MAP7386 Project

Justin Gosselin, Corey Prachniak, Kurtis Doobay

May 4, 2021

1 Introduction

Domain decomposition methods have become ubiquitous in solving partial differential equations over recent decades due to their capability to exploit parallelism. Rather than solving a boundary-value problem on sequentially the entire domain, the domain is instead split up into subdomains and the BVP is solved on each of those subdomains. Because the subdomains are independent, except at the boundaries, the BVP can be solved in parallel on those subdomains. Some agreement is then necessary at the boundaries, but this allows for a PDE to solved much faster than a usual sequential algorithm.

Typically, a domain decomposition method is implemented as a *preconditioner* that is then used in an iterative solver [1]. Many PDEs discretized with finite difference and finite element methods result in a linear system $Ax = b$ to be solved for x. Although A is most often a sparse matrix, convergence with reasonable accuracy can be difficult to achieve in solving this system with direct methods (e.g., Gaussian elimination) due to the high condition number of A that usually arises in such discretizations. A preconditioner P applies the transformation $P^{-1}A$ to A such that P has a lower condition number. This essentially makes the system easier to solve for, with better convergence and accuracy. Iterative methods, such as the conjugate gradient method and the generalized minimal residual method (GMRES), are particularly useful for preconditioners because their rate of convergence is dependent on the condition number of the coefficient matrix. The construction of such a preconditioner is based on the decomposition of the BVP's domain.

Overlapping domain decomposition methods have also been introduced, in which the subdomains overlap beyond their boundaries. The additive Schwarz method (AS) is a common example that adds the results of each

subdomain approximation and constructs the preconditioner by extending each subdomain to a larger domain [2]; i.e.,

$$P^{-1}A \equiv (T_1 + \cdots + T_N)x = b,$$

where T_i represents the matrix form of the subdomain problem to solve and its action on b can be carried out in parallel. AS has proven itself to be a very useful and powerful tool for preconditioning ill-conditioned systems and is available in many well-known parallel libraries. Several extensions to it have been considered, including restrictive additive Schwarz (RAS) that was found accidentally by removing some parts of the communication routine [3].

In this paper, we describe our efforts to replicate the results of the paper that introduced RAS [3] by implementing both AS and RAS, as well as their versions with harmonic extensions. We evaluate our implementation with two of the three case studies introduced in the paper, the 2D convection-diffusion equation and the 2D Helmholtz equation. Our results follow a similar pattern to those found in the reference paper, although we made some modifications in our evaluation.

The rest of this paper is organized as follows. In Section 2, we discuss the theory behind the preconditioners. In Sections 3 and 4, we discuss our replication of the two aforementioned case studies, including the discretizations used. Finally, we provide our concluding remarks in Section 5.

2 The RA and RAS Preconditioners

The goal of this section is to outline the construction of the RA and RAS preconditioners. Consider a system given by:

$$Ax = b$$

where A is an $n \times n$ non-zero sparse matrix. Define a graph $G = (W, E)$ so that the nodes $W = \{1, ..., n\}$ represent the n unknowns, and the edges $E := \{(i, j) : A_{i,j} \neq 0\}$ are associated with the non-zero elements of A. Throughout this section, we'll consider the index i to be an arbitrary node of W. For many instances of analysis, the matrix A will be symmetric which would mean the adjacency graph of G is undirected. Now, suppose the nodes W are partitioned into $N \in \mathbb{N}$ components $\{W_i^0\}_{i=1}^N$. Such a partition is called a minimal overlap partition. The word minimal refers to the fact that

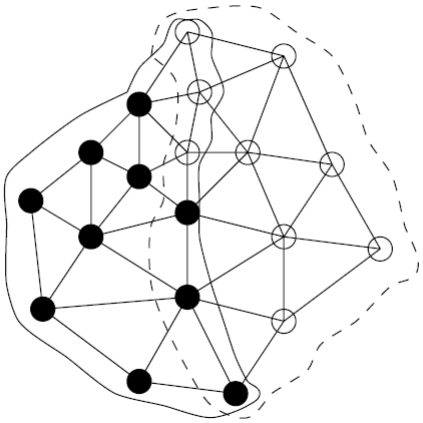
there is no intersection among components. The first overlapping partition $\{W_i^1\}_{i=1}^N$ is defined by:

$$W_i^1 := \text{nodes}(\text{nbhd}_G(W_i^0))$$

This just means W_i^1 is the set of nodes of the neighborhood of W_i^0. Using this procedure recursively, we can define the δ-overlap partition for an integer $\delta \geq 0$:

$$W = \bigcup_{i=1}^{N} W_i^\delta$$

It should be noted that, as the name suggests, the overlapping partition will contain overlap among components. Here is a figure [4] depicting the situation when W is partitioned into two components W_1^0 and W_2^0. The black nodes represent W_1^0 and the white nodes represent W_2^0. The nodes inside of the solid line represent W_1^1 and those inside the dashed line represent W_2^1.



Next, let R_i^δ be the $n \times n$ sub-identity matrix with 1 in its ith diagonal entry if the corresponding node belongs to W_i^δ, and 0 otherwise. With this define:

$$A_i = R_i^\delta A R_i^\delta$$

As the paper notes, while A_i may not be invertible, it can be inverted with respect to a subspace L_i:

$$A_i^{-1} \equiv ((A_i)|_{L_i})^{-1}$$

with L_i representing the span of W_i^δ in $\mathbb{R}^n$. With this we can define the AS, RAS, ASH, and RASH preconditioners as follows:

$$M_{RA}^{-1} := \sum R_i^\delta A_i^{-1} R_i^\delta \tag{1}$$

$$M_{RAS}^{-1} := \sum R_i^0 A_i^{-1} R_i^\delta \tag{2}$$

$$M_{ASH}^{-1} := \sum R_i^\delta A_i^{-1} R_i^0 \tag{3}$$

$$M_{RASH}^{-1} := \sum R_i^0 A_i^{-1} R_i^0 \tag{4}$$

3 Case Study 1

In this section, we describe our replication of Case Study 1 in [3], corresponding to the 2-dimensional convection-diffusion equation with a finite difference discretization. We describe the equation, the discretization scheme used, and the numerical results, including those obtained from preconditioning the linear system using both the AS and RAS preconditioners described in Section 2.

3.1 The 2D Convection-Diffusion Equation

As per the reference paper [3], we consider the 2D convection-diffusion equation defined on the unit square with zero Dirichlet boundary conditions with a prescribed initial condition for time $t = 0$[1]; i.e.,

$$\begin{aligned} u_t = \Delta u + b_1 u_x + b_2 u_y, &\quad x \in (0,1),\ y \in (0,1),\ t > 0 \\ u(0,y,t) = u(1,y,t) = 0, &\quad y \in (0,1),\ t > 0 \\ u(x,0,t) = u(x,1,t) = 0, &\quad x \in (0,1),\ t > 0 \\ u(x,y,0) = u_0, &\quad x \in (0,1),\ y \in (0,1), \end{aligned}$$

where $u = u(x,y,t)$, u_0 is given, and b_1, b_2 are the convection coefficients, generally the x- and y-components of velocity. Note if $b_1 = b_2 = 0$, the equation reduces to the Poisson case.

[1]The paper does not prescribe an initial condition and also has a negative diffusive term. For our purposes of modeling dye transport through time, we instead augment this information and model positive diffusion. The numerical scheme we use remains the same as in the paper.

As the name implies, the convection-diffusion equation models both convective and diffusive processes and is a combination of those two equations. An example of a process modeled by the equation is that of dye transport in which a chemical or dye is placed in a flowing fluid. As the fluid flows (convection), the dye will mix throughout the fluid and move from areas of higher concentration to areas of lower concentration (diffusion). $u(x, y, t)$ in this case then models the concentration of dye at a particular location (x, y) in the domain at a time t. As time advances, the dye concentration will spread throughout the domain subject to the convection of the fluid.

3.2 Discretization

Before we discuss discretizing the equation, we need to discuss discretizing the domain. Here we use a standard 2-dimensional $M \times M$ non-overlapping grid corresponding to the unit square domain with equal spacing $h = 1/M$ in both spatial dimensions. Thus, the x-component of the domain is partitioned into M points along the mesh that we denote by $x_j = jh$ for $j = 1, 2, \ldots, M-1$. Similarly, the y-component is partitioned as $y_k = kh$ for $k = 1, 2, \ldots, M-1$. To prescribe the boundary conditions, we take $x_0 = x_M = 0$ and $y_0 = y_M = 0$. For the temporal discretization, we partition time into N time steps with $t_n = nm$ for $n = 1, 2, \ldots, N$ with $m = 1/N$ with $t_0 = u_0$. Then, $U^n_{j,k}$ denotes the approximation of $u(x_j, y_k, t_n)$.

We need to discretize both the diffusion and convection terms together. Because we are working in two dimensions and have a Laplacian operator in the diffusion term, a five-point finite difference scheme lends itself well. This stencil takes the four neighbors of the current point, as well as the point itself, and is of order $O(h^2)$ by approximating the second derivatives. Thus, we can discretize the diffusion term as

$$\Delta u = \frac{1}{h^2}\left(U^n_{j+1,k} + U^n_{j-1,k} + U^n_{j,k+1} + U^n_{j,k-1} - 4U^n_{j,k}\right) + O(h^2)$$

The convection term requires more consideration. While it may seem intuitive to use a second-order central finite difference for the u_x and u_y terms, this can lead to oscillatory behavior in the context of convection [5]. Instead, a first-order *upwinding* scheme is favorable. Upwinding schemes are similar to first-order backward and forward finite differences, except the choice of backward or forward is based on the direction of the solution. In our case, the convection occurs in the *positive* x- and y-directions as the fluid moves

in the positive direction. Thus, we use the following discretization for the convection terms.

$$b_1 u_x = \frac{b_1}{h}\left(U^n_{j,k} - U^n_{j-1,k}\right) + O(h)$$
$$b_2 u_y = \frac{b_2}{h}\left(U^n_{j,k} - U^n_{j,k-1}\right) + O(h)$$

Finally, for the temporal discretization, we use an implicit scheme so that we can solve the solution using matrix-inverse methods. While more computationally expensive than an explicit scheme, an implicit scheme also has the added benefit of greater stability. Putting it all together, we obtain our finite difference scheme of the convection-diffusion equation:

$$\frac{U^{n+1}_{j,k} - U^n_{j,k}}{m} = \frac{1}{h^2}\left(U^{n+1}_{j+1,k} + U^{n+1}_{j-1,k}U^{n+1}_{j,k+1} + U^{n+1}_{j,k-1} - 4U^{n+1}_{j,k}\right) + \frac{b_1}{h}\left(U^{n+1}_{j,k} - U^{n+1}_{j-1,k}\right) + \frac{b_2}{h}\left(U^{n+1}_{j,k} - U^{n+1}_{j,k-1}\right) + O(h^2)$$

This can be rewritten as

$$U^n_{j,k} = U^{n+1}_{j,k}\left(\frac{4m}{h^2} - \frac{b_1 m}{h} - \frac{b_2 m}{h}\right) + U^{n+1}_{j-1,k}\left(-\frac{m}{h^2} + \frac{b_1 m}{h}\right) + U^{n+1}_{j,k-1}\left(-\frac{m}{h^2} + \frac{b_2 m}{h}\right) + U^{n+1}_{j+1,k}\left(-\frac{m}{h^2}\right) + U^{n+1}_{j,k+1}\left(-\frac{m}{h^2}\right),$$

or, in matrix-vector form,

$$(I_{(M-1)^2} + mA)U^{n+1} = U^n$$

where U^n is the $(M-1)^2$-vector of lexicographically stacked values of $U^n_{j,k}$ and A is the $(M-1)^2 \times (M-1)^2$ block tridiagonal matrix; i.e.,

$$U^n = \begin{pmatrix} U^n_{1,1} \\ U^n_{1,2} \\ \vdots \\ U^n_{1,M-1} \\ U^n_{2,1} \\ \vdots \\ U^n_{M-2,M-1} \\ U^n_{M-1,1} \\ \vdots \\ U^n_{M-1,M-1} \end{pmatrix}, \quad A = \begin{pmatrix} B & I_\delta & 0 & \cdots & 0 \\ I_\beta & B & I_\delta & \ddots & \vdots \\ 0 & I_\beta & \ddots & \ddots & 0 \\ \vdots & \ddots & I_\beta & B & I_\delta \\ 0 & \dots & 0 & I_\beta & B \end{pmatrix}$$

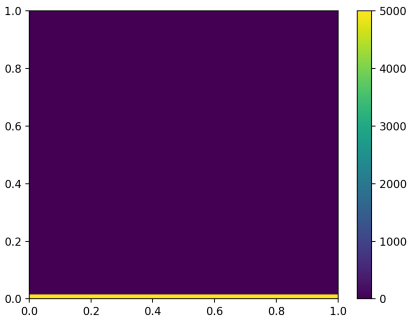
A is defined in terms of the submatrices each of size $(M-1)\times(M-1)$

$$B = \begin{pmatrix} \alpha & \delta & 0 & \dots & 0 \\ \gamma & \alpha & \delta & \ddots & \vdots \\ 0 & \gamma & \ddots & \ddots & 0 \\ \vdots & \ddots & \gamma & \alpha & \delta \\ 0 & \dots & 0 & \gamma & \alpha \end{pmatrix}, \quad I_\beta = \beta I_{M-1}, \quad I_\delta = \delta I_{M-1}$$

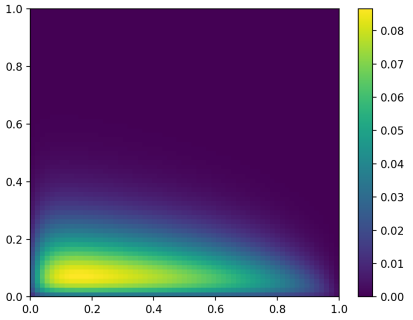
with $\alpha = \frac{4}{h^2} - \frac{b_1}{h} - \frac{b_2}{h}$, $\delta = -\frac{1}{h^2}$, $\beta = \delta + \frac{b_1}{h}$, and $\gamma = \delta + \frac{b_2}{h}$. It is important to note that since the Dirichlet boundary conditions are zero, all $U_{0,k}$, $U_{M,k}$, $U_{j,0}$, and $U_{j,M}$ terms drop out of the difference equation for all $j,k = 1,2,\dots,M-1$. Thus, these values do not need to be considered in the linear system; in the case where we have nonzero Dirichlet boundary conditions, we would need to move them to the right-hand U^n vector.

3.3 Numerical Results

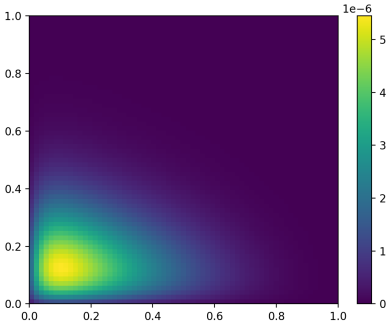
We implemented the finite difference scheme discussed above in Python 3.8 for $M = 64$ and $N = 5$. For each time step, we solve the linear system by first preconditioning the matrix $I_{(M-1)^2} + mA$ using AS or RAS and then solve the system with the GMRES accelerator with `scipy.sparse.linalg.gmres`. We restart the accelerator at 30 iterations, as in the reference paper [3], and set the maximum number of iterations at 10000. We collected the results of each time step into one list and plotted the results as a colormap showing the value of $u(x,y,t)$, representing dye concentration, at each point in the mesh for a given timestep. We considered both the Poisson case ($b_1 = b_2 = 0$) and the case $b_1 = 10, b_2 = 20$ as per the paper. At time $t = 0$, we prescribed an initial condition of $u(x,1,0) = 5000$; i.e., $U^0_{1,k} = 5000$ for all k in the mesh. Figure 1 shows results at various timesteps obtained with the RAS preconditioner with $\delta = 3$ and $subd = 16$.



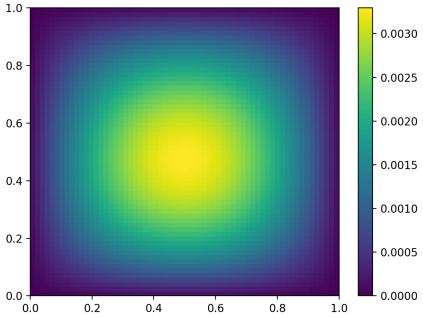
(a) $b_1 = 10, b_2 = 20$, timestep 0



(b) $b_1 = 10, b_2 = 20$, timestep 2



(c) $b_1 = 10, b_2 = 20$, timestep 5



(d) $b_1, b_2 = 0$ (Poisson), timestep 5

Figure 1: Numerical solution of various time steps and the Poisson case. Color represents the concentration of dye (u) subject to diffusion and convection.

Figure 4a shows the initial condition at time $t = 0$ whereby we have placed a concentration of dye at the bottom of the domain. Figure 1b shows the numerical approximation two timesteps later. We notice that the dye has begun to diffuse as it travels from areas of higher concentration to areas of lower concentration. In Figure 1c, representing timestep $t = 2$, the dye has undergone significant convection as it has moved toward the left and upwards with the fluid. Note that because $b_2 > b_1$, the y-component of velocity is grater, and therefore the flow is larger in the y-direction, as evidenced by the plot. Figure 1d shows the Poisson case when $b_1 = b_2 = 0$ at the final timestep $t = 5$; note that the dye has evenly diffused throughout the fluid as

expected, as there is no convection process.

For comparison of AS and RAS, we measured the number of iterations needed by GMRES at a chosen timestep $t = 1$. We ran the preconditioners for $\delta = 0, 1, 2, 3$ and for 16 and 32 subdomains. Although the paper uses a mesh size of 128×128 with 16 and 64 subdomains, we found this to take prohibitively long to execute due to the large size of the coefficient matrix and instead chose 64×64. Table 1 summarizes the results.

	$subd = 16$				$subd = 32$			
	$b_1 = b_2 = 0$		$b_1 = 10, b_2 = 20$		$b_1 = b_2 = 0$		$b_1 = 10, b_2 = 20$	
δ	AS	RAS	AS	RAS	AS	RAS	AS	RAS
1	31	32	20	20	43	59	28	32
2	26	28	17	16	28	30	19	17
3	20	26	12	10	22	27	15	14

Table 1: Iteration counts for GMRES with the AS and RAS preconditioners. GMRES restarts at 30 iterations and we consider various values of δ and subdomains. We also compare against the Poisson case and the non-Poisson case.

Overall, we see that RAS performs better, in general, for the $b_1 = 10, b_2 = 20$ case. For 16 subdomains, RAS saves a few iterations over AS. For 32 subdomains, with $\delta = 1$, RAS performs a bit worse than AS. This may be because of we only consider 1-level overlap but have a high number of subdomains. For the Poisson case, RAS performed worse than AS in all tests, although it seemed to perform closest to AS for $\delta = 2$. Interestingly, the Poisson case had higher iterations over the non-Poisson case. The coefficient matrix $I + mA$ in the Poisson case becomes symmetric, so this may explain this behavior. However, we note that without any preconditioning, GMRES took 102 iterations, significantly higher than our results. While our results don't quite match up with the paper, we were still able to show that RAS generally performed as well, or better, than AS.

We also compared the harmonic extensions of AS and RAS (ASH and RASH, respectively) when used with GMRES. Table 2 summarizes the results. We use the same parameters as in the evaluation of AS and RAS. Here, we see that RASH performs significantly worse than ASH. This is actually expected behavior; the paper mentions in Remark 2.5 that RASH is always weaker than ASH. Indeed, we see that is the case here. We also note that

ASH performs similar to AS and is in fact better in some cases.

	$subd = 16$				$subd = 32$			
	$b_1 = b_2 = 0$		$b_1 = 10, b_2 = 20$		$b_1 = b_2 = 0$		$b_1 = 10, b_2 = 20$	
δ	ASH	RASH	ASH	RASH	ASH	RASH	ASH	RASH
1	31	63	20	33	59	80	31	54
2	27	64	16	33	31	84	20	52
3	25	64	14	32	27	86	16	50

Table 2: Iteration counts for GMRES with the ASH and RASH preconditioners.

4 Case Study 2

Our second case study is for a two-dimensional Helmhotlz with Sommerfield boundary conditions under the unit square. Our equation is defined as such:

$$\begin{aligned} -\Delta u - k^2 u &= f \quad in\ \Omega \\ \frac{\partial u}{\partial n} - iku &= 0 \quad on\ \partial\Omega \end{aligned}$$

4.1 Variation Formulation

For some basis function ϕ_n, where $u = \sum_m \xi_m \phi_m$

$$
\begin{aligned}
-(\Delta u, \phi_n) - k^2(\Delta, \phi_n) &= (f, \phi_n) \\
(\nabla u, \nabla \phi_n) - ik(u, \phi_n)_\Gamma - k^2(u, \phi_n) &= (f, \phi_n) \\
\sum_m \xi_m \Big[(\nabla \phi_m, \nabla \phi_n) - k^2(\phi_m, \phi_n) - ik(\phi_m, \phi_n)_\Gamma \Big] &= (f, \phi_n)
\end{aligned}
$$

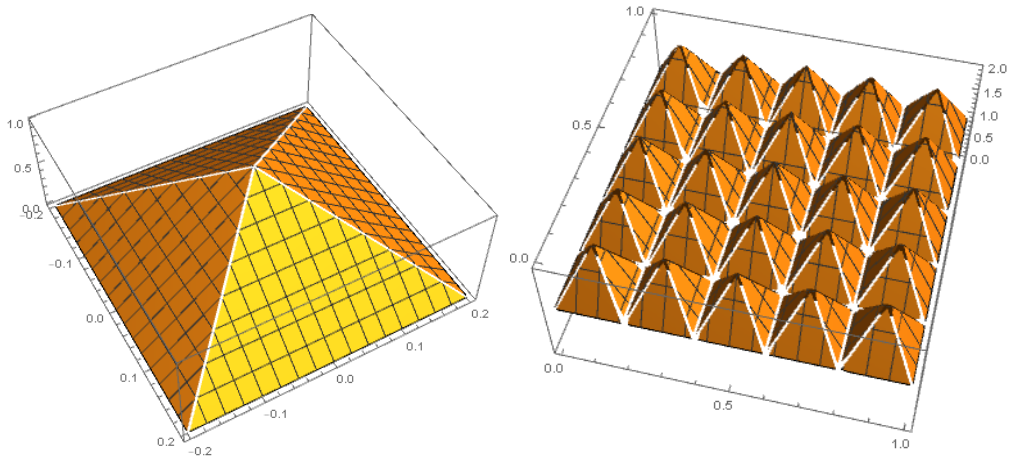
$$
\begin{aligned}
(\phi_m, \phi_n)_\Gamma = &\int_0^1 \phi_m(x,0)\phi_n(x,0)dx \\
&+ \int_0^1 \phi_m(1,y)\phi_n(1,y)dy \\
&+ \int_1^0 \phi_m(x,1)\phi_n(x,1)dx \\
&+ \int_1^0 \phi_m(0,y)\phi_n(0,y)dy
\end{aligned}
$$

Using Green's Theorem with our boundary condition, the surface integral becomes a sum of products of basis functions.

4.2 Basis Construction

Our initial basis attempt was over square shaped cells for a pyramid basis. This basis was insufficient, as they sum of basis functions had additional overlap.

$$
phi_{i,j}^{pyramid} = \begin{cases}
0 & \left|\frac{x-x_i}{h}\right| > 1 \vee \left|\frac{y-y_i}{h}\right| > 1 \\
1 - \frac{x-x_i}{h} & \left|\frac{x-x_i}{h}\right| > \left|\frac{y-y_i}{h}\right| \wedge \frac{x-x_i}{h} > 0 \\
\frac{x-x_i}{h} + 1 & \left|\frac{x-x_i}{h}\right| > \left|\frac{y-y_i}{h}\right| \wedge \frac{x-x_i}{h} \leq 0 \\
1 - \frac{y-y_i}{h} & \left|\frac{x-x_i}{h}\right| \leq \left|\frac{y-y_i}{h}\right| \wedge \frac{y-y_i}{h} > 0 \\
\frac{y-y_i}{h} + 1 & \left|\frac{x-x_i}{h}\right| \leq \left|\frac{y-y_i}{h}\right| \wedge \frac{y-y_i}{h} \leq 0
\end{cases}
$$

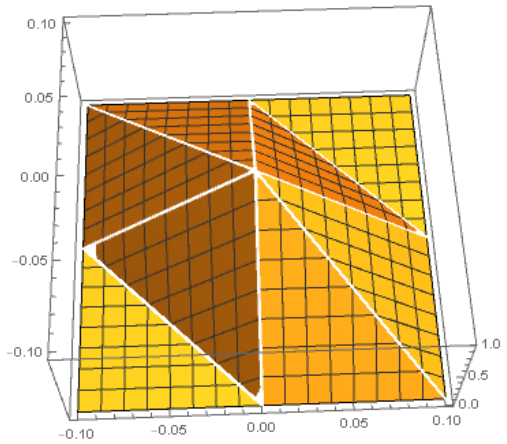


(a) $h = 1/5$, The pyramid basis function.

(b) $h = \frac{1}{5}$, The sum of basis functions with additional overlap.

Our second and successful attempt was to change the square shaped cells into diagonal shaped cells, and transform the basis into a 6 sized pyramid. While this increased the complexity of the integrals, the shape sufficed as a proper basis function.

$$\phi_{i,j}(x,y) = \begin{cases} \frac{h-x+x_i-y+y_i}{h} & x_i \leq x \wedge y_i \leq y \wedge (x+y) \leq (h+x_i+y_i) \\ \frac{h+x-x_i}{h} & x_i \leq (h+x) \wedge y_i \leq y \wedge (x+y) \leq (x_i+y_i) \\ \frac{h-y+y_i}{h} & x \leq x_i \wedge y \leq (h+y_i) \wedge (x_i+y_i) \leq (x+y) \\ \frac{h+x-x_i+y-y_i}{h} & x \leq x_i \wedge y \leq y_i \wedge (x_i+y_i) \leq (h+x+y) \\ \frac{h-x+x_i}{h} & x \leq (h+x_i) \wedge y \leq y_i \wedge (x_i+y_i) \leq (x+y) \\ \frac{h+y-y_i}{h} & x_i \leq x \wedge y_i \leq (h+y) \wedge (x+y) \leq (x_i+y_i) \end{cases}$$

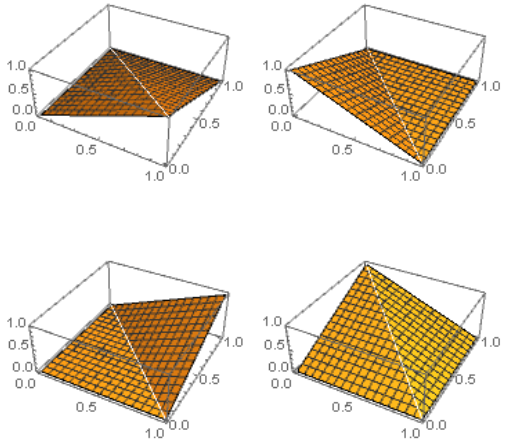


(a) $h = 1/10$, The 6 sided pyramid with triangular cells.

4.3 Matrix Calculation

With the Variation Formulation, there would need to be 2 double integrals and 4 surface integrals of 9 possible overlaps and 9 boundary conditions of a product of a piece-wise function with 6 parts. We decided based on the complexity of function and number of integrals to turn to Mathematica to solve it. But the sheer number of cases meant we needed to simplify the problem.

The first step we took was to calculate the integral by the sum of products of individual quadrants of the basis. The basis was separated into four quadrants, the integral of the products were calculated to a 4×4 matrix, and we could reference the sums for each overlap case.



(a) $h = 1$, The quadrants of ϕ used to calculate the integral.

$$(Quad(\phi), Quad(\phi)) = \begin{bmatrix} \frac{h^2}{12} & \frac{h^2}{24} & 0 & \frac{h^2}{24} \\ \frac{h^2}{24} & \frac{h^2}{6} & \frac{h^2}{24} & \frac{h^2}{12} \\ 0 & \frac{h^2}{24} & \frac{h^2}{12} & \frac{h^2}{24} \\ \frac{h^2}{24} & \frac{h^2}{12} & \frac{h^2}{24} & \frac{h^2}{6} \end{bmatrix}$$

$$(Quad(\phi'), Quad(\phi')) = \begin{bmatrix} 1 & -\frac{1}{2} & 0 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} & 0 \\ 0 & -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 0 & -\frac{1}{2} & 1 \end{bmatrix}$$

$$(Quad(\phi), Quad(\phi))_\Gamma = \begin{bmatrix} \frac{h}{3} & \frac{h}{6} & 0 & 0 \\ \frac{h}{6} & \frac{h}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \frac{h}{3} & \frac{h}{6} & 0 \\ 0 & \frac{h}{6} & \frac{h}{3} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$
$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{h}{3} & -\frac{h}{6} \\ 0 & 0 & -\frac{h}{6} & -\frac{h}{3} \end{bmatrix}, \begin{bmatrix} -\frac{h}{3} & 0 & 0 & -\frac{h}{6} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{h}{6} & 0 & 0 & -\frac{h}{3} \end{bmatrix}$$

Notice the symmetry over the diagonal. Let $f_{i,j}$ be some function with an inner product and can be broken up into quadrants, and $[x, y] = (Quad(f), Quad(f))_{x,y}$

coordinates of the inner product quadrant matrix. The possible overlaps are as follows.

$$
\begin{aligned}
(f_{i,j}, f_{i,j}) &= [1,1] + [2,2] + [3,3] + [4,4] \\
(f_{i,j}, f_{i+1,j}) &= (f_{i,j}, f_{i-1,j}) = [1,2] + [3,4] \\
(f_{i,j}, f_{i,j+1}) &= (f_{i,j}, f_{i,j-1}) = [1,4] + [2,3] \\
(f_{i,j}, f_{i+1,j-1}) &= (f_{i,j}, f_{i-1,j+1}) = [2,4] \\
(f_{i,j}, f_{i+1,j+1}) &= (f_{i,j}, f_{i-1,j-1}) = [1,3]
\end{aligned}
$$

For the edge cases on the side and full overlaps of phi, ϕ', we notice some symmetries that simplify the evaluation. Instead of evaluating different sums for each case, we can divide the inner product function by 2. For the corner cases of the full overlap case, we have two different divisions based on which diagonal.

$$
\begin{aligned}
[1,1] + [2,2] &= [3,3] + [4,4] \\
[1,1] + [4,4] &= [2,2] + [3,3] \\
[1,2] &= [3,4], \quad [2,3] = [1,4] \\
[1,1] = [3,3] &= \frac{(f_{i,j}, f_{i,j})}{6} \\
[2,2] = [4,4] &= \frac{(f_{i,j}, f_{i,j})}{3}
\end{aligned}
$$

5 Conclusion

Overall, our results indicate that RAS performs better than AS in some cases. While this does not necessarily agree with the results of the paper that indicated that RAS performs better than AS in both case studies, we do note that the paper employed a parallel approach for their preconditioners. This may explain the difference in results. Future work should consider implementing AS and RAS with parallel techniques or libraries such as OpenMP or MPI.

Regardless, we have shown that preconditioners play an important role in numerical linear algebra, especially in the systems that arise in solving PDEs. Their capability and relation to domain decomposition methods allow

us to both reduce the rate of convergence due to preconditioning *and* exploit parallelism for faster execution. Notably, AS and RAS are only two classes of preconditioners and many others exist, each with their own pros and cons. We plan to examine other preconditioners in the future and their applications to other numerical methods.

5.1 Distribution of Work

- **Justin**. Case study 1 and its implementation
- **Corey**. Preconditioner theory and implementations
- **Kurtis**. Case study 2 and its implementation

Otherwise, we all worked together when problems in our individual assignments arose.

References

[1] Andrea Toselli and Olof Widlund. *Domain decomposition methods-algorithms and theory*, volume 34. Springer Science & Business Media, 2006.

[2] Xiao-Chuan Cai and Yousef Saad. Overlapping domain decomposition algorithms for general sparse matrices. *Numerical linear algebra with applications*, 3(3):221–237, 1996.

[3] Xiao-Chuan Cai and Marcus Sarkis. A restricted additive schwarz preconditioner for general sparse linear systems. *SIAM journal on scientific computing*, 21(2):792–797, 1999.

[4] Xiao-Chuan Cai, Charbel Farhat, and Marcus Sarkis. A minimum overlap restricted additive schwarz preconditioner and applications in 3d flow simulations. *Contemporary Mathematics*, 218:482, 1998.

[5] Daniel Bouche, G Bonnaud, and D Ramos. Comparison of numerical schemes for solving the advection equation. *Applied mathematics letters*, 16(2):147–154, 2003.