, while the ML smoother does not appear to handle them any differently from the Dirichlet case. Because of this, the convergence factors are 0.35937 for SA and 0.42993 for ML.

3.1 Learning Neumann Conditions

As an experiment to see if the network can even represent the Neumann conditions at all, it was first trained in a supervised fashion to mimic the output of the SA smoother, then allowed to learn with the unsupervised loss to "explore" and hopefully obtain a more optimal interpolator.

$$\hat{\boldsymbol{P}} = \begin{bmatrix} 0.862 & 0.602 & & & & & & & \\ 0.984 & 0.88 & 0 & & & & & & \\ & 0.961 & 0.416 & 0.501 & & & & & \\ & & 0.501 & 0.416 & 0.432 & & & & \\ & & & 0.432 & 0.416 & 0.432 & & & \\ & & & & 0.432 & 0.416 & 0.501 & & \\ & & & & & 0.501 & 0.416 & 0.961 & \\ & & & & & & 0 & 0.88 & 0.984 \\ & & & & & & & 0.602 & 0.862 \end{bmatrix} \tag{12}$$

and

$$\boldsymbol{P}_{SA} = \begin{bmatrix} 1.4636066 & & & \\ 1.863446 & & & \\ 1.3770666 & 0.50145984 & & \\ 0.50145984 & 0.8478845 & & \\ & 1.2795522 & & \\ & 0.8478845 & 0.50145984 & \\ & 0.50145984 & 1.3770669 & \\ & & 1.863446 & \\ & & 1.4636066 & \end{bmatrix}. \tag{13}$$

This appears to indicate that the network can indeed represent the interpolation for Neumann conditions, and perhaps some tweaking of parameters is needed in order to push the network to learn such a representation. For the $\boldsymbol{P}$ above, a convergence factor of 0.34305 is obtained, matching SA's 0.35937.

A Proof of column scaling

The multigrid cycle is invariant to any nonzero scaling of the columns of the interpolation operator.

Proof. Let $\boldsymbol{P}$ be the unscaled interpolation operator, and $\bar{\boldsymbol{P}} = \boldsymbol{P}\boldsymbol{\Sigma}$ be some operator whose columns are scaled by a square, diagonal scaling matrix $\boldsymbol{\Sigma}$. We assume $\boldsymbol{\Sigma}$ to have nonzero values on the diagonal, i.e. that $\boldsymbol{\Sigma}$ is full rank and thus invertible.

Using the scaled interpolator, define the multigrid coarse grid solve as

$$\boldsymbol{A}_H \boldsymbol{e}_H = \boldsymbol{r}_H = \bar{\boldsymbol{P}}^T \boldsymbol{r}_h. \tag{14}$$

This implies that

$$\begin{aligned} \boldsymbol{e}_H &= \boldsymbol{A}_H^{-1} \bar{\boldsymbol{P}}^T \boldsymbol{r}_h && (15)\\ &= \left(\bar{\boldsymbol{P}}^T \boldsymbol{A}_h \bar{\boldsymbol{P}}\right)^{-1} \bar{\boldsymbol{P}}^T \boldsymbol{r}_h && (16)\\ &= \left(\boldsymbol{\Sigma}^T \boldsymbol{P} \boldsymbol{A}_h \boldsymbol{P}^T \boldsymbol{\Sigma}\right)^{-1} \boldsymbol{\Sigma}^T \boldsymbol{P}^T \boldsymbol{r}_h. && (17) \end{aligned}$$

If we interpolate (17) to the fine grid, we obtain

$$\begin{aligned} \boldsymbol{e}_h &= \bar{\boldsymbol{P}} \boldsymbol{e}_H && (18)\\ &= \boldsymbol{P}\boldsymbol{\Sigma} \left(\boldsymbol{\Sigma}^T \boldsymbol{P}^T \boldsymbol{A}_h \boldsymbol{P} \boldsymbol{\Sigma}\right)^{-1} \boldsymbol{\Sigma}^T \boldsymbol{P}^T \boldsymbol{r}_h && (19)\\ &= \boldsymbol{P}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^{-1} \left(\boldsymbol{P}^T \boldsymbol{A}_h \boldsymbol{P}\right)^{-1} \boldsymbol{\Sigma}^{-T} \boldsymbol{\Sigma}^T \boldsymbol{P}^T \boldsymbol{r}_h && (20)\\ &= \boldsymbol{P} \left(\boldsymbol{P}^T \boldsymbol{A}_h \boldsymbol{P}\right)^{-1} \boldsymbol{P}^T \boldsymbol{r}_h. && (21) \end{aligned}$$

Thus, the column scalings drop out and we recover the regular multigrid cycle. □