Domain Decomposition Method of an Elliptic Model Problem

by

Hyun Lim

Contents

1 Introduction **1**

1.1 An Overview 1

1.2 Iterative Methods 1

1.2.1 Direct Methods 1

1.2.2 Stationary Iterative Methods 2

1.2.3 Krylov Subspace Methods 3

1.3 Preconditioning 4

2 Model Problem Derivation **6**

2.1 The Elliptic Problem 6

2.2 Finite Element Method 9

2.2.1 An Introduction 9

2.2.2 Developing the Discrete Problem 10

2.3 Developing the Stiffness Matrix on the Unit Square 16

2.3.1 Element Stiffness Matrix 18

2.3.2 Dirichlet Stiffness Matrix 23

2.3.3 Neumann Stiffness Matrix 29

3 Domain Decomposition Techniques **38**

3.1 An Introduction 38

3.2 One-Level Methods 39

3.2.1 Classical Alternating Schwarz Method 39

3.2.2 Multiplicative Schwarz Method 40
3.2.3 Additive Schwarz Method 42
3.3 Multi-Level Methods 42

4 Solving the Model Problem **43**
4.1 Introduction 43
4.2 General Programming Overview 44
4.3 One-Time Costs 45
4.3.1 Stiffness Matrix 45
4.3.2 Indexing 46
4.3.3 Splitting 57
4.3.4 Local Solver Set-up 58
4.4 Iterative Method 58
4.5 Numerical Results 58

List of Figures

2.1 Triangulation of a Domain 13
2.2 Uniform Triangulation of the Unit Square 16
2.3 Uniform Triangular Element 16
2.4 General Node Numbering Scheme 17
2.5 Single Triangular Element 18
2.6 Plane for Basis Function A 19
2.7 Plane for Basis Function B 20
2.8 Plane for Basis Function C 20
2.9 Node Numbering Scheme for the Dirichlet Matrix 23
2.10 Dirichlet Node 1 with Surrounding Elements 24
2.11 Relation of t_1 to T_j 24
2.12 Relation of t_2 to T_j 25
2.13 Relation of t_3 to T_j 25
2.14 Relation of t_4 to T_j 25
2.15 Relation of t_5 to T_j 26
2.16 Relation of t_6 to T_j 26
2.17 Elements Sharing Dirichlet Nodes 1 and 2 27
2.18 Elements Sharing Dirichlet Nodes 1 and $W-1$ 28
2.19 Node Numbering Scheme for the Neumann Matrix 30
2.20 Neumann Interior Node with Surrounding Elements 31
2.21 Relation of Neumann Corner Node to T_j 31
2.22 Relation of Neumann Corner Element 1 to Corner Element W 32

2.23 Relation of Neumann Corner Element 1 to Corner Element $(W-1)W+1$ 32
2.24 Relation of Neumann Corner Element 1 to Corner Element W^2 33
2.25 Neumann Edge Boundary Node with Surrounding Elements . 33
2.26 Relation of Neumann t_1 to T_j 34
2.27 Relation of Neumann t_2 to T_j 34
2.28 Relation of Neumann t_2 to T_j 34

4.1 General Node Numbering Scheme 46
4.2 Partitioning of Domain Ω 47
4.3 Local Node Numbering of a Partitioned Subdomain 47
4.4 Global Node Numbering of the $(1,1)$ Subdomain 48
4.5 Global Node Numbering of the $(1,2)$ Subdomain 49
4.6 Global Node Numbering of the $(2,1)$ Subdomain 50
4.7 Overlapping Division of Domain Ω 51
4.8 Local Node Numbering of a Corner Subdomain 53
4.9 Local Node Numbering of a Upper/Lower Boundary Subdomain 54
4.10 Local Node Numbering of a Side Boundary Subdomain 54
4.11 Local Node Numbering of an Interior Subdomain 55
4.12 Local Node Numbering of a General Overlapped Subdomain . 55
4.13 Global Node Numbering of a General Overlapped Subdomain 57

Chapter 1

Introduction

1.1 An Overview

1.2 Iterative Methods

1.2.1 Direct Methods

Consider a basic model problem of Poisson's equation on a square. By discretizing this elliptic partial differential equation over the domain using a finite difference or a finite element method, we form the linear system of equations $Ax = b$, where A is the $n \times n$ stiffness matrix.

Over time, many methods have been developed to solve such a system. The first methods were *direct methods*, which give exact solutions after a finite number of steps if exact arithmetic is used. The most common direct methods are the well-known Gaussian elimination and triangular factorization methods such as LU factorization. Although these methods are valuable in certain situations, if n is large, the $O(n^3)$ operations required to solve the system directly becomes very costly [8]. This high operation count,

combined with the relative inability of direct methods to take advantage of highly structured (i.e. sparse) stiffness matrices, makes solving the elliptic model problem with a direct method expensive and impractical.

1.2.2 Stationary Iterative Methods

For solving large systems, an *iterative method* is often used. Such methods differ from direct methods in that an exact solution is not being found. Instead, iterative methods begin with an arbitrary starting solution vector x^0 and then generate a sequence $\{x^m\}$ of iterates, or approximate solutions. The goal of any iterative method is to decrease the error after each step so that the sequence $\{x^m\}$ converges toward the actual solution x as quickly as possible using as little computational cost as possible [4]. Once the error $\|x - x^m\|$ becomes less than the stopping criteria provided by the user, the iterations stop and the final iterate is taken to be the solution.

Two of the most basic and common iterative methods are the Jacobi iteration and the Gauss-Seidel iteration. By decomposing the stiffness matrix A into the sum $A = D - L - U$ where D is a diagonal matrix, L is a strictly lower triangular matrix, and U is a strictly upper triangular matrix, these two methods can be defined in terms of matrix operators. The system $Ax = b$ can now be written as $(D - L - U)x = b$, and through matrix algebra, we develop the definitions of the iterative methods

$$x^{m+1} = D^{-1}\left(b + (L + U)x^m\right) \qquad \text{(Jacobi)}$$

and

$$x^{m+1} = (D - L)^{-1}\left(b + Ux^m\right) \quad [9]. \qquad \text{(Gauss-Seidel)}$$

Based on these definitions, both the Jacobi and Gauss-Seidel iterations can be expressed in the form

$$x^{m+1} = Cx^m + f,$$

where C and f do not depend on the iteration count m [1]. This implies that both methods are *stationary methods*, or methods for which the operator remains unchanged throughout the iterative process. Convergence of these methods is conditional, as convergence is only guaranteed for all starting vectors x^0 when $\rho(C) < 1$, where $\rho(C)$ is the spectral radius of the operator C [5]. When these methods do converge, they typically require $O(n^2)$ operations [3].

1.2.3 Krylov Subspace Methods

In contrast to a stationary method, a *non-stationary method*, or a *Krylov subspace method*, is an iterative method whose operator is iteration-dependent. In other words, the basic computations of the method involve information that changes with each iteration. This typically makes non-stationary methods more difficult to understand and implement than stationary methods. However, the computational cost saved is often well worth the extra initial effort, since these methods only require $O\left(n^{3/2}\right)$ operations [3].

Krylov subspace methods do not require direct access to the entries of the stiffness matrix A. Instead, these methods try to solve the system $Ax = b$ by iteratively utilizing a subroutine that performs matrix-vector multiplication involving A. At a given iteration m, Krylov subspace methods find approximate solutions in the Krylov subspace $\mathcal{K}_m$, which is dependent on the stiffness matrix A and the right-hand side vector b and is defined as

$$\mathcal{K}_m(A, b) \equiv \text{span}\{b, Ab, A^2b, \ldots, A^{m-1}b\} \quad [3].$$

For the elliptic model problem where the stiffness matrix A is symmetric and positive definite, the most common Krylov method used is the conjugate gradient method (CG). If the residual at iteration m is given as $r^m = b - Ax^m$, then CG works to find the solution x^m that minimizes the norm

$$\|r^m\| = \left(r^T A^{-1} r\right)^{1/2} \quad [3].$$

While CG is often highly effective, its convergence rate depends greatly on the condition number of the stiffness matrix A. If the condition number of A is defined as

$$\kappa(A) = \frac{\lambda_{max}(A)}{\lambda_{min}(A)},$$

then the bound for the error norm at iteration m is

$$\|x - x^m\| \leq 2\left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1}\right)^m \|x - x_0\| \quad [4].$$

This bound implies that as $\kappa(A) \to \infty$, the number of iterations expected before convergence of the CG method is $O\left(\sqrt{\kappa(A)}\right)$. Therefore, CG performs best when the stiffness matrix is well-conditioned. However, the discretization of the model problem often results in systems that are ill-conditioned.

1.3 Preconditioning

In these types of systems, a preconditioner M can be introduced to create the altered system $M^{-1}Ax = M^{-1}b$. For the preconditioner to be effective, M must be chosen so that

1) the linear system $Mx = b$ can be solved easily

2) M can be applied efficiently to a given vector, often the residual r^m

3) the condition number $\kappa\left(M^{-1}A\right)$ is small [7].

The final condition is especially important, as choosing M carefully to make $\kappa\left(M^{-1}A\right) \ll \kappa(A)$ guarantees convergence of the preconditioned conjugate gradient method (PCG) in a small number of iterations. The PCG algorithm is as follows [5]:

Algorithm 1 Preconditioned Conjugate Gradient Algorithm

Choose x^0

$r^0 = b - Ax^0$

Solve $Mz^0 = r^0$

$p^0 = z^0$

for $m = 0, 1, \ldots$ **do**

$\alpha^m = -\left(z^m, r^m\right) / \left(p^m, Ap^m\right)$

$x^{m+1} = x^m - \alpha^m p^m$

$r^{m+1} = r^m + \alpha^m A p^m$

Test for convergence

Solve $Mz^{m+1} = r^{m+1}$

$\beta^m = \left(z^{m+1}, r^{m+1}\right) / \left(z^m, r^m\right)$

$p^{m+1} = z^{m+1} + \beta^m p^m$

end for

Chapter 2

Model Problem Derivation

2.1 The Elliptic Problem

sec:pde

In Section 3.2.1, we discussed how to solve the general linear partial differential equation $Lu = f$, where L was the PDE operator and f was the "forcing function".

Consider now the elliptic case of this general linear PDE. In this case, the operator L changes to the elliptic operator Δ, called the *Laplacian*. The Laplacian can be expressed as

$$\Delta = \nabla \cdot \nabla = \sum_{i}^{D} \frac{\partial^2}{\partial x_i^2}$$

where $\nabla\cdot$ is the divergence, ∇ is the gradient, $\mathbf{D}$ is the number of dimensions considered, and $\mathbf{i}$ indexes the particular dimensions. Then, the inhomogeneous linear PDE becomes *Poisson's Equation*, which can be given as

$$\Delta u = f.$$

If it were such that $f = 0$, or that the PDE was homogeneous, then the equation would be referred to as *Laplace's equation*.

In practical applications, we often consider Poisson's Equation on a given domain Ω with certain conditions on its boundary $\partial\Omega$. Boundary conditions vary with the physical phenomenon encountered, but the three most common types of conditions are the Dirichlet, Neumann, and Robin boundary conditions. Dirichlet conditions provide the solution u on the boundary, whereas Neumann conditions provide the normal derivative $\frac{\partial u}{\partial n}$. The Robin conditions somewhat represent a combination of the previous two types, as Robin conditions specify $\frac{\partial u}{\partial n} + au$, where a is some function of the spatial dimensions.

Combining Poisson's Equation with any one of the different boundary conditions creates a boundary value problem, and with a known function g, the three main boundary value problems can be written as follows:

Dirichlet

$$\begin{aligned} \Delta u &= f \qquad \text{in} \qquad \Omega \\ u &= g \qquad \text{on} \qquad \partial\Omega \end{aligned}$$

Neumann

$$\begin{aligned} \Delta u &= f \qquad \text{in} \qquad \Omega \\ \tfrac{\partial u}{\partial n} &= g \qquad \text{on} \qquad \partial\Omega \end{aligned}$$

Robin

$$\begin{aligned} \Delta u &= f \qquad \text{in} \qquad \Omega \\ \tfrac{\partial u}{\partial n} + au &= g \qquad \text{on} \qquad \partial\Omega. \end{aligned}$$

For simplicity, we will focus on Laplace's equation with the negative

Laplacian, -Δ, as the operator and with Dirichlet boundary conditions:

$$\begin{aligned} -\Delta u &= 0 \qquad \text{in} \qquad \Omega \\ u &= g \qquad \text{on} \qquad \partial\Omega. \end{aligned}$$

Solutions u to this problem are called *harmonic functions*, and the following theorem states the uniqueness of these solutions.

Uniqueness Theorem of the Dirichlet Problem

The linear elliptic PDE

$$\begin{aligned} -\Delta u &= 0 \qquad \text{in} \qquad \Omega \\ u &= g \qquad \text{on} \qquad \partial\Omega. \end{aligned}$$

is unique on the domain Ω.

Proof: Suppose both v and w solve the Dirichlet Problem shown above. To show uniqueness, we must show that $v \equiv w$ in Ω.

Define $z = v - w$. Then, $-\Delta z = -\Delta v - (-\Delta w) = 0 - 0 = 0$ on Ω and $z = v - w = g - g = 0$ on $\partial\Omega$.

The maximum principle states that the maximum and minimum values of the solution are found on $\partial\Omega$. Since $z = 0$ on $\partial\Omega$, the maximum principle directly implies that $z = 0$ on Ω or that $z = 0 = v - w$ on Ω. Thus, $v \equiv w$ on Ω.$_\diamond$

Solutions to the Dirichlet Problem can be stated as

$$u(x_0) = \iint_{\partial\Omega} u(x) \frac{\partial G(x, x_0)}{\partial n} dS$$

2.2 Finite Element Method

2.2.1 An Introduction

Partial differential equations can be difficult or somewhat impossible to solve directly by formula, so analysts look for ways to develop solutions numerically. As mentioned in Section 3.2.2, when solving differential equations numerically, we often discretize the given problem on the domain Ω. This discretization converts the original continuous problem, which has infinitely many degrees of freedom, into a discrete problem, which has finitely many unknowns. A computer can then be utilized to solve the discrete problem.

For example, consider a case where the interior temperature of a flat steel plate is unknown. The flat plate, which constitutes the domain Ω, can be considered as a two-dimensional rectangular region on the Cartesian coordinate system. Solving the continuous problem would require attempts to find a formula for the temperature over the entire region so that the temperature at any one of the infinite number of points in the region could be calculated directly. The finite problem would usually involve approximating the temperature at a finite number of locations, or *nodes*, in the region, and then using these values to approximate the temperature at any point on the plate.

Finite difference methods can be used to discretize and solve the differential equation. In these methods, the derivatives in the differential equation are replaced by difference quotients, which means that the differential equation itself is approximated. Since the difference quotients involve the values of the unknown at the nodes, introducing these difference quotients typically leads to explicit formulas for the approximate solution at the nodes.

Finite element methods (FEM) are another way to form and solve the discrete version of certain differential equations. Instead of finding a way to approximate the derivates, these methods require the reformulation of the differential equation to a variational problem. Then, the domain is divided

into a finite number of *elements*, and a piecewise *basis function* is applied for each node on the domain. The original problem is thus converted to a discrete problem for which the solution can be algorithmically approximated.

While finite difference methods are very useful and sometimes very easy to implement, the finite element methods' main advantage lies in their ability to handle complicated domain and boundary geometries. Simplicity of the domain cannot be guaranteed in practical applications, and for this reason, we chose to implement the finite element method in this research.

2.2.2 Developing the Discrete Problem

subsec:dis

In this section, we wish to apply FEM to the elliptic model problem, which is the following form of Poisson's Equation:

$$\begin{aligned} -\Delta u &= f \qquad \text{in} \qquad \Omega \\ u &= 0 \qquad \text{on} \qquad \partial\Omega. \end{aligned}$$

Consider the case where Ω is a two-dimensional, bounded domain with x_1 and x_2 serving as the dimensions. The operator takes the form

$$\Delta = \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2},$$

and if we define the function space

$V = \{v | v \in C^0(\Omega)$, v' is piecewise continuous and bounded on Ω, and $v(\partial\Omega) = 0\}$,

the variational formulation of the model problem can be stated as follows [6]:

Find $u \in V$ such that

$$a(u, v) = (f, v) \qquad \text{for all } v \in V$$

where

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega \left[\frac{\partial u}{\partial x_1}\frac{\partial v}{\partial x_1} + \frac{\partial u}{\partial x_2}\frac{\partial v}{\partial x_2} \right] dx,$$

and

$$(f, v) = \int_\Omega f v dx.$$

We will now show that solutions to the variational problem also solve the differential equation.

Equivalence of Variational Solutions

A continuous function $u \in V$ solves the variational problem

$$a(u,v) = (f,v) \qquad \text{for all } v \in V$$

where

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega \left[\frac{\partial u}{\partial x_1}\frac{\partial v}{\partial x_1} + \frac{\partial u}{\partial x_2}\frac{\partial v}{\partial x_2} \right] dx,$$

and

$$(f,v) = \int_\Omega f v dx$$

if and only if u solves the linear elliptic PDE

$$\begin{aligned} -\Delta u &= f \qquad \text{in} \qquad \Omega \\ u &= 0 \qquad \text{on} \qquad \partial\Omega. \end{aligned}$$

Proof: Suppose u solves the PDE, and consider any function v in V.

If we multiply both sides of $-\Delta u = f$ by v and then integrate over Ω, we have $-\int_\Omega \Delta u v dx = \int_\Omega f v dx$.

By definition, we know that $-\int_\Omega f v dx = (f,v)$, and by Green's First Identity, $-\int_\Omega \Delta u v dx = \int_\Omega \nabla u \cdot \nabla v dx - \int_{\partial\Omega} v \frac{\partial u}{\partial n} ds$. Then, $a(u,v) = \int_\Omega \nabla u \cdot \nabla v dx$, so we now have $-\int_\Omega \Delta u v dx = a(u,v) - \int_{\partial\Omega} v \frac{\partial u}{\partial n} ds$.

If we substitute the new representations for $-\int_\Omega \Delta u v dx$ and $\int_\Omega f v dx$ into

the original expression, we have $a(u,v) - \int_{\partial\Omega} v\frac{\partial u}{\partial n}ds = (f,v)$.

Since the definition of the space V requires $v = 0$ on $\partial\Omega$, the integral term vanishes, and we are left with $a(u,v) = (f,v)$. Therefore, u solves the variational problem.

Conversely, if u solves $a(u,v) = (f,v)$, then $\int_\Omega \nabla u \cdot \nabla v dx - \int_\Omega fv dx = 0$.
But, $\int_\Omega \nabla u \cdot \nabla v dx = \int_{\partial\Omega} v\frac{\partial u}{\partial n}ds - \int_\Omega v\Delta u dx$ by Green's First Identity,
so $\int_{\partial\Omega} v\frac{\partial u}{\partial n}ds - \int_\Omega v\Delta u dx - \int_\Omega fv dx = 0$.

Because $v = 0$ on $\partial\Omega$, the integral over the boundary goes away, and we are left with $-\int_\Omega v\,(\Delta u + f)\,dx = 0$ for all $v \in V$.

If $(\Delta u + f)$ is continuous, then we can say that $(\Delta u + f) = 0$, or that $-\Delta u = f$ on Ω.
Hence, u solves the PDE. $\diamond$

This theorem proves that in this case, solving the variational problem should give us a solution to the original PDE. However, the variational problem is still difficult to solve, so we need to find a finite subspace of V to use in the variational problem that will provide the finiteness required in numerics. To do this, we must divide the domain Ω into a finite number of elements. In two dimensions, these elements can be shaped like almost any polygon, but for simplicity, we will consider triangular elements. Dividing Ω into triangular elements creates the triangulation of Ω as shown in the following figure:

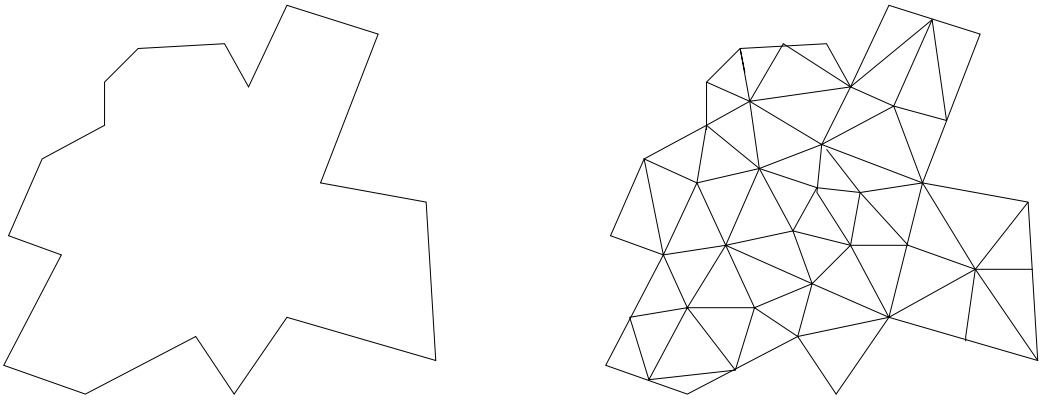

Figure 2.1: The left figure shows a generic domain Ω and the right figure shows an example triangulation on Ω.

In the triangulation, the vertices of the triangular elements represent the nodes in the domain. Therefore, we must require that no vertices lie on the edge of another triangle.

Assume that there are M triangles in the triangulation of Ω, where M is some natural number. If we label the triangles T_j for $j = 1, \ldots, M$, then we can see that $\Omega = \bigcup_{j=1}^{M} T_j$. We now define the finite subspace of V as

$$V_h = \{v | v \text{ is continuous on } \Omega, v|_{T_j} \text{ is linear for } T_j \in \Omega, \text{ and } v = 0 \text{ on } \partial\Omega\}.$$

In other words, V_h contains all continuous functions that are linear on each triangle of the triangulation of Ω.

Define the $N \in \aleph$ nodes on $\Omega \backslash \partial\Omega$ as N_i for $i = 1, \ldots, N$. The boundary nodes in this problem are known to have solution values of zero, so they need not be included.

What do functions $v \in V_h$ look like? If we create a basis function at each of the N nodes on the domain, we can represent v as a linear combination of these basis functions. Thus, define the basis functions ϕ_k for $k = 1, \ldots, N$

on Ω as

$$\phi_k(N_i) = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{if } i \neq k \end{cases} \quad \text{for } i = 1, \ldots, N.$$

This definition directly implies that ϕ_k only takes on non-zero values on the triangles T_j which share the node N_k. For all points $x \in \Omega$ that are not in one of those triangles, ϕ_k vanishes.

Now, functions $v \in V_h$ can be written as

$$v(x) = \sum_{k=1}^{N} \eta_k \phi_k(x) \quad \text{for } x \in \Omega$$

where $\eta_k = v(N_k)$ and Ω includes its boundaries.

By using the space V_h in the variational problem instead of V, we change the problem to finding $u_h \in V_h$ such that $a(u_h, v) = (f, v) \quad$ for all $v \in V_h$. This process is known as Galerkin's method.

Let us consider $a(u_h, v)$. Since $u_h \in V_h$, we can write it as

$$u_h(x) = \sum_{k=1}^{N} \eta_k \phi_k(x) \quad \text{with} \quad \eta_k = u_h(N_k).$$

Thus,

$$a(u_h, v) = \int_\Omega \nabla u_h \cdot \nabla v dx = \int_\Omega \nabla \left(\sum_{k=1}^{N} \eta_k \phi_k \right) \cdot \nabla v dx = \sum_{k=1}^{N} \eta_k \int_\Omega \nabla \phi_k \cdot \nabla v dx = \sum_{k=1}^{N} \eta_k \left[a(\phi_k, v) \right]$$

by the finiteness of the summation. Then, $a(u_h, v) = (f, v)$ becomes

$$\sum_{k=1}^{N} \eta_k \left[a(\phi_k, v) \right] = (f, v) \quad \text{for all} \quad v \in V_h.$$

But, $\phi_j \in V_h$ for some $j = 1, \ldots, N$, so we can write

$$\sum_{k=1}^{N} \eta_k \left[a(\phi_k, \phi_j) \right] = (f, v).$$

This is a linear system of equations with N unknowns: η_k for $k = 1, \ldots, N$.

If we define the vectors

$$\eta = \begin{pmatrix} \eta_1 \\ \vdots \\ \eta_N \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} (f, \phi_1) \\ \vdots \\ (f, \phi_N) \end{pmatrix}$$

and the matrix

$$A = \begin{pmatrix} a(\phi_1, phi_1) & \cdots & a(\phi_1, phi_N) \\ \vdots & \ddots & \vdots \\ a(\phi_N, phi_1) & \cdots & a(\phi_N, phi_N) \end{pmatrix},$$

then the linear system of equations can be written in matrix form as $A\eta = b$. Here, A is called the *stiffness matrix* and b is called the *force vector*. This is the system we wish to solve numerically.

Let us examine the stiffness matrix A. Since $a(\phi_i, \phi_j) = \int_\Omega \nabla\phi_i \cdot \nabla\phi_j dx = \int_\Omega \nabla\phi_j \cdot \nabla\phi_i dx = a(\phi_i, \phi_j)$, we can see that A is a symmetric matrix. By the definition of the basis functions, we can also see that many of the $a(\phi_i, \phi_j) = 0$, which means that A will be sparse. In addition, we can show that A is positive definite:

Proof: Pick any vector $z \neq 0 \in \Re^N$. We need to show that $z^T Az > 0$.

Because A is symmetric and positive definite, A is non-singular. This guarantees that a unique solution to the system $A\eta = b$ exists. In the next chapter, we consider multiple elliptic BVP's, develop the stiffness matrix A in each case using FEM, and then utilize the preconditioned conjugate gradient algorithm with domain decomposition as the preconditioner to solve the system.

2.3 Developing the Stiffness Matrix on the Unit Square

sec:stiff

Define $\Omega = [0,1] \times [0,1] \subset \Re^2$, or in other words, let Ω be the *unit square*. One possible triangulation of this domain, as shown in the following figure, is uniform:

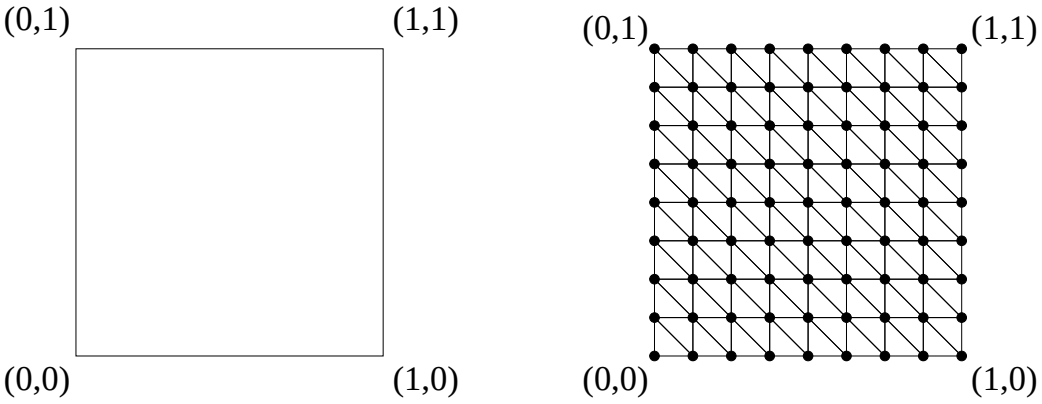

Figure 2.2: The left figure shows the unit square domain Ω and the right figure shows an example uniform triangulation on Ω.

In this case, uniform means that all triangles are of the same size, or that each right triangle can be defined to have height h equal to its base no matter which way the triangle is oriented.

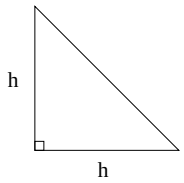

Figure 2.3: A standard uniform triangle with dimension h as both its base and height.

Note that this triangluation creates an evenly-spaced grid of nodes (indicated by dark circles in the previous figure.) If we create the triangulation such that each row and each column in the grid of nodes contains W nodes, then it is clear from the figure that $h = \frac{1}{W-1}$.

Remember that the stiffness matrix A is an $N \times N$ matrix, where N is the total number of nodes on the domain. In the unit square example we are considering, we have $N = W^2$, which makes choosing W a critical process. If W is too large, the system of equations becomes very large, and the numerical computation time will increase. If W is too small, then the approximation of the domain is coarser, and the numerical solution and interpolation process may become less accurate. Therefore, it may be beneficial to involve triangulations of multiple sizes when solving the problem on Ω.

In any case, we choose W nodes in a row and have $h = \frac{1}{W-1}$ as the triangle side length. We can number the nodes on the domain in the following fashion:

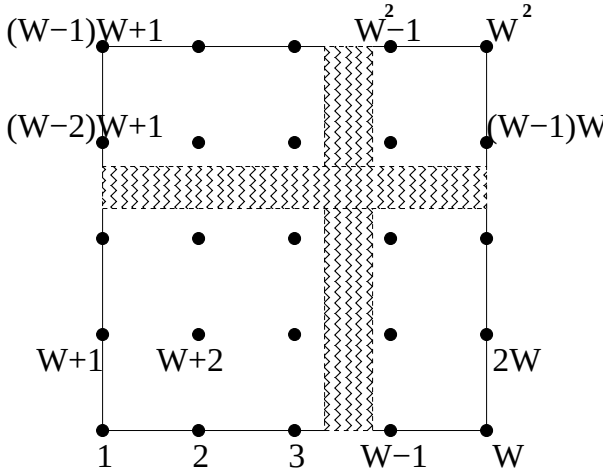

Figure 2.4: Node numbering for a domain with W nodes in each column and row.

We then look to find the stiffness matrix A, defined as

$$A = \begin{bmatrix} a(\phi_1, \phi_1) & \cdots & a(\phi_1, \phi_{W^2}) \\ \vdots & \ddots & \vdots \\ a(\phi_{W^2}, \phi_1) & \cdots & a(\phi_{W^2}, \phi_{W^2}) \end{bmatrix},$$

where ϕ_i represents the basis function at node i for $i = 1, \ldots, W^2$.

2.3.1 Element Stiffness Matrix

subsec:elem

To find A, we first consider a single triangular element

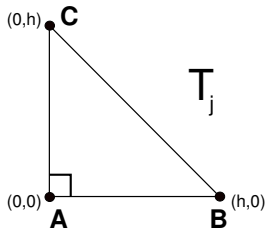

Figure 2.5: A single trianglular element T_j with nodes $A, B,$ and C.

The *element stiffness matrix* for this singular element is represented as

$$A_j = \begin{bmatrix} a_j(\phi_A, \phi_A) & a_j(\phi_A, \phi_B) & a_j(\phi_A, \phi_C) \\ a_j(\phi_B, \phi_A) & a_j(\phi_B, \phi_B) & a_j(\phi_B, \phi_C) \\ a_j(\phi_C, \phi_A) & a_j(\phi_C, \phi_B) & a_j(\phi_C, \phi_C) \end{bmatrix},$$

but since $a(\phi_D, \phi_F) = a(\phi_F, \phi_D)$ for any $D, F \in \{A, B, C\}$, we really only have to calculate the upper (or lower) triangular entries.

We know that $a(\phi_D, \phi_F) = \int_{T_j} \nabla\phi_d \cdot \nabla\phi_F dx$ for any $D, F \in \{A, B, C\}$, so we must determine the basis functions explicitly at the nodes:

$$\phi_A(D) = \begin{cases} 1 & \text{if } D = A \\ 0 & \text{if } D \neq A \end{cases} \quad \text{for } D \in \{A, B, C\}$$

$$\phi_B(D) = \begin{cases} 1 & \text{if } D = B \\ 0 & \text{if } D \neq B \end{cases} \quad \text{for } D \in \{A, B, C\}$$

$$\phi_C(D) = \begin{cases} 1 & \text{if } D = C \\ 0 & \text{if } D \neq C \end{cases} \quad \text{for } D \in \{A, B, C\}\,.$$

Consider first the basis function ϕ_A, shown in the following figure:

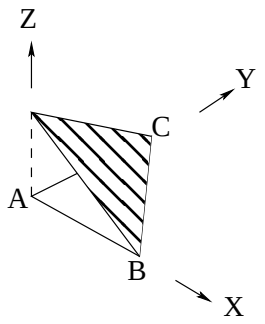

Figure 2.6: Plane formed on T_j by the basis function ϕ_A.

This creates a plane over the element region, which can be represented as $\phi_A = u_A x + v_A y + w_A$, where $u_A, v_A, w_A \in \Re$. If we note that $\phi_A = 1$ when $x = 0$ and $y = 0$, that $\phi_A = 0$ when $x = h$ and $y = 0$, and that $\phi_A = 0$ when $x = 0$ and $y = h$, we can develop a set of three equations for the three unknowns. Hence,

$$\begin{aligned} 1 &= u_A(0) + v_A(0) + w_A \implies w_A &= 1 \\ 0 &= u_A(h) + v_A(0) + w_A \implies u_A &= -\tfrac{1}{h} \\ 0 &= u_A(0) + v_A(h) + w_A \implies v_A &= -\tfrac{1}{h} \end{aligned}$$

and

$$\phi_A = -\frac{1}{h}x - \frac{1}{h}y + 1 \quad \text{for } (x, y) \in T_j.$$

Now, consider the basis function ϕ_B as shown below:

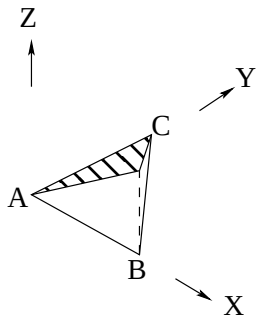

Figure 2.7: Plane formed on T_j by the basis function ϕ_B.

We again have a plane, so $\phi_B = u_B x + v_B y + w_B$. This time, $\phi_B = 1$ at $(h, 0)$ and $\phi_B = 0$ at $(0, 0)$ and $(0, h)$. Our system of equations to solve for $u_B, v_B,$ and w_B becomes

$$\begin{aligned} 0 &= u_B(0) + v_B(0) + w_B \\ 1 &= u_B(h) + v_B(0) + w_B \\ 0 &= u_B(0) + v_B(h) + w_B \end{aligned}$$

so that

$$\phi_B = \frac{1}{h}x \quad \text{for } (x, y) \in T_j.$$

Finally, consider ϕ_C:

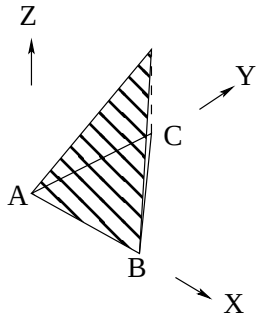

Figure 2.8: Plane formed on T_j by the basis function ϕ_C.

This plane is $\phi_C = u_C x + v_C y + w_C$, with $\phi_C = 1$ at $(0, h)$ and $\phi_C = 0$ at $(0, 0)$ and $(h, 0)$. By solving the system

$$\begin{aligned} 0 &= u_C(0) + v_C(0) + w_C \\ 0 &= u_C(h) + v_C(0) + w_C \\ 1 &= u_C(0) + v_C(h) + w_C, \end{aligned}$$

we find

$$\phi_C = \frac{1}{h} y \quad \text{for } (x, y) \in T_j.$$

Knowing the basis functions with respect to the (x, y) coordinates allows us to compute the components of the element stiffness matrix directly:

$$\begin{aligned} a_j(\phi_A, \phi_A) &= \int_{T_j} \nabla\phi_A \cdot \nabla\phi_A dx \quad = \int_0^h \int_0^{h-x} \nabla\phi_A \cdot \nabla\phi_A dy dx \\ &= \int_0^h \int_0^{h-x} \left[\tfrac{\partial\phi_A}{\partial x} \tfrac{\partial\phi_A}{\partial x} + \tfrac{\partial\phi_A}{\partial y} \tfrac{\partial\phi_A}{\partial y} \right] dy dx \\ &= \int_0^h \int_0^{h-x} \left[\left(-\tfrac{1}{h}\right) \left(-\tfrac{1}{h}\right) + \left(-\tfrac{1}{h}\right) \left(-\tfrac{1}{h}\right) \right] dy dx \\ &= \int_0^h \int_0^{h-x} \tfrac{2}{h^2} dy dx \quad = \int_0^h \left[\tfrac{2y}{h^2} \right]_0^{h-x} dx \\ &= \int_0^h \left[\tfrac{2(h-x)}{h^2} - 0 \right] dx \quad = \int_0^h \left[\tfrac{2}{h} - \tfrac{2x}{h^2} \right] dx \\ &= \left[\tfrac{2x}{h} - \tfrac{x^2}{h^2} \right]_0^h \quad = \left[\tfrac{2h}{h} - \tfrac{h^2}{h^2} \right] - [0 - 0] \\ &= 1 \end{aligned}$$

$$\begin{aligned} a_j(\phi_A, \phi_B) &= \int_{T_j} \nabla\phi_A \cdot \nabla\phi_B dx \quad = \int_0^h \int_0^{h-x} \nabla\phi_A \cdot \nabla\phi_B dy dx \\ &= \int_0^h \int_0^{h-x} \left[\tfrac{\partial\phi_A}{\partial x} \tfrac{\partial\phi_B}{\partial x} + \tfrac{\partial\phi_A}{\partial y} \tfrac{\partial\phi_B}{\partial y} \right] dy dx \\ &= \int_0^h \int_0^{h-x} \left[\left(-\tfrac{1}{h}\right) \left(\tfrac{1}{h}\right) + \left(-\tfrac{1}{h}\right) (0) \right] dy dx \\ &= \int_0^h \int_0^{h-x} -\tfrac{1}{h^2} dy dx \quad = \int_0^h \left[-\tfrac{y}{h^2} \right]_0^{h-x} dx \\ &= \int_0^h \left[-\tfrac{h-x}{h^2} + 0 \right] dx \quad = \int_0^h \left[-\tfrac{1}{h} + \tfrac{x}{h^2} \right] dx \\ &= \left[-\tfrac{x}{h} + \tfrac{x^2}{2h^2} \right]_0^h \quad = \left[-\tfrac{h}{h} + \tfrac{h^2}{2h^2} \right] - [0 - 0] \\ &= -\tfrac{1}{2} \end{aligned}$$

$$
\begin{aligned}
a_j\left(\phi_A, \phi_C\right) &= \int_{T_j} \nabla\phi_A \cdot \nabla\phi_C dx \quad = \int_0^h \int_0^{h-x} \nabla\phi_A \cdot \nabla\phi_C dy dx \\
&= \int_0^h \int_0^{h-x} \left[\tfrac{\partial\phi_A}{\partial x}\tfrac{\partial\phi_C}{\partial x} + \tfrac{\partial\phi_A}{\partial y}\tfrac{\partial\phi_C}{\partial y}\right] dy dx \\
&= \int_0^h \int_0^{h-x} \left[\left(-\tfrac{1}{h}\right)(0) + \left(-\tfrac{1}{h}\right)\left(\tfrac{1}{h}\right)\right] dy dx \\
&= \int_0^h \int_0^{h-x} \left(-\tfrac{1}{h^2}\right) dy dx \\
&= -\tfrac{1}{2}
\end{aligned}
$$

$$
\begin{aligned}
a_j\left(\phi_B, \phi_B\right) &= \int_{T_j} \nabla\phi_B \cdot \nabla\phi_B dx \quad = \int_0^h \int_0^{h-x} \nabla\phi_B \cdot \nabla\phi_B dy dx \\
&= \int_0^h \int_0^{h-x} \left[\tfrac{\partial\phi_B}{\partial x}\tfrac{\partial\phi_B}{\partial x} + \tfrac{\partial\phi_B}{\partial y}\tfrac{\partial\phi_B}{\partial y}\right] dy dx \\
&= \int_0^h \int_0^{h-x} \left[\left(\tfrac{1}{h}\right)\left(\tfrac{1}{h}\right) + (0)(0)\right] dy dx \\
&= \int_0^h \int_0^{h-x} \left(\tfrac{1}{h^2}\right) dy dx \\
&= \tfrac{1}{2}
\end{aligned}
$$

$$
\begin{aligned}
a_j\left(\phi_B, \phi_C\right) &= \int_{T_j} \nabla\phi_B \cdot \nabla\phi_C dx \quad = \int_0^h \int_0^{h-x} \nabla\phi_B \cdot \nabla\phi_C dy dx \\
&= \int_0^h \int_0^{h-x} \left[\tfrac{\partial\phi_B}{\partial x}\tfrac{\partial\phi_C}{\partial x} + \tfrac{\partial\phi_B}{\partial y}\tfrac{\partial\phi_C}{\partial y}\right] dy dx \\
&= \int_0^h \int_0^{h-x} \left[\left(\tfrac{1}{h}\right)(0) + (0)\left(\tfrac{1}{h}\right)\right] dy dx \\
&= \int_0^h \int_0^{h-x} (0) dy dx \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
a_j\left(\phi_C, \phi_C\right) &= \int_{T_j} \nabla\phi_C \cdot \nabla\phi_C dx \quad = \int_0^h \int_0^{h-x} \nabla\phi_C \cdot \nabla\phi_C dy dx \\
&= \int_0^h \int_0^{h-x} \left[\tfrac{\partial\phi_C}{\partial x}\tfrac{\partial\phi_C}{\partial x} + \tfrac{\partial\phi_C}{\partial y}\tfrac{\partial\phi_C}{\partial y}\right] dy dx \\
&= \int_0^h \int_0^{h-x} \left[(0)(0) + \left(\tfrac{1}{h}\right)\left(\tfrac{1}{h}\right)\right] dy dx \\
&= \int_0^h \int_0^{h-x} \left(\tfrac{1}{h^2}\right) dy dx \\
&= \tfrac{1}{2}.
\end{aligned}
$$

Using these calculated values, we can create the element stiffness matrix as

$$A_j = \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix}.$$

We now wish to use A_j to assemble the stiffness matrix for the entire domain. Because all of the elements in the triangulation of Ω are the same size, we can utilize A_j for all elements, only having to adjust for the orientation of the triangle.

subsec:dir

2.3.2 Dirichlet Stiffness Matrix

Let us first assemble A_D for the Dirichlet elliptic boundary value problem. For this BVP, the solution values at the nodes on the boundary $\partial\Omega$ are known, so these nodes need not be included in our system of equations. Hence, our stiffness matrix reduces from a $W^2 \times W^2$ matrix to a $(W-2)^2 \times (W-2)^2$ matrix, and the new node numbering is shown in the following figure:

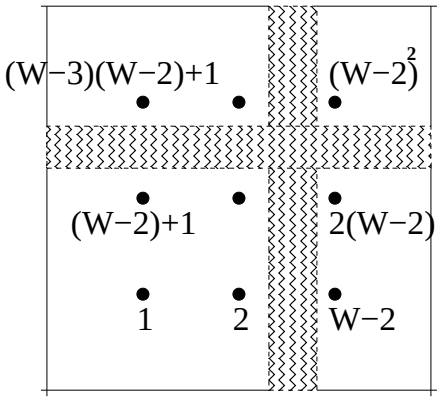

Figure 2.9: Node numbering used to form the Dirichlet stiffnes matrix for a domain with W nodes in each column and row.

If we label the stiffness matrix component in row i and column k as a_{ik}, then $a_{ik} = a(\phi_i, \phi_k)$, which can be calculated by adding the effects of all the triangular elements: $a_{ik} = \sum_{T_j \in \Omega} a_j(\phi_i, \phi_k)$. Those triangles not adjacent to nodes N_i and N_k have the property $a_j(\phi_i, \phi_k) = 0$, so essentially a_{ik} can be found by summing the effects of the elements that share nodes N_i and N_k.

As an example, we consider node 1, shown here with its adjacent triangular elements:

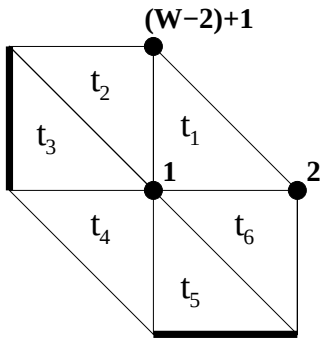

Figure 2.10: The elements surrounding node 1 with boundary edges bolded.

To find a_{11}, we must consider all elements adjacent to node 1.

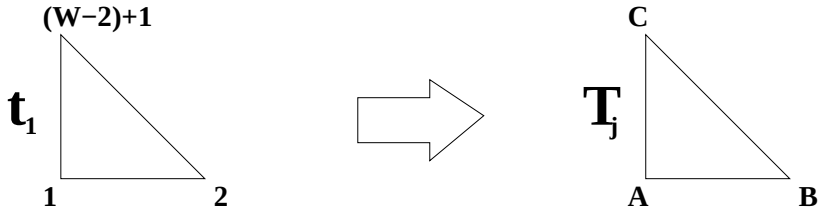

Figure 2.11: How element t_1 corresponds to element T_j from 2.3.1.

Triangle t_1 is oriented exactly the same as T_j when the element stiffness matrix was derived, so we can directly use A_j. From A_j, we see that $a_j(\phi_A, \phi_A) = 1$, so $a_{t_1}(\phi_1, \phi_1) = 1$.

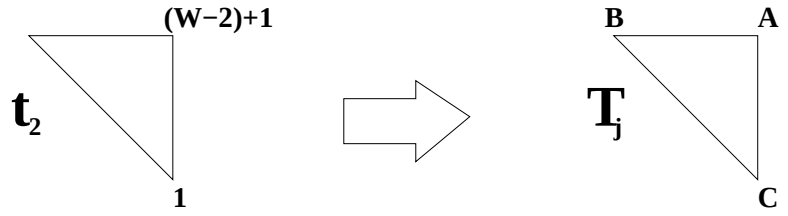

Figure 2.12: How element t_2 corresponds to element T_j.

In triangle t_2, node 1 corresponds to node C in T_j. This means that we use $a_j(\phi_C, \phi_C) = \frac{1}{2}$ to find $a_{t_2}(\phi_1, \phi_1) = \frac{1}{2}$.

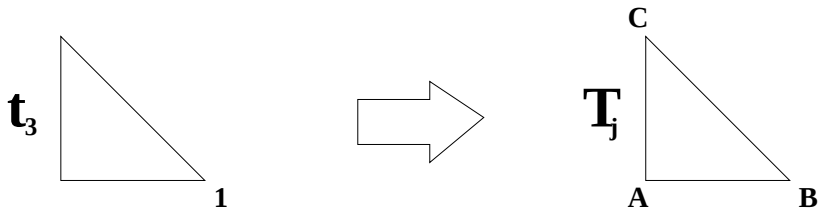

Figure 2.13: How element t_3 corresponds to element T_j.

We see that node 1 corresponds to node B in T_j for triangle t_3, and since $a_j(\phi_B, \phi_B) = \frac{1}{2}$, then $a_{t_3}(\phi_1, \phi_1) = \frac{1}{2}$.

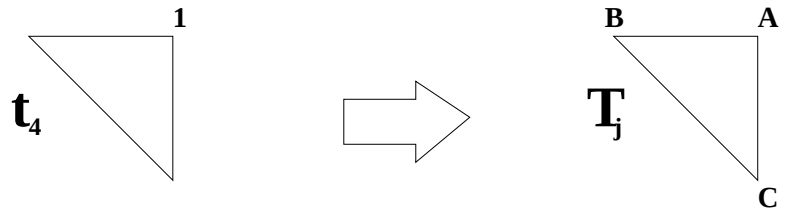

Figure 2.14: How element t_4 corresponds to element T_j.

For triangle t_4, node 1 corresponds to node A in T_j. Thus, like t_1, we have $a_{t_4}(\phi_1, \phi_1) = 1$.

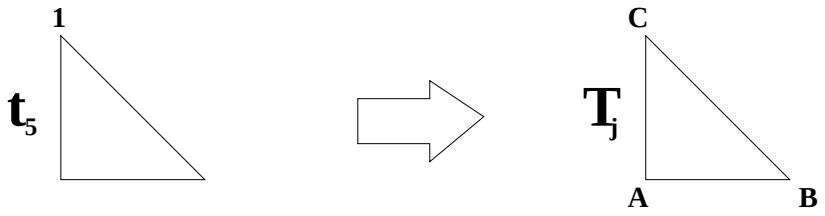

Figure 2.15: How element t_5 corresponds to element T_j.

Just like triangle t_2, node 1 corresponds to node C for triangle t_5. We then know that $a_{t_5}(\phi_1, \phi_1) = \frac{1}{2}$.

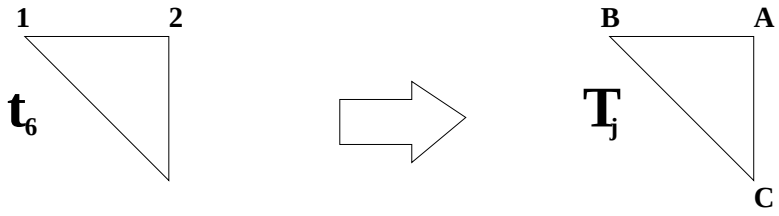

Figure 2.16: How element t_6 corresponds to element T_j.

Finally, triangle t_6 relates to T_j by having node 1 correspond to node B. This means that $a_{t_6}(\phi_1, \phi_1) = \frac{1}{2}$.

Therefore, we find a_{11} by summing the contributions from each of the six triangles adjacent to node 1. We have

$$a_{11} = \sum_{i=1}^{6} a_{t_i}(\phi_1, \phi_1) = 1 + \frac{1}{2} + \frac{1}{2} + 1 + \frac{1}{2} + \frac{1}{2} = 4.$$

Next, we look to calculate a_{12}, so we must consider all triangles that share nodes 1 and 2.

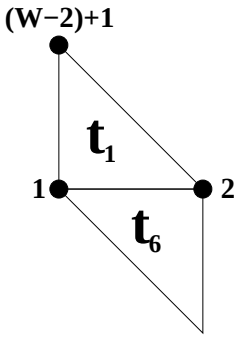

Figure 2.17: The elements which have vertices at both node 1 and node 2.

From our calculation of a_{11}, we know that in triangle t_1, node 1 corresponds to node A in T_j. Then, it is clear that node 2 corresponds to node B in T_j. Hence, $a_{t_1}(\phi_1, \phi_2) = a_j(\phi_A, \phi_B) = -\frac{1}{2}$.

Similarly, we know in triangle t_6 that node 1 corresponds to node B in T_j, and we find that node 2 corresponds to node A in T_j. We then calculate $a_{t_2}(\phi_1, \phi_2) = a_j(\phi_B, \phi_A) = -\frac{1}{2}$.

Therefore,

$$a_{12} = \sum_{i=1,6} a_{t_i}(\phi_1, \phi_2) = -\frac{1}{2} - \frac{1}{2} = -1.$$

The only remaining node in our Dirichlet system that shares elements with node 1 is node $(W-2)+1$, or $W-1$. To find $a_{1(W-1)}$, we look at the triangles sharing the two nodes.

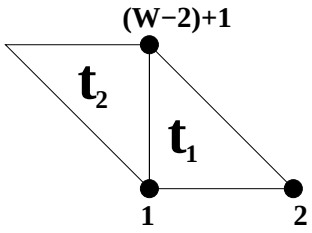

Figure 2.18: The elements which have vertices at both node 1 and node $W-1$.

In triangle t_1, nodes 1 and $W-1$ correspond to nodes A and C in T_j. In triangle t_2, nodes 1 and $W-1$ correspond to nodes C and A in T_j. Thus,

$$\begin{aligned} a_{t_1}(\phi_1, \phi_{W-1}) &= a_j(\phi_A, \phi_C) &= -\tfrac{1}{2} \\ \textit{and} \quad a_{t_2}(\phi_1, \phi_{W-1}) &= a_j(\phi_C, \phi_A) &= -\tfrac{1}{2}. \end{aligned}$$

Then,

$$a_{1(W-1)} = \sum_{i=1}^{2} a_{t_i}(\phi_1, \phi_{W-1}) = -\frac{1}{2} - \frac{1}{2} = -1.$$

The remaining components a_{1k} are all zeros since node 1 does not share elements with any of the remaining nodes in the domain.

If we would continue to complete this calculation process for the remaining nodes in the domain, we could determine all of the stiffness matrix components a_{ik} for $i, k = 1, \ldots, (W-2)^2$. We would find that $a_{ik} = 4$ for $i = k$ and $a_{ik} = -1$ for all $i \neq k$ when nodes N_i and N_k share common elements. This creates the block tri-diagonal stiffness matrix

$$A_D = \begin{bmatrix} D & -I_D & 0_D & \cdots & 0_D \\ -I_D & D & \ddots & \ddots & \vdots \\ 0_D & \ddots & \ddots & \ddots & 0_D \\ \vdots & \ddots & \ddots & D & -I_D \\ 0_D & \cdots & 0_D & -I_D & D \end{bmatrix}$$

where

$$D = \begin{bmatrix} 4 & -1 & 0 & \cdots & 0 \\ -1 & 4 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 4 & -1 \\ 0 & \cdots & 0 & -1 & 4 \end{bmatrix}$$

is $(W-2) \times (W-2)$, I_D is the $(W-2) \times (W-2)$ identity matrix, and 0_D is the $(W-2) \times (W-2)$ zero matrix.

Then, A_D serves as the $(W-2)^2 \times (W-2)^2$ stiffness matrix used in the system $A_D \eta_D = b_D$. We see that A_D is symmetric and sparse, just as we determined for the general stiffness matrix A in 2.2.2.

2.3.3 Neumann Stiffness Matrix

subsec:neu

Let us now assemble the stiffness matrix A_N for the Neumann elliptic boundary value problem. Since the solution values on $\partial\Omega$ are not known in this problem, we must include them in the system. Thus, we must revert back to the original node numbering

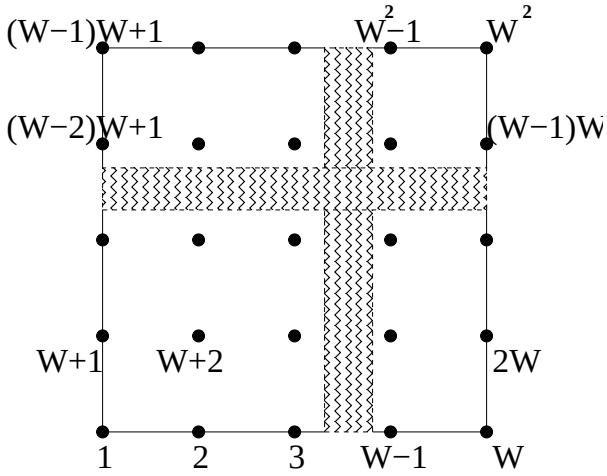

Figure 2.19: Node numbering used to form the Neumann stiffnes matrix for a domain with W nodes in each column and row.

This makes our stiffness matrix A_N have dimensions $W^2 \times W^2$.

Because the boundary nodes are included in A_N, the formulation of the matrix will be somewhat different than in the Dirichlet case. Instead of only having one type of node to consider, there are now three types: interior nodes, corner boundary nodes, and edge boundary nodes.

First, consider an interior node, namely N_i. Nodes of this type are adjacent to six triangular elements:

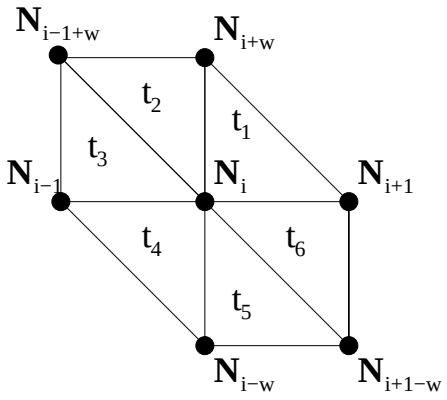

Figure 2.20: The elements surrounding arbitrary interior node N_i.

For these nodes, we have a situation similar to the nodes considered in the formulation of the Dirichlet stiffness matrix, A_D. Thus, we know from the previous derivation that $a_{ii} = 4$, $a_{i(i-W)} = -1$, $a_{i(i-1)} = -1$, and $a_{i(i+1)} = -1$, and that the remaining $a_{ik} = 0$.

Now, consider node 1:

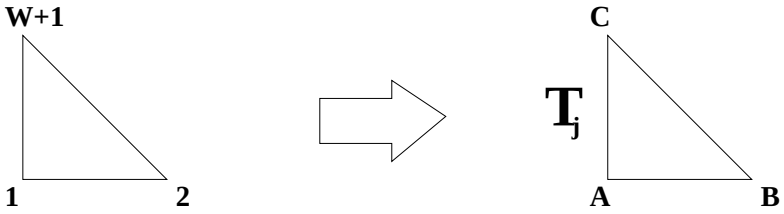

Figure 2.21: How a Neumann corner boundary element corresponds to element T_j.

It serves as an example of a corner boundary node. From the figure, it is apparent that the corner boundary nodes only border one element, and we can consider this element the same as the triangle T_j from 2.3.1. Therefore,

we can pull the a_{1k} values directly from the element stiffness matrix:

$$\begin{aligned} a_{11} = &\quad a_j(\phi_A, \phi_A) = \quad 1 \\ a_{12} = &\quad a_j(\phi_A, \phi_B) = \quad -\tfrac{1}{2} \\ a_{1(W+1)} = &\quad a_j(\phi_A, \phi_C) = \quad -\tfrac{1}{2}. \end{aligned}$$

These values hold for all corner elements, if we adjust for orientation:

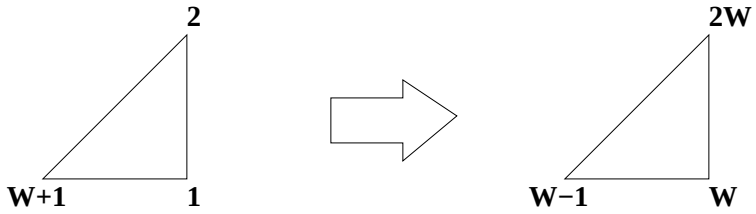

Figure 2.22: How Neumann corner boundary element 1 corresponds to element W.

$$\begin{aligned} a_{11} \Rightarrow &\quad a_{WW} &&= 1 \\ a_{12} \Rightarrow &\quad a_{W(2W)} &&= -\tfrac{1}{2} \\ a_{1(W+1)} \Rightarrow &\quad a_{W(W-1)} &&= -\tfrac{1}{2} \end{aligned}$$

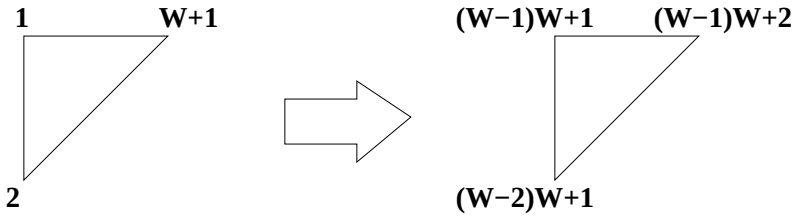

Figure 2.23: How Neumann corner boundary element 1 corresponds to element $(W-1)W+1$.

$$
\begin{aligned}
a_{11} \Rightarrow \quad & a_{[(W-1)(W+1)][(W-1)(W+1)]} &&= 1 \\
a_{12} \Rightarrow \quad & a_{[(W-1)(W+1)][(W-2)(W+1)]} &&= -\tfrac{1}{2} \\
a_{1(W+1)} \Rightarrow \quad & a_{[(W-1)(W+1)][(W-1)(W+2)])} &&= -\tfrac{1}{2}
\end{aligned}
$$

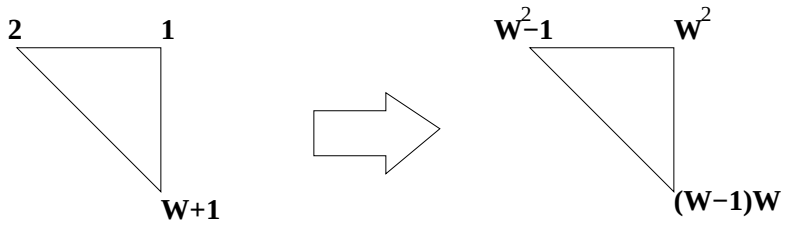

Figure 2.24: How Neumann corner boundary element 1 corresponds to element W^2.

$$
\begin{aligned}
a_{11} \Rightarrow \quad & a_{W^2W^2} &&= 1 \\
a_{12} \Rightarrow \quad & a_{W^2(W^2-1)} &&= -\tfrac{1}{2} \\
a_{1(W+1)} \Rightarrow \quad & a_{W^2[(W-1)W])} &&= -\tfrac{1}{2}
\end{aligned}
$$

Finally, we consider the edge boundary nodes, using node 2 as an example:

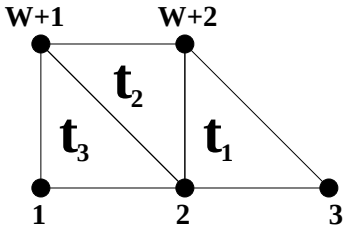

Figure 2.25: The elements surrounding arbitrary edge boundary node 2.

To find a_{22}, we need to consider the contribution of all three triangles.

For t_1, node 2 corresponds to node A in T_j:

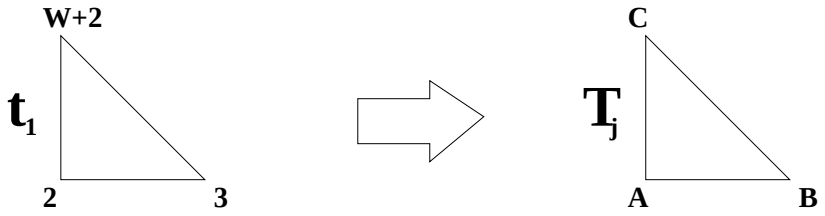

Figure 2.26: How Neumann element t_1 corresponds to element T_j from 2.3.1.

Thus, $a_{t_1}(\phi_2, \phi_2) = a_j(\phi_A, \phi_A) = 1$.
For t_2, node 2 corresponds to node C in T_j:

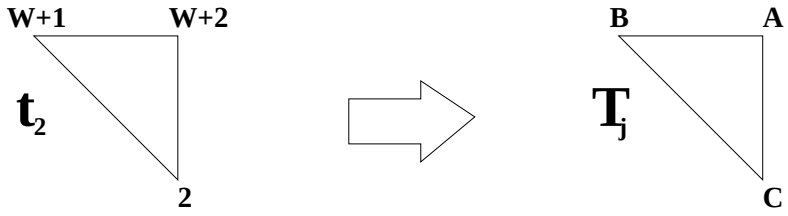

Figure 2.27: How Neumann element t_2 corresponds to element T_j.

and we find $a_{t_2}(\phi_2, \phi_2) = a_j(\phi_C, \phi_C) = \frac{1}{2}$.
For t_3, node 2 corresponds to node B in T_j:

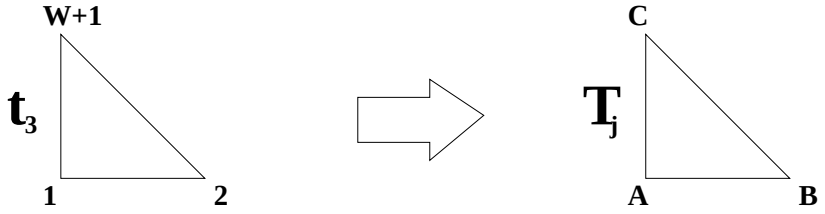

Figure 2.28: How Neumann element t_3 corresponds to element T_j.

We then calculate $a_{t_3}(\phi_2, \phi_2) = a_j(\phi_B, \phi_B) = \frac{1}{2}$.

Therefore,

$$a_{22} = \sum_{i=1}^{3} a_{t_i}(\phi_2, \phi_2) = 1 + \frac{1}{2} + \frac{1}{2} = 2.$$

Next, we want to find a_{21} and a_{23}. In each case, node 2 shares only one element with the respective nodes. We can see from the figure that in triangle t_3, node 2 corresponds to node B in T_j, so

$$a_{21} = a_{t_3}(\phi_2, \phi_1) = a_j(\phi_B, \phi_A) = -\frac{1}{2}.$$

Similarly, in triangle t_1, node 2 corresponds to node A in T_j while node 3 corresponds to node B. Hence,

$$a_{23} = a_{t_1}(\phi_2, \phi_3) = a_j(\phi_A, \phi_B) = -\frac{1}{2}.$$

To find $a_{2(W+1)}$, we must consider triangles t_2 and t_3. For t_2, we see that node 2 corresponds to node C and node $W+1$ corresponds to node B. This means that $a_{t_2}(\phi_2, \phi_{W+1}) = a_j(\phi_C, \phi_B) = 0$. For t_3, node 2 projects to node B of T_j and node W =! projects to node C. We then find that $a_{t_3}(\phi_2, \phi_{W+1}) = a_j(\phi_B, \phi_C) = 0$.

This means that

$$a_{2(W+1)} = \sum_{i=2}^{3} a_{t_i}(\phi_2, \phi_{W+1}) = 0 + 0 = 0.$$

The last node that shares elements with node 2 is node $W+2$, and we must determine the contribution from elements t_1 and t_2. By utilizing the correspondance shown in the figure, $a_{t_1}(\phi_2, \phi_{W+2}) = a_j(\phi_A, \phi_C) = -frac12$ and $a_{t_2}(\phi_2, \phi_{W+2}) = a_j(\phi_C, \phi_A) = -frac12$.

Thus,

$$a_{2(W+2)} = \sum_{i=1}^{2} a_{t_i}(\phi_2, \phi_{W+2}) = -\frac{1}{2} - \frac{1}{2} = -1.$$

Just like with the corner boundary nodes, the values calculated for the edge boundary nodes hold for all of the nodes of this type, as long as orientation of the elements is considered.

Applying all of these calculated values to the remaining nodes in the domain, we find the block tri-diagonal stiffness matrix

$$A_N = \begin{bmatrix} N_B & L & 0_N & \cdots & 0_N \\ L & N_I & \ddots & \ddots & \vdots \\ 0_N & \ddots & \ddots & \ddots & 0_N \\ \vdots & \ddots & \ddots & N_I & L \\ 0_N & \cdots & 0_N & L & N_B \end{bmatrix}$$

where

$$N_B = \begin{bmatrix} 1 & -\frac{1}{2} & 0 & \cdots & 0 \\ -\frac{1}{2} & 2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 2 & -\frac{1}{2} \\ 0 & \cdots & 0 & -\frac{1}{2} & 1 \end{bmatrix}$$

is $W \times W$,

$$N_I = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & & & & \vdots \\ 0 & & D & & 0 \\ \vdots & & & & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix}$$

is $W \times W$,

$$L = \begin{bmatrix} -\frac{1}{2} & 0 & \cdots & \cdots & 0 \\ 0 & & & & \vdots \\ \vdots & & -I_D & & \vdots \\ \vdots & & & & 0 \\ 0 & \cdots & \cdots & 0 & -\frac{1}{2} \end{bmatrix}$$

is $W \times W$, 0_N is the $W \times W$ zero matrix, and I_D and D come from 2.3.2.

Chapter 3

Domain Decomposition Techniques

3.1 An Introduction

As technology advances and parallel computing becomes more relevant, an increasing amount of research is being done to explore methods that more quickly solve difficult numerical problems by utilizing the benefits of parallel processing. Domain decomposition methods are one such type of method; they involve solving both linear and nonlinear systems that result from discretizing partial differential equations. Typically, domain decomposition methods function as pre-conditioners for some type of iterative method, most often a Krylov subspace method such as the conjugate gradient method or the generalized minimum residual method.

When used as a preconditioner, domain decomposition involves taking a very large system and dividing the domain Ω into numerous subdomains Ω_i for $i = 1, 2, \ldots$ The idea is to reduce the problem from one huge, cumbersome problem to multiple smaller problems that are more manageable and less

expensive to solve. Each of these smaller problems can be sent to its own processor so that all of them can be solved simultaneously, and the solutions on the subdomains can then be applied as a preconditioner for the discretized system of the partial differential equation on the entire domain.

3.2 One-Level Methods

3.2.1 Classical Alternating Schwarz Method

sec:alt

The first known domain decomposition method was developed by Hermann A. Schwarz and introduced in 1870 [2]. Initially, this method, now called the *classical alternating Schwarz method*, was not created as a numerical method; instead, it was used to prove that a solution to an elliptic boundary value problem existed on domains which were formed by the union of two simpler, overlapping subdomains [7]. Despite its original intentions, the alternating Schwarz method can be used to solve the elliptic boundary value problem on the entire domain.

Consider the domain Ω shown in Figure 1. It is clear that Ω consists of two overlapping subdomains, Ω_1 and Ω_2, so that $\Omega = \Omega_1 \cup \Omega_2$. The boundary of Ω is defined as $\partial\Omega$, and similarly, the boundaries of Ω_1 and Ω_2 are defined as $\partial\Omega_1$ and $\partial\Omega_2$, respectively. Let Γ_i represent the artificial boundaries, or, as shown in Figure 1, the boundary of Ω_i that is in the interior of Ω.

Assume that we wish to solve the linear partial differential equation (PDE)

$$\begin{aligned} Lu &= f \qquad \text{in} \qquad \Omega \\ u &= g \qquad \text{on} \qquad \partial\Omega, \end{aligned}$$

where L is the PDE operator, f is the "forcing function", and g depends on the given boundary conditions. In the elliptic case, L might represent the negative Laplacian, $-\Delta$, and g could be determined by Dirichlet boundary

conditions.

To solve this problem, the alternating Schwarz method could be employed by first choosing an initial guess vector u_2^0 for the solution values in Γ_1, and then iteratively solving the elliptic problem on the individual subdomains as follows [2]:

For u_1^n,

$$\begin{aligned} Lu_1^n &= f \qquad \text{in} \qquad \Omega_1 \\ u_1^n &= g \qquad \text{on} \qquad \partial\Omega_1 \setminus \Gamma_1 \\ u_1^n &= u_2^{n-1} \qquad \text{on} \qquad \Gamma_1, \end{aligned}$$

and then for u_2^n,

$$\begin{aligned} Lu_2^n &= f \qquad \text{in} \qquad \Omega_2 \\ u_2^n &= g \qquad \text{on} \qquad \partial\Omega_2 \setminus \Gamma_2 \\ u_2^n &= u_1^n \qquad \text{on} \qquad \Gamma_2. \end{aligned}$$

In other words, for each iteration, the elliptic boundary value problem is first solved on the subdomain Ω_1 using the boundary condition g on the non-artificial boundary portion of $\partial\Omega_1$ and the previous approximation of the solution on the artificial boundary Γ_1. Then, the problem is solved similarly on Ω_2.

It should be noted that this method can be extended to domains containing more than two overlapping subdomains by utilizing recursion. However, only the two subdomain case is shown for clarity.

3.2.2 Multiplicative Schwarz Method

sec:msm

Practical applications often require the discretization of the linear PDE on Ω. The discretization creates a grid of points on Ω, and in some cases, the grid points in Ω_1 will not match the grid points in Ω_2 in the overlap region. When the grids do not match between subdomains, the alternating Schwarz

method is applied as described before, except the operator L is replaced by its discrete form, A.

When the subdomains can be discretized so that the grids match in the overlap region, the alternating Schwarz method can be optimized. The resulting method is known as the *multiplicative Schwarz method*. Using A again as the discrete form of the operator L, we define the multiplicative Schwarz method by its fractional steps

$$u^{n+\frac{1}{2}} = u^n + \begin{pmatrix} A_{\Omega_1^{-1}} & 0 \\ 0 & 0 \end{pmatrix} (f - Au^n)$$

and

$$u^{n+1} = u^{n+\frac{1}{2}} + \begin{pmatrix} 0 & 0 \\ 0 & A_{\Omega_2^{-1}} \end{pmatrix} \left(f - Au^{n+\frac{1}{2}}\right) \quad [2].$$

Here, A_{Ω_i} represents the operator A restricted to Ω_i.

If the restriction matrix R_i is defined by $u_{\Omega_i} = R_i u$, we see that R_i restricts the solution vector u to just the solution values on the interior of Ω_i. Applying the restriction matrix to A, we find that $A_{\Omega_i} = R_i A R_i^{\mathrm{T}}$. Thus,

$$\begin{pmatrix} A_{\Omega_1^{-1}} & 0 \\ 0 & 0 \end{pmatrix} = R_1^{\mathrm{T}} \left(R_1 A R_1^{\mathrm{T}}\right) R_1 = B_1$$

and

$$\begin{pmatrix} 0 & 0 \\ 0 & A_{\Omega_2^{-1}} \end{pmatrix} = R_2^{\mathrm{T}} \left(R_2 A R_2^{\mathrm{T}}\right) R_2 = B_2$$

so that the multiplicative Schwarz method can be written as

$$u^{n+\frac{1}{2}} = u^n + B_1 (f - Au^n)$$

and

$$u^{n+1} = u^{n+\frac{1}{2}} + B_2 \left(f - Au^{n+\frac{1}{2}}\right) \quad [2].$$

3.2.3 Additive Schwarz Method

3.3 Multi-Level Methods

Chapter 4

Solving the Model Problem

4.1 Introduction

In this chapter, we present an approach to utilize a Domain Decomposition Method while finding a solution to the model problem derived in Chapter 2. We restrict ourselves to solving LaPlace equation with homogeneous boundary conditions on the unit square, but the application of the Domain Decomposition Method can be extended to problems on a general domain with general boundary conditions. We select this particular model problem for its lack of complexity in hopes that our methodology for applying the Domain Decomposition Method can be understood more clearly. Additionally, solving a problem for which we know the solution allows us to test the accuracy of our methods, since we can directly compare our numerical results to the known solution.

The underlying intent of this chapter is to 1) provide insight on how to program a Domain Decomposition Method and 2) stimulate ideas as to how these methods can be applied usefully when solving systems in parallel. Thus, we begin the chapter with a general overview of our programming

structure, presented pictorally on a flow chart. We then discuss in detail the introductory processes that are critical for setting up the iterative method and Domain Decomposition Method of choice. Finally, we demonstrate how to implement both the iterative and Domain Decomposition Methods.

4.2 General Programming Overview

The above diagram displays the overall programming structure for our method of solving LaPlace equation with homogeneous boundaries on the unit square. As discussed in Chapter 3, we use the preconditioned conjugate gradient method as our global solver for this problem, and we use the additive Schwarz method as the preconditioner to demonstrate the application of a Domain Decomposition Method.

The diagram is divided into two sections: one-time processes and the iterative method. Processes that appear in the first section are done as preparation for the iterative method. This means they are only required to be done once, and not during every iteration of the iterative method. By keeping these processes outside the loop of the iterative method, we have prevented a repetition of flops that could have become costly, especially for large systems.

The one-time processes are further sub-divided into two categories: problem set-up and Domain Decomposition Method set-up. Creating the stiffness matrix A (as discussed in 2.3) constitutes the problem set-up category. The creation of this matrix depends only on the definition of the problem, and A defines the system to be solved whether we use a Domain Decomposition Method or not. The Domain Decomposition Method set-up consists of relating the node indexes of the global and local domains, splitting the stiffness matrix on the subdomains, and completing any preparatory steps needed for the local solver. Each of these processes are done so that the additive Schwarz method can be implemented as a preconditioner for the conjugate

gradient method.

4.3 One-Time Costs

4.3.1 Stiffness Matrix

In section 2.3, we developed the structure of the stiffness matrices for both the Dirichlet elliptic boundary value problem and the Neumann elliptic boundary value problem on the unit square when the triangulation of the domain was uniform. Recall that solution values at the boundary nodes were known in the Dirichlet problem, so the Dirichlet stiffness matrix A_D had fewer degrees of freedom than the Neumann stiffness matrix A_N for the same triangulation. However, the size of both matrices depended on W, the number of nodes in each row and each column of the triangulation.

As we attempt to construct the stiffness matrices during programming, we note that both stiffness matrices can be written in block form, and that some of the blocks in A_D appear in A_N as well. Therefore, if we know the value of W, we can first create A_D block by block, and then create A_N block by block inside the same program, if desired.

For the Dirichlet stiffness matrix, we can initialize A_D to a $(W-2)^2 \times (W-2)^2$ matrix of all zeros. Then, we create the $(W-2) \times (W-2)$ blocks D and I_D following the structures discussed in section 2.3.2. Finally, we loop through the block positions on the diagonal, subdiagonal, and superdiagonal of A_D and assign the correct blocks into the correct locations.

After A_D is constructed, we can initialize A_N to a $W^2 \times W^2$ matrix of all zeros. Using blocks D and I_D already created, we then develop the $W \times W$ blocks N_B, N_I, and L as shown in section 2.3.3. We complete A_N by assinging the correct blocks into the correct locations on the diagonal, subdiagonal, and superdiagonal.

4.3.2 Indexing

ubsec:index

To apply a Domain Decomposition Method, we divide Ω into subdomains on which we intend to solve the model problem locally. Once the local solutions are found on the subdomains, they need to be combined to form the domain's global solution. Thus, we need to develop a way to relate the subdomain (local) node numbers to the domain (global) node numbers.

We recall our triangulation and global node numbering of the original domain Ω:

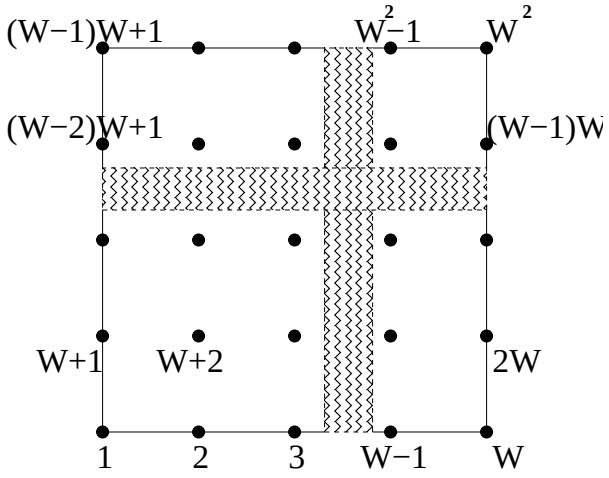

Figure 4.1: Node numbering for a domain with W nodes in each column and row.

Note that there are other choices for a node numbering scheme, and the particular scheme that is used controls how the global node numbers relate to the local node numbers. The following information defines the indexing procedure for the node numbering scheme we have selected, but it can be extended to other enumeration schemes.

Let us first partition Ω into a $P \times P$ grid of subdomains, where P is a divisior of $W - 1$:

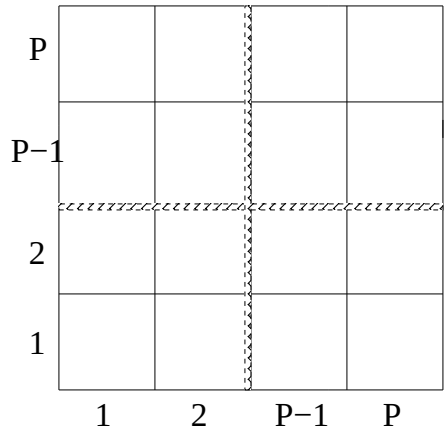

Figure 4.2: Partitioning of the unit square into a $P \times P$ grid of subdomains.

Then, each of the P^2 subdomains has a grid of $\left(\frac{W-1}{P}+1\right) \times \left(\frac{W-1}{P}+1\right)$ nodes, which can be numbered locally as follows when we let $Z = \frac{W-1}{P}+1$:

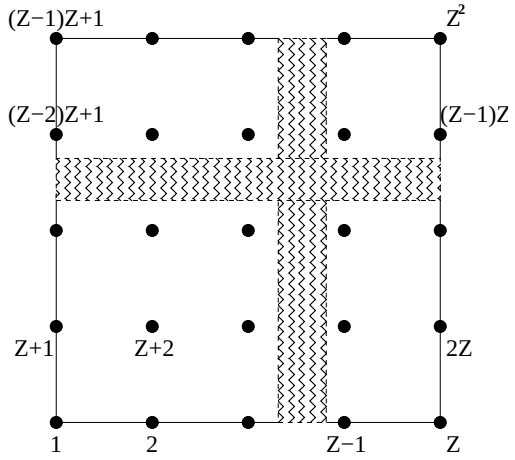

Figure 4.3: Local node numbering for a typical subdomain from the partitioned domain.

Hence, we chose to apply the same numbering scheme locally as we did

globally.

Take for example the subdomain in position $(1, 1)$ of the subdomain grid. Its local numbering is shown in the above figure, and its global numbering would be as follows:

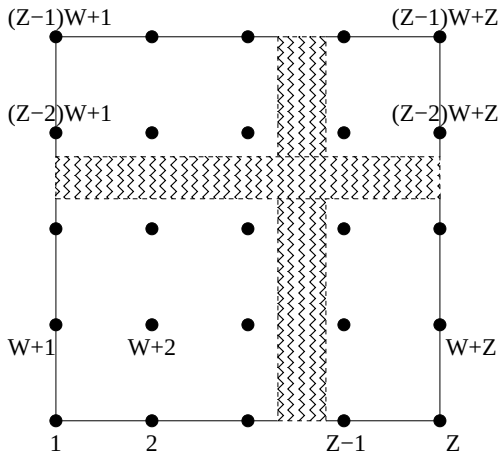

Figure 4.4: Global node numbering for subdomain $(1, 1)$ **from the partitioned domain.**

Similarly, the subdomain in position $(1, 2)$ of the subdomain grid would have the standard local node numbering and a global node numbering

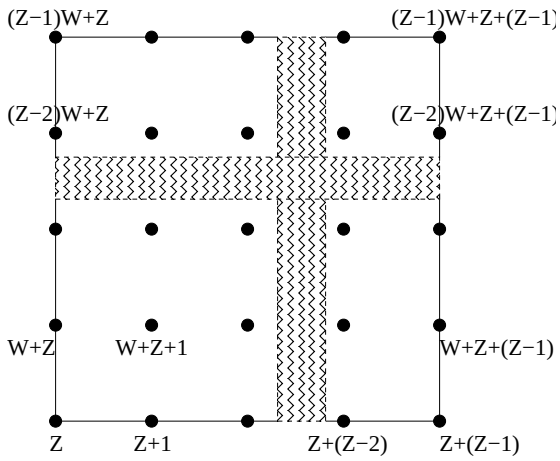

Figure 4.5: Global node numbering for subdomain $(1, 2)$ from the partitioned domain.

We see that the nodes on the partition line between the $(1, 1)$ subdomain and the $(1, 2)$ subdomain are shared by both subdomains. Also, the global node numbers of the $(1, 2)$ subdomain are essentially the global node numbers of the $(1, 1)$ subdomain increased by $Z - 1$.

Additionally, we find the global node numbering of the subdomain in position $(2, 1)$ of the subdomain grid to be

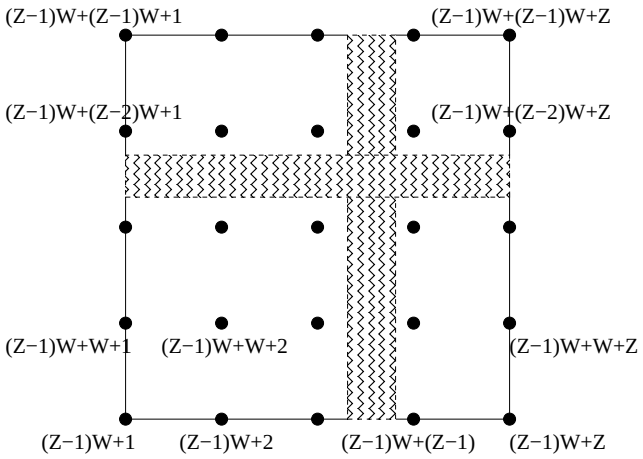

Figure 4.6: Global node numbering for subdomain $(2,1)$ from the partitioned domain.

Again, we see that the nodes on the partition line between the $(1,1)$ and $(2,1)$ subdomains are shared by both subdomains, and in particular, the $(Z-1)W+Z$ global node is shared by the $(1,1)$, $(1,2)$, and $(2,1)$ subdomains. Also, the global node numbers of the $(2,1)$ subdomain are the global node numbers of the $(1,1)$ subdomain increased by $(Z-1)W$.

Through the examination of these individual subdomains, we have found that given a global node numbering for a particular subdomain Ω_G, we can determine the global node numbering for the subdomain directly to the right of Ω_G by adding $Z-1$ to the global node numbers of Ω_G. Similarly, we can find the global node numbering for the subdomain directly above Ω_G by adding $(Z-1)W$ to the node numbers of Ω_G. We then extend this shifting concept to identify that for the subdomain one unit to the right and one unit above Ω_G in the grid of subdomains, we find the global node numbering by adding $(Z-1)+(Z-1)W=(Z-1)(W+1)$ to the node numbers of the original subdomain.

Thus, we can find the global node numbering for any subdomain in the grid of subdomains by first finding the global numbering for the $(1,1)$ subdo-

main and then adding the appropriate shift constant to those numbers. The shift constant would be calculated by comparing the location of the unknown subdomain to the location of the $(1, 1)$ subdomain.

For example, consider the $(4, 3)$ subdomain. This subdomain is located three units to the right and two units above the $(1, 1)$ subdomain on the the grid of subdomains. We would find the global node numbering for the $(4, 3)$ by adding the shift constant $3(Z-1)+2(Z-1)W = (Z-1)(2W+3)$ to the global node numbers of the $(1, 1)$ subdomain.

Using this shifting technique, we can create a vector for each subdomain that contains the global node numbering for that particular domain. To do this, we would store the global node number for local node i in the i^{th} position of the vector. Then, for the given subdomain, the direct relationship between the local node numbers and the global node numbers could be given as the ordered pair (i, N_i), where i is the position in the vector and N_i is the entry in that position.

Now, instead of partitioning Ω, let us divide Ω into a $P \times P$ grid of overlapping subdomains, where P is a divisor of $(W-1)$ and the overlaps are uniform:

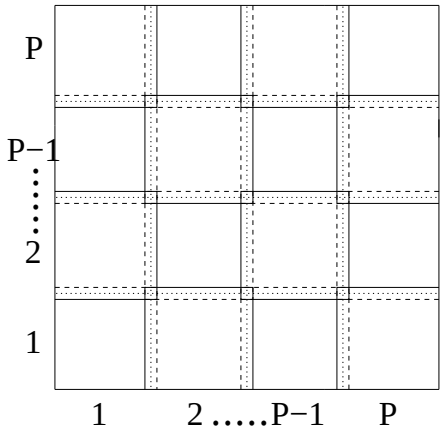

Figure 4.7: Division of the unit square into a $P \times P$ grid of overlapping subdomains. fig:overlap

We see from the above figure that not all subdomains are the same size. Take for example a corner subdomain, particularly the $(1,1)$ subdomain. This subdomain is similar in size to the $(1,1)$ subdomain in the partitioned grid except that its right edge and top edge have extended past the partitioned edge a specified number of nodes, called the overlap V. This means that if the partitioned subdomain was $\frac{W-1}{P}+1=Z$ nodes wide, then this subdomain is $Z+V$ nodes wide. Similarly, this subdomain is $Z+V$ nodes high, and we can say that the $(1,1)$ overlapped subdomain is a grid of $(Z+V)\times(Z+V)$ nodes. In fact, the size of the node grid applies to all of the four corner subdomains in Ω.

Consider the $(1,2)$ overlapped subdomain, which represents a non-corner, top/bottom boundary subdomain. In this subdomain, the left, right, and top edges have extended V nodes past the partitioned edge. Hence, this subdomain is a $(Z+2V)\times(Z+V)$ grid of nodes, and since this is a representative subdomain, we can say that the node grid size applies to all non-corner subdomains falling on the top or bottom edge of Ω.

Now, it is clear from the diagram that the $(2,1)$ subdomain has the exact same size as all non-corner, left/right boundary subdomains. By looking at the shape of the subdomain, we see that the top, bottom, and right edges have extended V nodes past the partitioned edge. This means that this subdomain, and all non-corner subdomains which fall on the left or right boundaries, is a $(Z+V)\times(Z+2V)$ grid of nodes.

Finally, consider the $(2,2)$ subdomain, which is an interior subdomain since it borders no boundaries. Every edge of this subdomain is extended V nodes past the partitioned edge, so we can say that this subdomain is a $(Z+2V)\times(Z+2V)$ grid of nodes. Again, by examining the shape of the interior subdomains, we see that the $(Z+2V)\times(Z+2V)$ grid size is true for all interior subdomains.

Due to the difference in subdomain sizes, we change our approach in finding the global node numbering of each subdomain. The goal is again to

create a vector for each subdomain that forms a relationship between the local node number and the global node number, but instead of using a direct shifting method as before, we use a modified shift to determine the global node number of local node 1 in the subdomain and reference the remaining nodes off local node 1.

First, let us describe the selected local node numbering of each subdomain type, based on the node grid size of each subdomain:

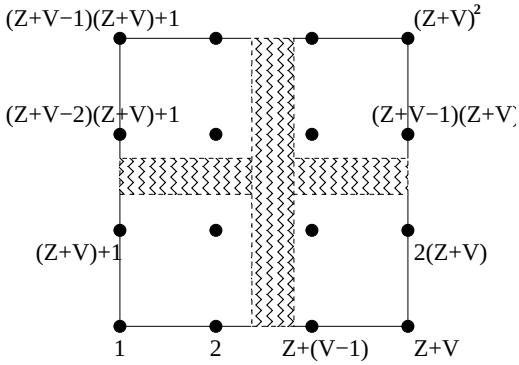

Figure 4.8: Local node numbering for a typical corner subdomain. These subdomains are each a $(Z+V)\times(Z+V)$ grid of nodes.

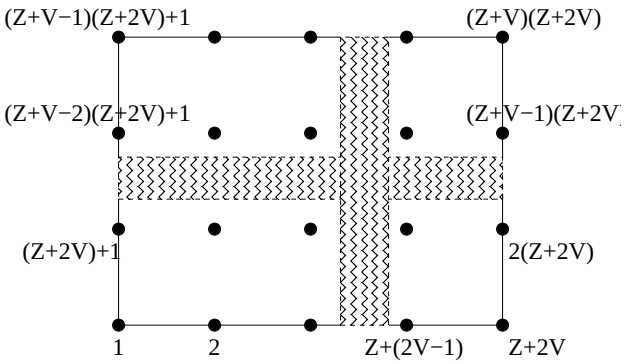

Figure 4.9: Local node numbering for a typical non-corner, top/bottom boundary subdomain. These subdomains are each a $(Z + 2V) \times (Z + V)$ grid of nodes.

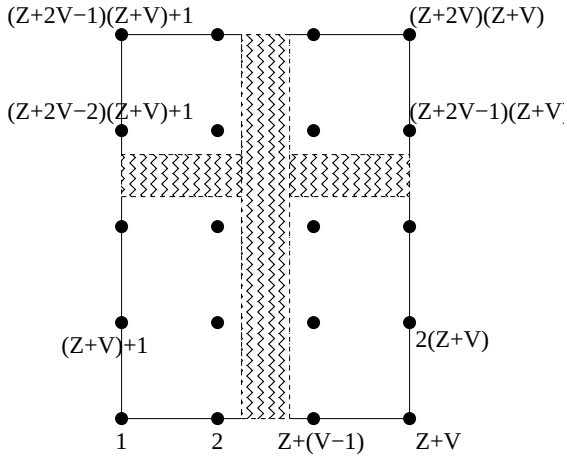

Figure 4.10: Local node numbering for a typical non-corner, left/right boundary subdomain. These subdomains are each a $(Z + V) \times (Z + 2V)$ grid of nodes.

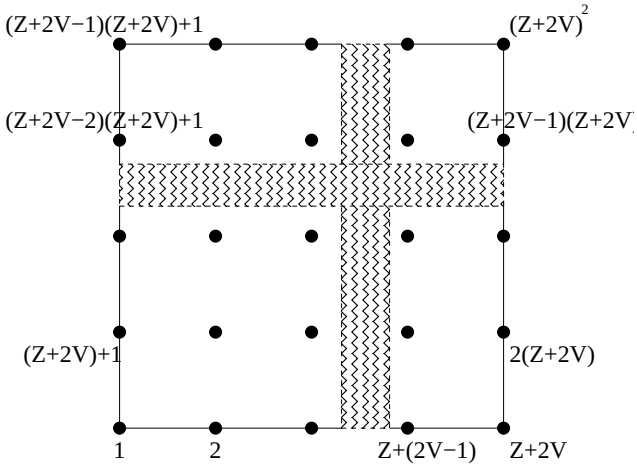

Figure 4.11: Local node numbering for a typical interior subdomain. These subdomains are each a $(Z + 2V) \times (Z + 2V)$ grid of nodes.

If we consider the size of each subdomain to be $DIM1 \times DIM2$, then we can generalize the previous four diagrams:

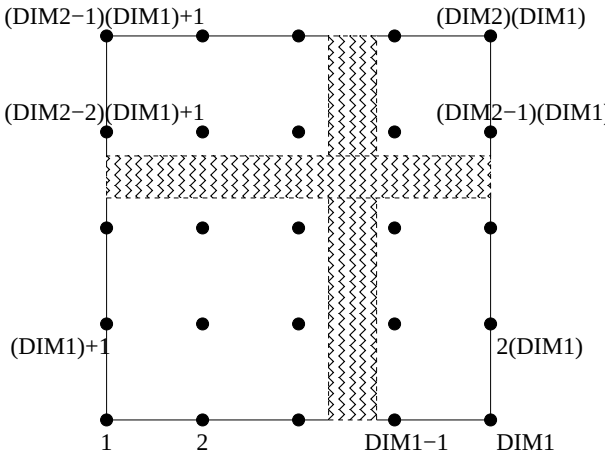

Figure 4.12: Generic local node numbering for any overlapped subdomain type.

Note that the values of $DIM1$ and $DIM2$ depend directly on the subdomain type, and that subdomain type is determined by the location of the subdomain in the subdomain grid. Therefore, for a given subdomain, we can find the local node numbering only by knowing the subdomain's location in the grid. Additionally, knowing the location of the subdomain allows us to find the global node number for local node 1, which in turn gives us a way to construct the global node numbering vector.

We see in figure 4.3.2 that local node 1 of any of the overlapping subdomains differs in global node number from local node 1 of the corresponding partitioned subdomain only by the shift created by the overlap region. For example, we determined that for the partitioned $(2, 1)$ subdomain, local node 1 had global node number $(Z - 1)W + 1$. For the overlapping $(2, 1)$ subdomain, however, we find that local node 1 has global node number $(Z-1)W+1-V$. Similarly, knowing that local node 1 in partitioned subdomain $(1, 2)$ had global node number $(Z-1)+1$, we can find the overlapping $(1, 2)$ subdomain has global node number $(Z - 1) + 1 - V$.

Thus, we can use the horizontal and vertical shifting factors for the partitioned case to find the global node number for local node 1 of the partitioned (m, n) subdomain. Then, we can apply the overlap shift to find the correct global node number of local node 1 of the overlapped (m, n) subdomain. Note that subdomains in row 1 of the grid do not require a vertical shift, and subdomains in column 1 do not require a horizontal shift. Otherwise, all other subdomains need both shifts.

Once the global node number of local node 1 is determined for a subdomain, the global node numbers of the remaining local nodes are referenced from the known node. If local node 1 has global node number X, we see that the global node numbering of the subdomain in question can be found as follows:

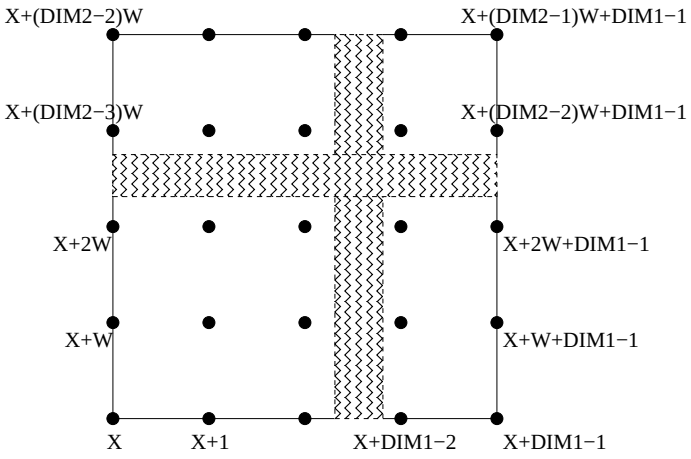

Figure 4.13: General global node numbering for any overlapped subdomain type.

Based on the modified shifting method, we can create the indexing vectors for all the subdomains using the following algorithm:

4.3.3 Splitting

Once the relationship between the global and local node numbers is known, we can split the global stiffness matrix to form the local matrices required for formulate the local problem. Essentially, the formation of any local matrix is done by extracting the entries of the global stiffness matrix which correspond to the nodes found in that particular subdomain.

As an example, consider the $(1, 1)$ overlapped subdomain from 4.3.2, shown with its global node numbers:

Here, W is the number of nodes in each row and column of the global triangulation, and Q is the number of nodes in each row and column of the subdomain.

The local matrix for this subdomain only involves the global nodes shown, and thus should only be comprised of the portion of the global stiffness matrix

that involves these global nodes. If $A(m,n)$ denotes the element of global stiffness matrix A in the m^{th} row and n^{th} column, then the local matrix A_{s1} can be written as

$$A_{s1} = \begin{bmatrix} A(1,1) & A(1,2) & \cdots & A(1,(Q-1)W+Q) \\ A(2,1) & A(2,2) & & \vdots \\ \vdots & & \ddots & \vdots \\ A((Q-1)W+Q,1) & \cdots & \cdots & A((Q-1)W+Q,(Q-1)W+Q) \end{bmatrix}.$$

Recall that the index vector created the this subdomain contains the global node numbers shown in figure..... Therefore, we can use this vector to locate the required entries in the global stiffness matrix. If $\vec{s_1}$ is the index vector of this subdomain and $\vec{s_1}(m)$ is the m^{th} entry of that vector, then

$$A_{s1} = \begin{bmatrix} A(\vec{s_1}(1),\vec{s_1}(1)) & A(\vec{s_1}(1),\vec{s_1}(2)) & \cdots & A(\vec{s_1}(1),\vec{s_1}(Q^2)) \\ A(\vec{s_1}(2),\vec{s_1}(1)) & A(\vec{s_1}(2),\vec{s_1}(2)) & & \vdots \\ \vdots & & \ddots & \vdots \\ A(\vec{s_1}(Q^2),\vec{s_1}(1)) & \cdots & \cdots & A(\vec{s_1}(Q^2),\vec{s_1}(Q^2)) \end{bmatrix}.$$

This idea holds for the n^{th} subdomain, which has an index vector $\vec{s_n}$ of dimension Q^2:

$$A_{sn} = \begin{bmatrix} A(\vec{s_n}(1),\vec{s_n}(1)) & A(\vec{s_n}(1),\vec{s_n}(2)) & \cdots & A(\vec{s_n}(1),\vec{s_n}(Q^2)) \\ A(\vec{s_n}(2),\vec{s_n}(1)) & A(\vec{s_n}(2),\vec{s_n}(2)) & & \vdots \\ \vdots & & \ddots & \vdots \\ A(\vec{s_n}(Q^2),\vec{s_n}(1)) & \cdots & \cdots & A(\vec{s_n}(Q^2),\vec{s_n}(Q^2)) \end{bmatrix}.$$

Due to the structure of the Dirichlet stiffness matrix, all corner subdomains in a Dirichlet system would have a local matrix exactly the same as the

first corner subdomain, or A_{s1}. The same is true for the three other types of subdomains: non-corner top/bottom subdomains, non-corner left/right subdomains, and interior subdomains. Therefore, when dealing with a system having Dirichlet boundary conditions, it is only necssary to construct four local matrices, one for each type of subdomain.

In the Neumann case, however, since the Neumann stiffness matrix is structured differently than the Dirichlet matrix, more than four local matrices must be constructed. The procedure described above can be followed in constructing the local matrices in the Neumann case, but more local matrices would need to be created.

4.3.4 Local Solver Set-up

4.4 Iterative Method

4.5 Numerical Results

Bibliography

book:axel [1] Owe Axelsson. *Iterative Solution Methods*. Cambridge University Press, New York, NY, 1994.

book:smith [2] Petter E. Bjorstad Barry F. Smith and William D. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, New York, NY, 1996.

book:demmel [3] James W. Demmel. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, PA, 1997.

k:deuflhard [4] Peter Deuflhard and Andreas Hohmann. *Numerical Analysis: A First Course in Scientific Computation*. Walter de Gruyter and Co., New York, NY, 1995.

book:golub [5] Gene H. Golub and Charles F. Van Loan. *Matrix Computations, 2nd Edition*. The John Hopkins University Press, Baltimore, Baltimore, MD, 1989.

ook:johnson [6] Claes Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Cambridge University Press, New York, NY, 1987.

sark_thesis [7] Marcus V. Sarkis Martins. *Schwarz Preconditioners for Elliptic Problems with Discontinuous Coefficients Using Conforming and Non-Corming El-*

ements. PhD thesis, Courant Institute of Mathematical Sciences, New York University, September 1994.

k:tref_bau [8] Lloyd N. Trefethen and David Bau III. *Numerical Linear Algebra*. SIAM, Philadelphia, PA, 1997.

book:varga [9] Richard S. Varga. *Matrix Iterative Analysis, 2nd Edition*. Springer, New York, NY, 2000.