Preconditioning of Iterative Methods for the Transport Equation

Jack C. H. Blake

MSc in Modern Applications of Mathematics

September 2011

Preconditioning of Iterative Methods for the Transport Equation

submitted by Jack C. H. Blake

for the degree of MSc in Modern Applications of Mathematics

of the University of Bath

Summary

Climate change and energy concerns are boosting the search for and interest in alternative energy sources. Whether nuclear energy is a feasible alternative or not is still being questioned, but there is no doubt that obtaining a thorough understanding of this option is important. Efficiently solving the Boltzmann Transport equation is a requirement of reactor criticality computations. In this document we look at several iterative solution methods for solving linear systems of equations. To discretise the Transport equation we define and understand the method of discrete ordinates, and apply it to this case. We also derive the P_1 diffusion approximation and use it to define Diffusion Synthetic Acceleration (DSA), which can be used to speed up iterative solution methods. Along the way we prove an original result concerning the convergence of source iteration. Lastly, we explore how DSA can be applied in practice to the transport equation. We implement the iterative methods discussed earlier on a model problem from A. Greenbaum, *Iterative Methods for Solving Linear Systems*, comparing our convergence results with the ones given there.

ACKNOWLEDGEMENTS

First a huge thank you to my brilliant supervisor, Ivan Graham. Your vast knowledge and patience have been invaluable to me whilst writing this dissertation. I very much look forward to working with you and Alastair over the coming years of my PhD. Next I would like to say thank you to everyone in the 'Cave' for entertaining discussions that occasionally even strayed into mathematics. You have all been a huge help in keeping spirits up and motivation high over this long summer. In particular Carla (original crew!), Dan, Gowri, Rich, Roxanne, Ruga (15 days is still not possible...), Sarah, Stephen and Steven (for all the helpful talks). Lastly I would like to thank Wannah for being an awesome friend, my lovely girlfriend Tara for putting up with all the maths, and my family for so much support throughout the year and for generally being amazing. I love you all.

CONTENTS

CHAPTER 1

INTRODUCTION

A frequently arising problem in many areas of mathematics is to solve the deceptively simple looking matrix-vector equation,

$$A\mathbf{x} = \mathbf{b}, \tag{1.1}$$

where A is an $m \times n$ matrix, $\mathbf{b}$ is a known m-dimensional vector and $\mathbf{x}$ is an unknown n-dimensional vector. In this study we will consider the case where n equals m. Typically, methods for solving such systems numerically are iterative with the rate of convergence dependent upon the specific nature of the problem. Certain methods exploit features of the system (such as sparsity or symmetry of the coefficient matrix, A) to reduce the complexity of the required calculations, or to reduce the amount of simultaneous storage required. These methods start with an initial guess for the solution, typically denoted $\mathbf{x}_0$, and apply some process to arrive at a new (and hopefully better) guess $\mathbf{x}_1$. This process is then repeated until the solution is accurate to some level of tolerance. To obtain some information about how accurate any given approximation is, a value known as the *residual* is considered. Iterative methods for solving linear systems of the form (1.1), will be considered in chapter 2. In particular we will look at the method of *Generalised Minimal Residuals* which is a commonly used and efficient linear solver, though it requires increasing amounts of storage as it proceeds.

Clearly, if the coefficient matrix, A, in (1.1) is the identity matrix then the system will converge to the correct solution in one iteration. Thus one way to improve the efficiency of finding a solution would be to multiply through by A^{-1}. Unfortunately, finding this inverse is in itself a difficult problem. Instead, it may be possible to multiply through by some easily calculable approximation to the inverse, preferably one for which matrix-vector multiplication is also straightforward. This process is known as *preconditioning*, and will be looked at in chapter 4.

In this study we want to explore how iterative methods and preconditioning can be used to solve the following eigenvalue problem for the Boltzmann Transport equation

$$\mathcal{T}\Psi - \mathcal{S}\Psi = \lambda\mathcal{F}\Psi, \tag{1.2}$$

where the operators $\mathcal{T}$, $\mathcal{S}$ and $\mathcal{F}$ describe transport, scattering and fission respectively and $\Psi \equiv \Psi\left(\mathbf{r}, E, \mathbf{\Omega}\right)$ is the flux of neutrons per unit volume, with energy $E \in \mathbb{R}^+$ at position $\mathbf{r} \in \mathbb{R}^3$ in direction $\mathbf{\Omega} \in \mathbb{S}^2$ (the unit sphere). This will be fully introduced in chapter 3, where we will see that to solve it involves solving a linear system of equations of the form

$$\left(\mathcal{T} - \mathcal{S} - \mathcal{F}\right)\Psi = \mathcal{Q}, \tag{1.3}$$

where $\mathcal{Q} \equiv \mathcal{Q}\left(\mathbf{r}, \mathbf{\Omega}\right)$ is a known non-fission source term of neutrons [1]. By considering just one energy group and with isotropic scattering, this can be written in full form as

$$\begin{gathered} \mathbf{\Omega} \cdot \nabla\Psi\left(\mathbf{r}, \mathbf{\Omega}\right) + \sigma_t\left(\mathbf{r}\right)\Psi\left(\mathbf{r}, \mathbf{\Omega}\right) - \left(\sigma_s\left(\mathbf{r}\right) + \nu\sigma_f\left(\mathbf{r}\right)\right)\phi = \mathcal{Q}, \\ \phi \equiv \frac{1}{4\pi}\int_{\mathbb{S}^2} \Psi\left(\mathbf{r}, \mathbf{\Omega}'\right)\,\mathrm{d}\mathbf{\Omega}'. \end{gathered} \tag{1.4}$$

where $\sigma_t\left(\mathbf{r}\right)$ is the known total cross section, $\sigma_s\left(\mathbf{r}\right)$ the known scatter cross section and $\sigma_f\left(\mathbf{r}\right)$ the known fission cross section, each at position $\mathbf{r}$, which will be defined in chapter 3. This is known as the *mono-energetic, steady-state, linear 3D Boltzmann transport equation*, and is an important problem in nuclear engineering [2]. In chapter 3 we will look at the *discrete ordinates* method for solving such problems, and develop the matrix formation given in [3].

If the scatter and fission cross sections are combined, (1.4) can be written in the following form as given in [4] and used in [5]

$$\Omega \cdot \nabla\Psi\left(r, \Omega\right) + \sigma_t\left(r\right)\Psi\left(r, \Omega\right) = \int_{\mathbb{S}^2} \Psi\left(r, \Omega'\right)\sigma_s\left(r, \Omega \cdot \Omega'\right)\,\mathrm{d}\Omega' + q\left(r, \Omega\right), \tag{1.5}$$

in which Ω and Ω' represent the *incoming* and *outgoing* (or *reflected*) neutron angles respectively. This form of the Transport equation will be used in chapter 5 to derive the so-called P_1 diffusion approximation [6].

In chapter 6 we start by defining *source iteration*. In theorem 6.1 we prove a new, or undocumented, convergence result for systems with constant cross sections. Then we introduce *synthetic acceleration* schemes, before finally using the derived P_1 approximation from chapter 5 to define and explain the *diffusion synthetic acceleration* (DSA) method of accelerating (or equivalently preconditioning, see section 6.2.1) iterative schemes [7].

Finally, in chapter 7 we will look at how DSA can be implemented computationally. We will also apply the iterative methods covered in chapter 2 to a model problem given in [3] and produce a graph showing the obtained convergence results. These will be compared to those found in [3], and any similarities and differences discussed. All code written for this implementation is included in appendix A, which also includes functions from [1] and [8].

CHAPTER 2

ITERATIVE SOLUTION METHODS

In this chapter we introduce and explain several different *iterative solution methods* for solving linear systems of equations. These methods work by repeatedly approximating the solution, using the previous approximation to hopefully improve upon the next. Two important factors of such algorithms are their rate of convergence and their complexity, as these determine how fast a suitably accurate approximation can be found. Some iterative methods are very complicated or very specific about what type of system they will solve, for example the *total reduction* method presented in [9] for solving the Poisson equation. Others are very general and are frequently applied to all sorts of systems, however they are often less efficient in that they take longer to converge to the correct solution, for example *simple iteration* which will be discussed later in this chapter. Techniques for speeding up such iterative methods can be used as a way to improve efficiency, such as *preconditioning* (see chapter 4).

In this chapter we introduce and describe three methods for iteratively solving linear systems: *simple iteration* (SI), *Orthomin(1)* (OM-1) and the *Generalised Minimal Residuals* (GMRES) method. SI and OM-1 are simple and very similar in nature, however GMRES is quite different and will require some background information before it can be fully explained.

2.1 Simple Iteration

Simple iteration is a very intuitive method for improving upon subsequent iterations. To explain it we work from the linear system (1.1) introduced in chapter 1: $Ax = b$ but with $m = n$, so A is an $n \times n$ matrix, b is a known n-dimensional vector and x is an unknown n-dimensional vector. Say we have some approximate solution, x_k, then the current error in the solution is given exactly by

$$\mathbf{err}_k = x - x_k. \tag{2.1}$$

Unfortunately evaluating this involves calculating the inverse matrix A^{-1} which, if known, would allow for direct solution of the problem and so no need for any approximations to be made. Typically this is difficult to calculate, but we would expect to be able to compute some approximation of the error by evaluating

$$r_k = b - Ax_k, \tag{2.2}$$

noting here that $A\,\mathbf{err}_k = r_k$. This vector, r_k, is known as the *residual vector* for iteration k, and is a central part of most iterative solution methods. Using this 'estimated error', we would expect that a better approximation to the solution may be obtained by calculating

$$x_{k+1} = x_k + r_k. \tag{2.3}$$

This process when started with an initial guess, x_0, and used to obtain successive guesses, $x_1, x_2, \ldots$, is called *simple iteration.* Iteration (2.3) can be stated in the form of an algorithm as follows:

```
Algorithm: Simple Iteration
For initial guess $x_0$, compute $r_0 = b - Ax_0$

for $k = 0, 1, \ldots$ do
    $x_{k+1} = x_k + r_k$
    $r_{k+1} = b - Ax_{k+1}$
end for
```

This is carried out until some desired tolerance level is reached by the residual.

To further explore when simple iteration will converge, we use (2.1) and (2.3) to obtain

$$\begin{aligned} \mathbf{err}_{k+1} &= A^{-1}b - x_{k+1} \\ &= A^{-1}b - (x_k + r_k) \\ &= \left(A^{-1}b - x_k\right) - A\,\mathbf{err}_k \\ &= (I - A)\,\mathbf{err}_k. \end{aligned} \tag{2.4}$$

Using this relation repeatedly we get

$$\mathbf{err}_{k+1} = (I - A)\,\mathbf{err}_k = \cdots = (I - A)^{k+1}\,\mathbf{err}_0. \tag{2.5}$$

Taking norms on both sides yields

$$\|\mathbf{err}_{k+1}\| \leq \| (I - A)^{k+1} \| \cdot \|\mathbf{err}_0\|, \tag{2.6}$$

in which $\|\cdot\|$ is any vector norm, with the matrix norm being the corresponding induced norm, $\|A\| \equiv \max_{\|x\|=1}\|Ax\|$ [3]. The following lemma is taken from A. Greenbaum [3] p.26 (with slight changes); it gives a test for convergence of simple iteration.

Lemma 2.1:
$\|\mathbf{err}_k\| \to 0$ *as* $k \to 0$ *for every initial error* $\mathbf{err}_0$ *if and only if*

$$\lim_{k\to\infty} \| (I - A)^k \| = 0. \tag{2.7}$$

Proof.
See A. Greenbaum [3], Chapter 2, pages 26-27. □

The next iterative solution method we will look at is called the *Orthomin(1)* method.

2.2 Orthomin(1) Method

The *Orthomin(1)* method is a way of improving upon simple iteration seen in the last section. This is achieved by introducing artificial parameters, computed at each iteration, into the algorithm. Looking at (2.3) it would be natural to attempt to improve upon it by scaling the residual, say by a factor a_k. Thus, (2.3) would become

$$x_{k+1} = x_k + a_k r_k. \tag{2.8}$$

Now, we have that

$$\begin{aligned} r_{k+1} &= b - A\left(x_k + a_k r_k\right), \\ &= b - Ax_k - a_k A r_k, \\ &= r_k - a_k A r_k, \end{aligned} \tag{2.9}$$

so minimising the norm $\|r_{k+1}\|^2$ is equivalent to solving the partial differential equation

$$\frac{\partial}{\partial a_k}\left(r_k - a_k A r_k\right)^2 = 0. \tag{2.10}$$

With $\langle \boldsymbol{u}, \boldsymbol{v} \rangle \equiv \boldsymbol{u}^T \boldsymbol{v}$, the solution of this is found as follows

$$\begin{aligned}
0 &= \frac{\partial}{\partial a_k}\left(r_k - a_k A r_k\right)^2, \\
&= \frac{\partial}{\partial a_k}\Big(\left(r_k - a_k A r_k\right)^T \left(r_k - a_k A r_k\right)\Big), \\
&= \frac{\partial}{\partial a_k}\left(r_k^T r_k - a_k (A r_k)^T r_k - a_k r_k^T A r_k + a_k^2 (A r_k)^T A r_k\right), \\
&= \frac{\partial}{\partial a_k}\left(\|r_k\|^2 - 2a_k\langle r_k, A r_k\rangle + a_k^2\|A r_k\|^2\right), \\
&= -2\langle r_k, A r_k\rangle + 2a_k\|A r_k\|^2.
\end{aligned} \tag{2.11}$$

Thus

$$a_k = \frac{\langle r_k, A r_k\rangle}{\|A r_k\|^2}, \tag{2.12}$$

and hence we can minimise the 2-norm of r_{k+1} by choosing

$$a_k \;=\; \frac{\langle r_k, A r_k\rangle}{\langle A r_k, A r_k\rangle}. \tag{2.13}$$

Similarly to simple iteration, (2.8) can now be stated in the form of an algorithm as follows:

```
Algorithm: Orthomin(1)
For initial guess $x_0$,
compute $r_0 = b - Ax_0$ and $a_0 = \frac{\langle r_0, Ar_0\rangle}{\langle Ar_0, Ar_0\rangle}$

for $k = 0, 1, \ldots$ do
    $x_{k+1} = x_k + a_k r_k$
    $r_{k+1} = b - Ax_{k+1}$
    $a_{k+1} = \frac{\langle r_{k+1}, Ar_{k+1}\rangle}{\langle Ar_{k+1}, Ar_{k+1}\rangle}$
end for
```

Again, this is carried out until some desired tolerance level is reached by the residual.

With the choice (2.13) of a_k and using (2.9), we have that

$$\begin{aligned}
r_{k+1} &= r_k - \frac{r_k^T A r_k}{\|A r_k\|^2} A r_k, \\
&= r_k - \frac{r_k^T A r_k}{\|A r_k\|}\frac{A r_k}{\|A r_k\|}.
\end{aligned} \tag{2.14}$$

Now, $\frac{Ar_k}{\|Ar_k\|}$ is just a unit vector in the direction of Ar_k, and

$$\begin{aligned} \frac{r_k^T A r_k}{\|Ar_k\|} &= \|r_k\| \frac{r_k^T A r_k}{\|r_k\|\|Ar_k\|}, \\ &= \|r_k\| \cos(\theta), \end{aligned} \tag{2.15}$$

where θ is the angle between r_k and Ar_k. $\|r_k\| \cos(\theta)$ is just the length of r_k in the direction of Ar_k, and so $\frac{r_k^T A r_k}{\|Ar_k\|^2} A r_k$ is the projection of r_k on to Ar_k. This means that r_{k+1} is equal to r_k minus its projection on to Ar_k, and it follows that

$$\begin{aligned} &\|r_{k+1}\| \leq \|r_k\| \quad \forall k = 0, 1, 2, \ldots \quad \text{and} \\ &\|r_{k+1}\| = \|r_k\| \quad \text{if and only if } r_k \text{ is orthogonal to } Ar_k. \end{aligned}$$

Using this inequality, monotonicity of the decrease in the value of the residual at each iteration can be proved [3]. This result can then be used to prove the following theorem regarding conditions for convergence of the Orthomin(1) method. Here, $(\cdot)^H$ denotes the *Hermitian transpose* of $(\cdot)$, and $\mathcal{F}(Z)$ denotes the *field of values* of a matrix Z, which is defined to be the set of all complex numbers of the form $z^H Z^H z / z^H z$, where z is any non-zero complex vector.

Theorem 2.2:
The iteration (2.8) *with coefficient formula* (2.13) *converges to the solution* $A^{-1}b$ *for all initial vectors* r_0 *if and only if* $0 \notin \mathcal{F}\left(A^H\right)$. *In this case, the 2-norm of the residual satisfies*

$$\|r_{k+1}\| \quad \leq \quad \sqrt{1 - d^2/\|A\|^2}\, \|r_k\|, \quad \forall k, \tag{2.16}$$

where d *is the distance from the origin to the field of values of* A^H.

Proof.
See A. Greenbaum [3], chapter 2, page 31. □

We now move on to look at the *GMRES* method for solving linear systems of equations.

2.3 Generalised Minimal Residuals (GMRES)

The method of Generalised Minimal Residuals, or *GMRES*, is a method for solving linear systems of equations such as (1.1). It works by computing approximations to x whilst minimising the residual r_k at each step. While it requires less storage in comparison to other similar methods (such as *Orthodir(n)*) [3], it is generally considered an expensive method in terms of storage. For solving systems of high dimension, other more storage efficient methods are often preferred.

Before we look at the GMRES method we first introduce the *Krylov subspace* and *Arnoldi's method*, which are integral to GMRES. In this section we will use the notation that ξ_b^a is the ath unit b-vector, i.e.

$$\xi_b^a = \underbrace{(0, \ldots, 0, \overbrace{1}^{a\text{th entry}}, 0, \ldots, 0)^T}_{\text{length } b}.$$

2.3.1 The Krylov Subspace

The *Krylov subspace* of order k is a space generated by a square matrix, A, and a vector, b, each of dimension n. It is defined as

$$\mathcal{K}_k(A, b) = \operatorname{span}\left\{b, Ab, A^2b, \ldots, A^{k-1}b\right\}, \tag{2.17}$$

i.e. it is the span of the first k powers of A multiplied by b. The Krylov subspace was first introduced by A. N. Krylov in his 1931 paper [10]. For powers greater than $n-1$ the vectors A^kb are linearly dependent, so methods which use the Krylov subspace usually involve some method of orthogonalisation. In GMRES this method is Arnoldi's method.

2.3.2 Arnoldi's Method

Arnoldi's method is based on the well-known Gram-Schmidt orthonormalisation process, and is an iterative method that is used to find the eigenvalues of general matrices. It achieves this by implementing the modified Gram-Schmidt process to construct an orthonormal basis for the Krylov subspace (2.17), denoted $q_1, q_2, \ldots, q_n$. Modified Gram-Schmidt is used because computational rounding errors lead to a loss of orthogonality, and hence instability, when applying standard Gram-Schmidt. The modified version avoids this by reducing the possibility of compounded errors, i.e. orthogonality is ensured for each newly calculated vector regardless of rounding errors that may have been introduced earlier [11]. These vectors are known as the *Arnoldi vectors.* The method can be stated as an algorithm as follows:

```
Algorithm: Arnoldi's Method
Given an arbitrary vector $q_1$ with $\|q_1\| = 1$

for $j = 1, 2, \ldots$ do
    $\tilde{q}_{j+1} = Aq_j$
    for $i = 1, 2, \ldots, j$ do
        $h_{i,j} = \langle \tilde{q}_{j+1}, q_i \rangle$
        $\tilde{q}_{j+1} \leftarrow \tilde{q}_{j+1} - h_{i,j} q_i$
    end for
    $h_{j+1,j} = \|\tilde{q}_{j+1}\|$
    $q_{j+1} = \frac{\tilde{q}_j + 1}{h_{j+1,j}}$
end for
```

If we define Q_k to be the $n \times k$ matrix with the orthonormal basis vectors $q_1, \ldots, q_k$ for columns, then each iteration can be written in matrix form as

$$AQ_k = Q_k H_k + h_{k+1,k} q_{k+1} \xi_k^{k^T} = Q_{k+1} H_{k+1,k}. \tag{2.18}$$

Here H_k is a $k \times k$ upper Hessenberg matrix composed of the elements $h_{i,j}$ for $j = 1, \ldots, k$ and $i = 1, \ldots, \min\{j+1, k\}$, with all other elements zero. Upper Hessenberg matrices are upper triangular matrices but with nonzero elements on the subdiagonal also. $H_{k+1,k}$ has the matrix H_k as its top $k \times k$ block, and a row at the bottom which is all zero except for the $(k+1, k)$th element which is $h_{k+1,k}$ [3].

2.3.3 The GMRES Method (Basic Steps)

The GMRES method for iteratively solving linear systems of equations uses Arnoldi's method applied to the Krylov subspace

$$\mathcal{K}_n = \text{span}\left\{r_0, Ar_0, A^2 r_0, \ldots, A^{n-1} r_0\right\}, \tag{2.19}$$

where r_0 is the residual of the system obtained by the initial guess x_0. In the GMRES method, the kth iterate is found via

$$x_k = x_0 + Q_k y_k \tag{2.20}$$

where y_k is some vector to be determined and Q_k is as defined in section 2.3.2. The vector y_k is chosen such that $r_k = r_0 - AQ_k y_k$ has the minimum 2-norm. This is accomplished by determining the y_k that solves the least squares problem

$$\begin{aligned}\min_y \| r_0 - AQ_k y \| &= \min_y \| r_0 - Q_{k+1} H_{k+1,k} y \| \\ &= \min_y \Big\| Q_{k+1} \left(\|r_0\| \xi^1_{k+1} - H_{k+1,k} y \right) \Big\| \\ &= \min_y \Big\| \|r_0\| \xi^1_{k+1} - H_{k+1,k} y \Big\|.\end{aligned} \tag{2.21}$$

Here, the equality $r_0 = Q_{k+1}\|r_0\|\xi^1_{k+1}$ holds since $Q_{k+1}\xi^1_{k+1}$ is equal to the first column of Q_{k+1}. This is the first of the orthonormal basis vectors for the Krylov subspace (2.19), which is the normalised vector $r_0/\|r_0\|$.

We are now in a position to state the fundamental steps of the GMRES method:

```
Algorithm: Basic steps of GMRES
Given an initial $x_0$
    compute $r_0 = b - Ax_0$
    set $q_1 = \frac{r_0}{\|r_0\|}$

for $k = 1, 2, \ldots$ do
    Compute $q_{k+1}$ and $h_{i,k}, i = 1, \ldots, k+1$
        using the Arnoldi Algorithm
    Find $y_k$, the solution to the least
        squares problem $\min_y \Big\| \|r_0\| \xi^1_{k+1} - H_{k+1,k} y \Big\|$
    Set $x_k = x_0 + Q_k y_k$
end for
```

In this algorithm, the only unknown method is that of solving the least squares problem. We now explore a standard method of solution for such problems.

2.3.4 Solving the Least Squares Problem

We are looking to solve the least squares problem

$$\min_y \Big\| \|r_0\| \xi^1_{k+1} - H_{k+1,k} y \Big\|. \tag{2.22}$$

One standard solution method is using QR factorisation [12] to factor the matrix $H_{k+1,k}$ into the product of a $k+1 \times k+1$ unitary matrix F^H (i.e. a matrix such that $FF^H = F^H F = I$, where $[\cdot]^H$ denotes the Hermitian (or conjugate) transpose), and a $k+1 \times k$ upper triangular (UT) matrix R, with last row zero. The QR factorisation can be found via the use of Givens rotations [13] [12] which are introduced below in (2.24). From this, we can then find y by solving

$$R_{k \times k}\, y = \|r_0\| \left(F \xi^1_{k+1} \right)_{k \times 1}, \tag{2.23}$$

a UT system of equations where $R_{k\times k}$ is the upper $k\times k$ block of R and $\left(F\xi^1_{k+1}\right)_{k\times 1}$ is the top k entries of the first column of F [14].

Now, suppose we already have the QR factorisation of $H_{k+1,k}$, then being able to cheaply compute the QR factorisation of $H_{k+2,k+1}$ from this would greatly reduce the computational complexity of the method. To this end, let $\mathcal{G}_i$ denote a *Givens rotation matrix* [12] that rotates the unit vectors ξ^i_l and ξ^{i+1}_l (with l variable) through the angle θ_i:

$$\mathcal{G}_i = \begin{array}{c} \begin{array}{cc} i & i+1 \\ \downarrow & \downarrow \end{array} \\ \left(\begin{array}{cccc} I & & & \\ & c_i & s_i & \\ & -\bar{s}_i & c_i & \\ & & & I \end{array}\right) \end{array} \begin{array}{l} \\ \\ \leftarrow i \\ \leftarrow i+1 \\ \\ \end{array} , \tag{2.24}$$

where $c_i \equiv \cos(\theta_i)$ and $s_i \equiv \sin(\theta_i)$. Here the size of the second identity block is context dependent, i.e. it is as large as it needs to be to ensure that each multiplication makes mathematical sense. Each rotation, when applied to a matrix, reduces an element in the subdiagonal of the matrix to zero yielding the required UT matrix. The cumulative effect of a series of Givens rotation matrices produces the unitary (hence orthogonal) matrix F, i.e. $\mathcal{G}_k\mathcal{G}_{k-1}\ldots\mathcal{G}_1 = F$.

When using Givens rotations, the sparse rotation matrix, $\mathcal{G}_i$, is not usually formed. Instead a Givens rotation procedure is performed which is equivalent to using the matrices, but without the added complexity of handling sparse elements. In the full GMRES algorithm we will work without building the matrices, however for the sake of simplicity we will maintain the matrix notation in this section.

Assume that the rotations $\mathcal{G}_1,\ldots,\mathcal{G}_k$ have already been applied to $H_{k+1,k}$, giving

$$\left(\mathcal{G}_k\mathcal{G}_{k-1}\ldots\mathcal{G}_1\right)H_{k+1,k} = R^{(k)} = \begin{pmatrix} \square & \square & \ldots & \square \\ & \square & \ldots & \square \\ & & \ddots & \vdots \\ & & & \square \\ 0 & 0 & \ldots & 0 \end{pmatrix}, \tag{2.25}$$

with $\square$ denoting any nonzero entry. From this we want to obtain $R^{(k+1)}$, the UT factor for $H_{k+2,k+1}$. To get this, we first note that the upper left $k+1\times k$ block of $H_{k+2,k+1}$ is equal to $H_{k+1,k}$, and its final column is composed of the relevant $h_{i,k+1}$ values, with the remainder of its entries on the last row equal to zero. Next we apply the Givens rotations to the last column of $H_{k+2,k+1}$ (since the first k columns are affected in the same way as in $H_{k+1,k}$). This gives

$$(\mathcal{G}_k\mathcal{G}_{k-1}\dots\mathcal{G}_1)\,H_{k+2,k+1} = \begin{pmatrix} \square & \square & \dots & \square & \square \\ & \square & \dots & \square & \square \\ & & \ddots & \vdots & \vdots \\ & & & \square & \square \\ 0 & 0 & \dots & 0 & d \\ 0 & 0 & \dots & 0 & h_{k+2,k+1} \end{pmatrix}. \tag{2.26}$$

Now the next rotation to be applied, $\mathcal{G}_{k+1}$, is chosen so that $h_{k+2,k+1}$ is eliminated. This is achieved by setting

$$\left.\begin{aligned} c_{k+1} &= \frac{|d|}{\sqrt{|d|^2+|h_{k+2,k+1}|^2}} \\ \bar{s}_{k+1} &= \frac{c_{k+1}h_{k+2,k+1}}{d} \end{aligned}\right\}\ \text{if } d \neq 0, \qquad \left.\begin{aligned} c_{k+1} &= 0 \\ s_{k+1} &= 1 \end{aligned}\right\}\ \text{if } d = 0, \tag{2.27}$$

in (2.24). Under these we can find $R^{(k+1)}$ via

$$\mathcal{G}_{k+1}R^{(k)} = R^{(k+1)} = \begin{pmatrix} \square & \square & \dots & \square & \square \\ & \square & \dots & \square & \square \\ & & \ddots & \vdots & \vdots \\ & & & \square & \square \\ 0 & 0 & \dots & 0 & \blacksquare \\ 0 & 0 & \dots & 0 & 0 \end{pmatrix}. \tag{2.28}$$

Using the definitions in (2.27) we get that

$$\blacksquare = \begin{cases} \dfrac{d}{|d|}\sqrt{|d|^2+|h_{k+2,k+1}|^2} & \text{if } d \neq 0 \\ h_{k+2,k+1} & \text{if } d = 0 \end{cases}. \tag{2.29}$$

Since $h_{k+2,k+1} \neq 0$, the $k+1$th diagonal entry of $R^{(k+1)}$ will never be zero, hence we have an UT matrix. Looking again at equation (2.23), we can calculate the right hand side by applying each of the rotations to ξ^1_{k+1}. Now, from the initial definitions

$$\begin{aligned} \|b - Ax_k\| &= \big\|\ \|r_0\|\xi^1_{k+1} - H_{k+1,k}y_k\ \big\| \\ &= \big\|\ \|r_0\|\xi^1_{k+1} - F^HRy_k\ \big\| \\ &= \big\|\ \|r_0\|F\xi^1_{k+1} - Ry_k\ \big\| \end{aligned} \tag{2.30}$$

and $\|r_0\|F\xi^1_{k+1} - Ry_k$ is zero except for its last entry, which is the last entry of $\|r_0\|F\xi^1_{k+1}$

(since the last row of R is zero). Hence the last entry of $\|r_0\|F\xi^1_{k+1}$ is the 2-norm of the residual at step k [14].

2.3.5 The GMRES Method (Full Algorithm)

Having covered the solution method of the least squares problem, we now have enough knowledge to fully state the GMRES algorithm:

```
Algorithm: Full GMRES
Given an initial $x_0$:
    compute $r_0 = b - Ax_0$
    set $q_1 = \frac{r_0}{\|r_0\|}$
    initialise $\xi = (1, 0, \dots, 0)^T$
for $k = 1, 2, \dots$ do
    Compute $q_{k+1}$ and $h_{i,k} \equiv H(i,k), i = 1, \dots, k+1$
        using the Arnoldi Algorithm

    Apply $\mathcal{G}_1, \dots, \mathcal{G}_{k-1}$ to the last column of $H$, i.e.:
    for $i = 1, \dots, k-1$ do
        $\begin{pmatrix} H(i,k) \\ H(i+1,k) \end{pmatrix} \leftarrow \begin{pmatrix} c_i & s_i \\ -\bar{s}_i & c_i \end{pmatrix} \begin{pmatrix} H(i,k) \\ H(i+1,k) \end{pmatrix}$
    end do

    Compute $c_k$ and $s_k$ for the $k$th rotation to annihilate the
    $(k+1,k)$th entry of $H$. This can be via:
        $c_k = \frac{|H(k,k)|}{\sqrt{|H(k,k)|^2 + |H(k+1,k)|^2}}$,
        $\bar{s}_k = \frac{c_k H(k+1,k)}{H(k,k)}$,
    however a more efficient implementation should be used [3].

    Apply the $k$th rotation to $\xi$ and the last column of $H$:
        $\begin{pmatrix} \xi(k) \\ \xi(k+1) \end{pmatrix} \leftarrow \begin{pmatrix} c_i & s_i \\ -\bar{s}_i & c_i \end{pmatrix} \begin{pmatrix} \xi(k) \\ 0 \end{pmatrix}$
                $H(k,k) \leftarrow c_k H(k,k) + s_k H(k+1,k)$,      .
              $H(k+1,k) \leftarrow 0$

    if residual norm estimate, $\|r_0\||\xi(k+1)|$, is small enough then
        Solve the UT system: $H_{k\times k} y_k = \|r_0\|\xi_{k\times 1}$
        Compute: $x_k = x_0 + Q_k y_k$
    end if
end for
```

The full GMRES method requires a large amount of storage and work, which increases at each iteration. If the number of iterations needed to reach a solution for a given linear system becomes too high, this method may become impractical. The *restarted GMRES* method, or *GMRES(j)*, simply restarts full GMRES every j iterations, using the most recent iterate as an initial guess in the next cycle. This cuts down on storage requirements for solutions requiring many iterations and can make GMRES a practical solution method once more. In chapter 7 we look at an example problem to which both full GMRES and GMRES(j) are applied.

The convergence properties of the full and the restarted GMRES algorithms are more complicated, and less conclusive, than for the SI and OM-1 methods. A discussion of these can be found in chapter 3 of [3] and in [15]. Further information on the GMRES and restarted GMRES methods can be found in [3], [15] and [14].

CHAPTER 3

DISCRETE ORDINATES

3.1 Problem Setup

An important problem in nuclear engineering is that of achieving a stable sustained fission reaction within a nuclear reactor. A source is initially used to start the reaction by releasing neutrons via radioactive decay. The reaction then proceeds as these neutrons collide with each other within the reactor. Upon collision a neutron is either captured, scattered or causes a *fission event.* Any of these outcomes causes the collided neutron to travel in a different direction, or with a different energy and so it is no longer a part of the considered flux, and is referred to as *lost.* The probability of any kind of collision occurring is called the *total cross section*, denoted σ_t. Within this, the probability of a neutron being captured, scattered or causing a fission event upon collision are referred to as the *capture*, *scatter* or *fission cross sections* respectively. These are similarly denoted σ_c, σ_s and σ_f. A fission event occurs when a neutron hits a nucleus and causes a reaction which releases more neutrons, travelling in any direction. In the problem we will consider they are the only type of collision that add neutrons to the considered flux, and thus allow for the progression of the reaction, stable or unstable. In reality, since the neutrons produced by fission travel in any direction, neutrons outside the considered flux could cause events which add neutrons to the considered flux. Stability, or *criticality*, is achieved when the number of neutrons produced equals the number lost [1]. We can represent this problem via the *time-independent neutron transport equation* with no non-fission neutron contribution. Considering the time-independent equation allows us to look at a settled system in steady-state, and thus hopefully understand better how to achieve this state. In operator form, this can be written as

$$\mathcal{T}\Psi = \mathcal{S}\Psi + \mathcal{F}\Psi, \tag{3.1}$$

where the operators $\mathcal{T}$, $\mathcal{S}$ and $\mathcal{F}$ describe transport, scattering and fission respectively and $\Psi \equiv \Psi(\mathbf{r}, E, \mathbf{\Omega})$ is the flux of neutrons per unit volume, with energy $E \in \mathbb{R}^+$ at position $\mathbf{r} \in \mathbb{R}^3$ in direction $\mathbf{\Omega} \in \mathbb{S}^2$ (the unit sphere). To measure how close the reactor is to a critical state, this problem can be formulated as an eigenvalue problem by introducing a scalar $\lambda > 0$

$$(\mathcal{T} - \mathcal{S})\,\Psi = \lambda \mathcal{F} \Psi. \tag{3.2}$$

Equation (3.2) is known as the *criticality problem* and its smallest positive real eigenvalue, or *principal* eigenvalue, gives information concerning the criticality of the reaction. Specifically, if $\lambda > 1$ then the reaction is *subcritical* with more neutrons required to achieve a critical state, and if $\lambda < 1$ then it is *supercritical* with less neutrons required [1]. As a result, this problem can be used by nuclear engineers to optimise nuclear reactors.

When using iterative methods to solve (3.2), it is necessary to repeatedly solve *shift invert* problems of the form

$$(\mathcal{T} - \mathcal{S} - \alpha\mathcal{F})\,\Psi = b, \tag{3.3}$$

where α is some shift, usually the most recent approximation to a required eigenvalue, and b is a known vector. These are equivalently a sequence of *source problems* [1] of the form

$$(\mathcal{T} - \mathcal{S} - \mathcal{F})\,\Psi = \mathcal{Q}, \tag{3.4}$$

where $\mathcal{Q} \equiv \mathcal{Q}(\mathbf{r}, \mathbf{\Omega})$ is a known non-fission source term of neutrons. Consequently being able to efficiently solve such problems is of great importance. This chapter is dedicated to one such method known as the method of *discrete ordinates*.

Considering just one energy group with isotropic scattering, writing the source transport problem in full form gives [1]

$$\begin{gathered}\mathbf{\Omega} \cdot \nabla\Psi(\mathbf{r}, \mathbf{\Omega}) + \sigma_t(\mathbf{r})\,\Psi(\mathbf{r}, \mathbf{\Omega}) - (\sigma_s(\mathbf{r}) + \nu\sigma_f(\mathbf{r}))\,\phi = \mathcal{Q},\\ \phi \equiv \frac{1}{4\pi}\int_{\mathbb{S}^2} \Psi(\mathbf{r}, \mathbf{\Omega}')\, \mathrm{d}\mathbf{\Omega}'.\end{gathered} \tag{3.5}$$

where $\sigma_t(\mathbf{r})$ is the known total cross section, $\sigma_s(\mathbf{r})$ the known scatter cross section and $\sigma_f(\mathbf{r})$ the known fission cross section, each at position $\mathbf{r}$. Consider standard polar coordinates on $\mathbb{S}^2$, denoting by μ the contribution of $\mathbf{\Omega}$ in the z-direction (see figure 3-1).

Now, reducing to one dimension and combining the scatter and fission cross sections for convenience gives

$$\begin{gathered}\mu\frac{\partial\Psi\left(x,\mu\right)}{\partial x}+\sigma_t\Psi\left(x,\mu\right)-\sigma_s\phi=q,\quad \mu\in\left[-1,1\right],\\ \phi\left(x\right)\equiv\frac{1}{2}\int_{-1}^{1}\Psi\left(x,\mu'\right)\,\mathrm{d}\mu'.\end{gathered}\tag{3.6}$$

Here the angular domain, $\mathbf{\Omega}$, is now in the unit 'sphere' in 1D and is parametrised by $\mu = \cos(\theta), \theta \in [0,\pi]$ through spherical polar coordinates (see figure 3-1), i.e. $\mu \in [-1,1]$. Also x now denotes the spatial position instead of $\mathbf{r}$ since we have moved into a 1D spatial domain, with $x \in [a,b]$.

In order to solve this equation we also require boundary conditions. In this chapter we will consider two different types: *Dirichlet* and *reflective*. For Dirichlet boundary conditions the flux, Ψ, is set to a constant on the upper and lower ends of the spatial domain, for $-1 \le \mu < 0$ and $0 < \mu \le 1$ respectively. In this case, the system (3.6) includes the conditions

$$\begin{gathered}\Psi\left(b,\mu\right)=\Psi_b\left(\mu\right),\quad -1\le\mu<0,\\ \Psi\left(a,\mu\right)=\Psi_a\left(\mu\right),\quad 0<\mu\le 1.\end{gathered}\tag{3.7}$$

Dirichlet conditions are known as *vacuum* boundary conditions at either end if $\Psi_b = 0 = \Psi_a$. Considering a reactor under this condition would allow for no neutrons to enter the reactor from outside.

For reflective boundary conditions the incoming flux for angle μ is the same as the

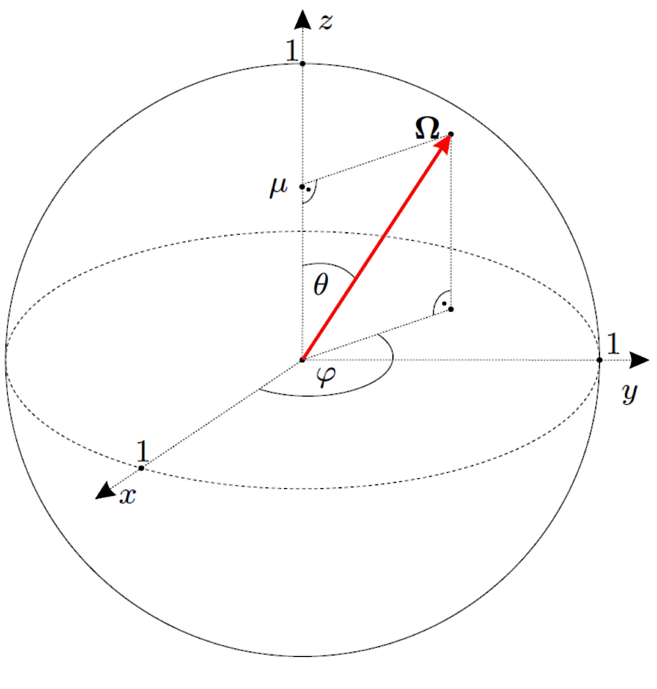

Figure 3-1: *Standard polar coordinates on $\mathbb{S}^2$, showing the contribution of $\mathbf{\Omega}$ in the z-direction.*

outgoing flux for angle $-\mu$. In this case, the system (3.6) includes the conditions

$$\begin{aligned} \Psi(b,\mu) &= \Psi(b,-\mu), \quad -1 \le \mu < 0, \\ \Psi(a,\mu) &= \Psi(a,-\mu), \quad 0 < \mu \le 1. \end{aligned} \tag{3.8}$$

This enforces the condition that no neutrons are lost from within the reactor, or scattered upon impact with the boundary.

When formulating the problem in matrix format we will merge these two types into one general boundary condition, allowing for the application of either (or both) types of condition.

3.2 The Method of Discrete Ordinates

One approach to solving the system (3.6) together with boundary conditions numerically is the method of *discrete ordinates.* It works by assuming that (3.6) holds only at discrete angles μ, here chosen to be Gauss quadrature points on $[-1,1]$. Then the integral definition of ϕ in (3.6) is replaced by a weighted Gauss quadrature sum. This leads to the system

$$\begin{aligned} \mu_j \frac{\partial \Psi_j}{\partial x} + \sigma_t \Psi_j - \sigma_s \phi = q_j, \quad j = 1, \ldots, n_\mu, \\ \phi \equiv \frac{1}{2} \sum_{k=1}^{n_\mu} w_k \Psi_k. \end{aligned} \tag{3.9}$$

In this expression μ_j are the quadrature points, w_j are the corresponding weights and Ψ_j denotes the approximation to $\Psi(x,\mu_j)$. It is assumed that there are an even number of quadrature points, n_μ, so that the points μ_j are all non-zero and are also symmetric about the origin. This allows for the introduction of reflective boundary conditions later on if required.

We then discretise (3.9) in space. One approach is to use a method known as *diamond differencing.* This involves replacing derivatives with respect to x with difference equations and setting function values at each node to be equal to the average of their values at the surrounding nodes. If we discretise the domain as follows

$$a \equiv x_0 < x_1 < \cdots < x_{n_x-1} < x_{n_x} \equiv b,$$

then we can replace (3.9) by

$$\mu_j \frac{\Psi_{i+1,j} - \Psi_{i,j}}{(\Delta x)_{i+1/2}} + \sigma_t \left(x_{i+1/2}\right) \frac{\Psi_{i+1,j} + \Psi_{i,j}}{2} - \sigma_s \left(x_{i+1/2}\right) \phi_{i+1/2} = q_{i+1/2,j},$$

$$j = 1, \ldots, n_\mu, \quad i = 0, \ldots, n_x - 1, \tag{3.10}$$

$$\phi_{i+1/2} \equiv \sum_{k=1}^{n_\mu} w_k \frac{\Psi_{i+1,k} + \Psi_{i,k}}{2},$$

where $(\Delta x)_{i+1/2} \equiv x_{i+1} - x_i$ and $x_{i+1/2} \equiv (x_{i+1} + x_i)/2$. Here we have $n_\mu n_x$ equations, but have $n_\mu(n_x + 1)$ unknowns. For this system to be solveable we require boundary conditions.

Under Dirichlet boundary conditions, this system also includes the equations

$$\begin{aligned} \Psi_{n_x,j} &= \Psi_b(\mu_j), \;\; j \le n_\mu/2, \\ \Psi_{0,j} &= \Psi_a(\mu_j), \;\; j > n_\mu/2, \end{aligned} \tag{3.11}$$

whereas under reflective boundary conditions it includes the equations

$$\begin{aligned} \Psi_{n_x,j} &= \Psi_{n_x,n_\mu-j+1}, \;\; j \le n_\mu/2, \\ \Psi_{0,j} &= \Psi_{0,n_\mu-j+1}, \;\; j > n_\mu/2. \end{aligned} \tag{3.12}$$

Alternatively a combination of these conditions may be used, for example Dirichlet conditions at the left end and reflective conditions at the right end of the domain. In writing (3.10) into matrix form we will allow for either of (3.11) or (3.12) to be used at each end. To achieve this generality, we note that these two types of condition may be written in the same form:

$$\begin{aligned} \Psi_{n_x,j} + \gamma_j \Psi_{n_x,n_\mu-j+1} &= \Psi_b(\mu_j), \quad j \le n_\mu/2, \\ \Psi_{0,j} + \gamma_j \Psi_{0,n_\mu-j+1} &= \Psi_a(\mu_j), \quad j > n_\mu/2. \end{aligned} \tag{3.13}$$

Where

$$\gamma_j = 0 \qquad \text{for Dirichlet boundary conditions,}$$

$$\left.\begin{aligned} \gamma_j &= -1 \\ \Psi_b(\mu_j), \Psi_a(\mu_j) &= 0 \end{aligned}\right\} \text{ for reflective boundary conditions.}$$

Thus the full system to be solved consists of (3.10) together with (3.13). Since (3.13) consists of n_μ equations, combining this with (3.10) means we have the same number of equations as unknowns, and thus a square system.

3.3 Matrix Formulation

In order to solve the above system, it would be useful to be able to represent it in matrix form as in equation (1.1) so that well-known solution techniques may be used. To this end, we first look at the equations that are involved. We begin by defining

$$\begin{aligned} d_{i-1/2,j} &\equiv \frac{|\mu_j|}{(\Delta x)_{i-1/2}} + \frac{\sigma_t\left(x_{i-1/2}\right)}{2}, \\ e_{i-1/2,j} &\equiv \frac{-|\mu_j|}{(\Delta x)_{i-1/2}} + \frac{\sigma_t\left(x_{i-1/2}\right)}{2}, \end{aligned} \tag{3.14}$$

for $i = 1, \ldots, n_x$. This change in range for i is to simplify the index references when programming this method. Now (3.10) may be written as

$$\left.\begin{array}{rcl} d_{i-1/2,j}\Psi_{i-1,j} \;+\; e_{i-1/2,j}\Psi_{i,j} \;-\; \sigma_{s,i-1/2}\phi_{i-1/2} &=& f\left(x_{i-1/2},\mu_j\right) \\ \text{with boundary conditions: } \Psi_{n_x,j} \;+\; \gamma_j\Psi_{n_x,n_\mu-j+1} &=& \Psi_b\left(\mu_j\right) \end{array}\right\} \quad \text{for } j \le n_\mu/2,$$

$$\left.\begin{array}{rcl} e_{i-1/2,j}\Psi_{i-1,j} \;+\; d_{i-1/2,j}\Psi_{i,j} \;-\; \sigma_{s,i-1/2}\phi_{i-1/2} &=& f\left(x_{i-1/2},\mu_j\right) \\ \text{with boundary conditions: } \Psi_{0,j} \;+\; \gamma_j\Psi_{0,n_\mu-j+1} &=& \Psi_a\left(\mu_j\right) \end{array}\right\} \quad \text{for } j > n_\mu/2,$$

$$\frac{1}{2}\left\{\omega_0\left(\Psi_{i,0}+\Psi_{i-1,0}\right)+\cdots+\omega_{n_\mu}\left(\Psi_{i,n_\mu}+\Psi_{i-1,n_\mu}\right)\right\} = \phi_{i-1/2}, \tag{3.15}$$

where $\sigma_{s,i-1/2} \equiv \sigma_s\left(x_{i-1/2}\right)$.

In order to construct a matrix representation of (3.15) we begin by defining the following matrix H_j

$$H_j \equiv \begin{pmatrix} d_{1/2,j} & e_{1/2,j} & & \\ & \ddots & \ddots & \\ & & d_{n_x-1/2,j} & e_{n_x-1/2,j} \\ & & & 1 \end{pmatrix} \in \mathbb{R}^{n_x+1\times n_x+1}, \quad j \le n_\mu/2, \tag{3.16}$$

and the vector Ψ_j

$$\Psi_j \equiv \begin{pmatrix} \Psi_{0,j} \\ \vdots \\ \Psi_{n_x,j} \end{pmatrix} \in \mathbb{R}^{n_x+1}, \quad j = 1, \ldots, n_\mu. \tag{3.17}$$

Multiplication of H_j with Ψ_j yields the terms $d_{i-1/2,j}\Psi_{i-1,j} + e_{i-1/2,j}\Psi_{i,j}$ for $i = 1, \ldots, n_x$. These are the first two terms in (3.15) for $j \le n_\mu/2$. Further, the last line of H_j when multiplied by Ψ_j extracts the element $\Psi_{n_x,j}$ required by the boundary condition equation for $j \leq n_\mu/2$.

To obtain the third term, $\sigma_{s,i-1/2}\phi_{i-1/2}$ from (3.15) for $j \leq n_\mu/2$ we define the matrix $\Sigma_{s,j}$

$$\Sigma_{s,j} \equiv \begin{pmatrix} \sigma_{s,1/2} & & \\ & \ddots & \\ & & \sigma_{s,n_x-1/2} \\ & & 0 \end{pmatrix} \in \mathbb{R}^{n_x+1\times n_x+1}, \quad j \leq n_\mu/2, \tag{3.18}$$

and the vector $\boldsymbol{\phi}$

$$\boldsymbol{\phi} \equiv \begin{pmatrix} \phi_{1/2} \\ \vdots \\ \phi_{n_x-1/2} \end{pmatrix} \in \mathbb{R}^{n_x}. \tag{3.19}$$

Multiplying these two together results in $\sigma_{s,i-1/2}\phi_{i-1/2}$, as well as a zero term. The zero is included so that this matrix vector multiplication does not add anything to the boundary condition construction started above with H_j.

The last term required is the right hand side value $f\left(x_{i-1/2}, \mu_j\right)$. We define these in the following vector $\boldsymbol{f}_j$ for $j \leq n_\mu/2$

$$\boldsymbol{f}_j \equiv \begin{pmatrix} f\left(x_{1/2}, \mu_j\right) \\ \vdots \\ f\left(x_{n_x-1/2}, \mu_j\right) \\ \Psi_b\left(\mu_j\right) \end{pmatrix} \in \mathbb{R}^{n_x+1}, \quad \text{for } j \leq n_\mu/2. \tag{3.20}$$

Here we have included the term $\Psi_b\left(\mu_j\right)$ from the right hand side of the boundary condition expression for $j \leq n_\mu/2$. To explain this, we now combine the above definitions to form the following expression

$$\left(H_j \;\mid\; -\Sigma_j\right) \begin{pmatrix} \Psi_j \\ - \\ \boldsymbol{\phi} \end{pmatrix} = \boldsymbol{f}_j. \tag{3.21}$$

This expression contains all the terms from (3.15) for $j \leq n_\mu/2$, however the last line of the matrix vector multiplication produces $\Psi_{n_x,j} = \Psi_b\left(\mu_j\right)$. This is the boundary condition equation for $j \leq n_\mu/2$ with $\gamma_j = 0$, or in other words prescribes Dirichlet boundary conditions to the upper end of the domain. We will later show how nonzero γ_j values may be included, but first we extend our set of definitions to allow the equations in (3.15) for $j > n_\mu/2$ to be included. To this end, define

$$H_j \equiv \begin{pmatrix} 1 & & & & \\ e_{1/2,j} & d_{1/2,j} & & & \\ & \ddots & \ddots & & \\ & & & e_{n_x-1/2,j} & d_{n_x-1/2,j} \end{pmatrix} \in \mathbb{R}^{n_x+1\times n_x+1}, \quad j > n_\mu/2, \tag{3.22}$$

and

$$\Sigma_{s,j} \equiv \begin{pmatrix} 0 & & & \\ \sigma_{s,1/2} & & & \\ & \ddots & & \\ & & & \sigma_{s,n_x-1/2} \end{pmatrix} \in \mathbb{R}^{n_x+1\times n_x+1}, \quad j > n_\mu/2, \tag{3.23}$$

along with the vectors

$$\boldsymbol{f}_j \equiv \begin{pmatrix} \Psi_a\left(\mu_j\right) \\ f\left(x_{1/2},\mu_j\right) \\ \vdots \\ f\left(x_{n_x-1/2},\mu_j\right) \end{pmatrix} \in \mathbb{R}^{n_x+1}, \quad \text{for } j > n_\mu/2. \tag{3.24}$$

Together with (3.17) and (3.19), these can be combined in the same way as (3.21) to give all the equations in (3.15) for $j > n_\mu/2$, as well as the corresponding boundary condition equations with $\gamma_j = 0$ once again.

Our next step is to extend the idea of (3.21) so that it includes all values of j in one expression. To achieve this, we include Ψ_j for $j = 1,\ldots,n_\mu$ in the left hand side vector as follows

$$\begin{pmatrix} \Psi_1 \\ \vdots \\ \Psi_{n_\mu} \\ - \\ \boldsymbol{\phi} \end{pmatrix} \in \mathbb{R}^{(n_\mu+1)(n_x+1)}.$$

Next, we place the matrices H_j in the coefficient matrix such that, if multiplied out, each one would be multiplied by the corresponding vector Ψ_j. Applying similar logic to the Σ_j and $\boldsymbol{f}_j$ we obtain the following extension of (3.21)

$$\begin{pmatrix} H_1 & & & \mid & -\Sigma_{s,1} \\ & \ddots & & \mid & \vdots \\ & & H_{n_\mu} & \mid & -\Sigma_{s,n_\mu} \end{pmatrix} \begin{pmatrix} \Psi_1 \\ \vdots \\ \Psi_{n_\mu} \\ - \\ \boldsymbol{\phi} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_1 \\ \vdots \\ \boldsymbol{f}_{n_\mu} \end{pmatrix}. \tag{3.25}$$

Now, in order to include the final line from equations (3.15) governing the definition of $\phi_{i-1/2}$, we now introduce a matrix S

$$S \equiv \frac{1}{2} \begin{pmatrix} 1 & 1 & & \\ & \ddots & \ddots & \\ & & 1 & 1 \end{pmatrix} \in \mathbb{R}^{n_x \times n_x+1}, \tag{3.26}$$

which averages nodal values to obtain the zone-centered values required by the diamond differencing method. Defining $\omega_j \equiv w_j/2$ so that $\sum_{j=1}^{n_\mu} \omega_j = 1$, we can obtain the full expression for $\phi_{i-1/2}$ given in (3.15) via

$$\begin{pmatrix} -\omega_1 S & \dots & -\omega_{n_\mu} S & \mid & I \end{pmatrix} \begin{pmatrix} \Psi_1 \\ \vdots \\ \Psi_{n_\mu} \\ - \\ \boldsymbol{\phi} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{3.27}$$

We are now in a position to state the full system in matrix form, albeit only allowing for Dirichlet boundary conditions. This is accomplished by combining (3.25) and (3.27) to get

$$\begin{pmatrix} H_1 & & & \mid & -\Sigma_{s,1} \\ & \ddots & & \mid & \vdots \\ & & H_{n_\mu} & \mid & -\Sigma_{s,n_\mu} \\ - & - & - & - & - \\ -\omega_1 S & \dots & -\omega_{n_\mu} S & \mid & I \end{pmatrix} \begin{pmatrix} \Psi_1 \\ \vdots \\ \Psi_{n_\mu} \\ - \\ \boldsymbol{\phi} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_1 \\ \vdots \\ \boldsymbol{f}_{n_\mu} \\ - \\ 0 \end{pmatrix}. \tag{3.28}$$

This is a square system of dimension $(n_\mu + 1)(n_x + 1)$. It is equivalent to the system obtained in [3] chapter 9, however we wish to extend this further to allow for the inclusion of reflecting boundary conditions. In order to do this we first define two matrices Γ_j

$$
\begin{aligned}
\Gamma_j &\equiv \begin{pmatrix} 0 & & & \\ & \ddots & & \\ & & 0 & \\ & & & \gamma_j \end{pmatrix} \in \mathbb{R}^{n_x+1 \times n_x+1}, \quad \text{for } j \leq n_\mu/2, \\
\Gamma_j &\equiv \begin{pmatrix} \gamma_j & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{pmatrix} \in \mathbb{R}^{n_x+1 \times n_x+1}, \quad \text{for } j > n_\mu/2.
\end{aligned}
\tag{3.29}
$$

Using these we can extract γ_j-multiples of the variables $\Psi_{n_x, n_\mu - j+1}$ for $j \leq n_\mu/2$ and $\Psi_{0, n_\mu - j+1}$ for $j > n_\mu/2$. By placing these matrices along the trailing diagonal of the coefficient matrix, we complete the boundary condition equations given in (3.15) for all j. The final matrix system we obtain is given by

$$
\begin{pmatrix}
H_1 & & \Gamma_1 & | & -\Sigma_{s,1} \\
& \ddots \quad \iddots & & & \\
& & & | & \vdots \\
& \iddots \quad \ddots & & & \\
\Gamma_{n_\mu} & & H_{n_\mu} & | & -\Sigma_{s,n_\mu} \\
- & - & - & - & - \\
-\omega_1 S & \dots & -\omega_{n_\mu} S & | & I
\end{pmatrix}
\begin{pmatrix} \Psi_1 \\ \vdots \\ \Psi_{n_\mu} \\ - \\ \boldsymbol{\phi} \end{pmatrix}
=
\begin{pmatrix} \boldsymbol{f}_1 \\ \vdots \\ \boldsymbol{f}_{n_\mu} \\ - \\ 0 \end{pmatrix}.
\tag{3.30}
$$

This is a solveable linear system of equations representing the Transport equation we originally hoped to solve, allowing for various different boundary conditions and cross sectional values. While this is solveable, it is a very large system of equations so may be inefficient to solve. We next look at a method of reducing the size of the system to possibly improve upon this efficiency.

3.4 The Schur Complement System

When working in high dimensions the angular flux vector $\boldsymbol{\phi}$ is very large. Because of this the above system becomes difficult to work with and instead the *Schur complement* system is constructed [16]. We will first define this system for a general matrix-vector problem.

Suppose we have a general block matrix system as follows

$$
\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} \Psi \\ \boldsymbol{\phi} \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix},
\tag{3.31}
$$

with A, B, C and D being $n \times n$, $n \times m$, $m \times n$ and $m \times m$ dimensional matrices respectively with A invertible. Also Ψ and f are vectors of length n and $\boldsymbol{\phi}$ and g vectors of length m. Then the Schur complement system is defined as

$$\left[D - CA^{-1}B\right] \boldsymbol{\phi} = g - CA^{-1}f. \tag{3.32}$$

This can be derived by carrying out block Gaussian elimination on (3.31). This is accomplished by multiplying through from the left by a block lower-triangular matrix

$$\begin{pmatrix} A^{-1} & 0 \\ -CA^{-1} & I \end{pmatrix}.$$

We proceed as follows

$$\begin{pmatrix} A^{-1} & 0 \\ -CA^{-1} & I \end{pmatrix} \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} \Psi \\ \boldsymbol{\phi} \end{pmatrix} = \begin{pmatrix} A^{-1} & 0 \\ -CA^{-1} & I \end{pmatrix} \begin{pmatrix} f \\ g \end{pmatrix}$$

$\Rightarrow$

$$\begin{pmatrix} I & A^{-1}B \\ 0 & D - CA^{-1}B \end{pmatrix} \begin{pmatrix} \Psi \\ \boldsymbol{\phi} \end{pmatrix} = \begin{pmatrix} A^{-1}f \\ g - CA^{-1}f \end{pmatrix}, \tag{3.33}$$

from which (3.32) can be seen. This system is often preferred to the original since it has a smaller condition number (see chapter 4), leading to a faster rate of convergence [16].

For the discrete ordinates system derived above, we have that

$$\begin{aligned}
A &= \begin{pmatrix} H_1 & & & \Gamma_1 \\ & \ddots & ⋰ & \\ & ⋰ & \ddots & \\ \Gamma_{n_\mu} & & & H_{n_\mu} \end{pmatrix}, \\
B &= \begin{pmatrix} -\Sigma_{s,1} \\ \vdots \\ -\Sigma_{s,n_\mu} \end{pmatrix}, \\
C &= \begin{pmatrix} -\omega_1 S \dots -\omega_{n_\mu} S \end{pmatrix}, \\
D &= I,
\end{aligned}$$

$$f = \begin{pmatrix} \boldsymbol{f}_1 \\ \vdots \\ \boldsymbol{f}_{n_\mu} \end{pmatrix},$$

$$g = 0.$$

This system is used in chapter 7 to ensure the convergence of simple iteration.

CHAPTER 4

PRECONDITIONING

4.1 Introduction to Preconditioning

This chapter focuses on the subject of improving the efficiency of iterative methods via a process called *preconditioning.* We consider again the basic linear system presented in chapter 1, namely

$$Ax = b, \tag{4.1}$$

where A is an $n \times n$ matrix, and x and b are n-dimensional vectorsn x unknown. Clearly in the case where the coefficient matrix is the identity matrix, any iterative scheme would converge to the correct solution in one iteration. Thus one way to improve the efficiency of finding a solution would be to multiply through by A^{-1}. Unfortunately finding this inverse is in itself a difficult problem. Instead it may be possible to multiply through by some easily calculable approximation to the inverse, preferably one for which matrix-vector multiplication is also straightforward. This can either be left or right multiplication. Firstly, if multiplying through from the left this process is known as *left preconditioning*, and leads to

$$M^{-1}Ax = M^{-1}b, \tag{4.2}$$

where M is known as the *preconditioner* of the system (4.1). Alternatively, multiplying through from the right this process is known as *right preconditioning*, and leads to

$$AM^{-1}\hat{x} = b, \quad \text{with } \hat{x} = Mx. \tag{4.3}$$

Both left and right preconditioned systems will converge to the same solution as the original system, provided M is nonsingular. Further, it is clear that the closer M^{-1} approximates the inverse of A (that is, the closer $M^{-1}A$ is to the identity) the faster the preconditioned system will converge. From this we can see that minimising the quantity $\|M^{-1}A\| \| \left(M^{-1}A\right)^{-1} \|$ will lead to a more easily solveable system. This quantity is know as the *condition number* and is denoted $\kappa\left(M^{-1}A\right)$. It is a commonly used value to judge how effective a particular preconditioner is. Generally when choosing a preconditioner there is a trade off between the effectiveness of the preconditioner and how easily calculable it is. Being too close to the true inverse of A would result in the preconditioner being too complex to compute. On the other hand, if it is chosen too far away from the inverse of A then any benefit of preconditioning will be so minimal that it is lost against the extra computation time involved.

4.2 An Example of Preconditioning

The iterative methods introduced in chapter 2 are all unpreconditioned and so typically converge slowly. Usually they are implemented on preconditioned systems, and in this section we will present the algorithm for simple iteration given in chapter 2 adapted to include a preconditioner, M. This is not a complicated process and effectively involves replacing the coefficient matrix A, and the right hand vector b, with their newly preconditioned versions. However it is a useful illustrative example and essentially provides the basis for a lot of advanced iterative methods such as *Multigrid methods* [3](Ch.12) [17]. We will work from the left preconditioned system (4.2) since this is more commonly used than right preconditioning.

Firstly, under the preconditioned system, the residual (2.2) becomes

$$r_k = M^{-1}\left(b - Ax_k\right). \qquad (4.4)$$

With this definition, the new approximation at each iteration is calculated via

$$x_{k+1} = x_k + r_k = x_k + M^{-1}\left(b - Ax_k\right), \qquad (4.5)$$

and so the algorithm becomes

```
**Algorithm: Preconditioned Simple Iteration**
For initial guess $x_0$:
    Compute: $r_0 = b - Ax_0$
    Solve: $Mz_0 = r_0$

for $k = 1, 2, \ldots$ do
        Set: $x_k = x_{k-1} + z_{k-1}$
        Compute: $r_k = b - Ax_k$
    Solve: $Mz_k = r_k$
end for
```

Since the true error of the method, (2.1), is unaffected by preconditioning, the results presented in chapter 2 relating to the convergence of simple iteration still hold.

CHAPTER 5

DERIVATION OF THE DIFFUSION APPROXIMATION

The aim of this chapter is to obtain an approximate version of the diffusion equation, known as the P_1 approximation. This approximation is used in preconditioning the Transport equation via Diffusion Synthetic Acceleration (see chapter 6). Before beginning this derivation however, we will first introduce *Legendre polynomials* and *spherical harmonics*.

5.1 Legendre Polynomials

The Legendre polynomials are defined by

$$\begin{aligned} &P_0(\mu) = 1 \\ &P_n(\mu) = \tfrac{1}{2^n n!}\tfrac{\mathrm{d}^n}{\mathrm{d}\mu^n}\left(\mu^2-1\right)^n, \quad n = 1, 2, \ldots. \end{aligned}$$

This is Rodrigues formula [18],[19]. The following *recurrence relation* is also satisfied by, or can define, the Legendre polynomials

$$\mu P_n(\mu) = \frac{n}{2n+1}P_{n-1}(\mu) + \frac{n+1}{2n+1}P_{n+1}(\mu), \quad \text{for } n = 1, 2, \ldots. \tag{5.1}$$

These $P_n(\mu)$ are a unique set of orthogonal polynomials [20]. This orthogonality property means that over the interval $-1 \leq \mu \leq 1$ they satisfy

$$\int_{-1}^{1} P_n(\mu)\, P_{\hat{n}}(\mu)\ \mathrm{d}\mu = 0, \quad \text{for } n \neq \hat{n}. \tag{5.2}$$

This can be seen by substituting (5.1) in to the left side of (5.2), and then using the 'by parts' method. In what follows we assume without loss of generality that $m < n$

$$\begin{aligned}\int_{-1}^{1} P_n(\mu) P_{\hat{n}}(\mu)\ \mathrm{d}\mu &= \frac{1}{2^{m+n} m! n!} \int_{-1}^{1} \frac{\mathrm{d}^m}{\mathrm{d}\mu^m}\left(\mu^2-1\right)^m \frac{\mathrm{d}^n}{\mathrm{d}\mu^n}\left(\mu^2-1\right)^n\ \mathrm{d}\mu,\\ &= \left[\frac{\mathrm{d}^{n-1}}{\mathrm{d}\mu^{n-1}}\left(\mu^2-1\right)^n \frac{\mathrm{d}^{m-1}}{\mathrm{d}\mu^{m-1}}\left(\mu^2-1\right)^m\right]\Bigg|_{-1}^{1} - \\ &\qquad \int_{-1}^{1} \frac{\mathrm{d}^{m+1}}{\mathrm{d}\mu^{m+1}}\left(\mu^2-1\right)^m \frac{\mathrm{d}^{n-1}}{\mathrm{d}\mu^{n-1}}\left(\mu^2-1\right)^n\ \mathrm{d}\mu.\end{aligned} \tag{5.3}$$

Now all the derivatives of $\left(\mu^2-1\right)^n$, up to the $(n-1)$th, have $\left(\mu^2-1\right)$ as a factor. Thus for $\mu = \pm 1$ they equal zero. Hence the boundary term in the last line of (5.3) equals zero and we may proceed treating just the integral term

$$\int_{-1}^{1} P_n(\mu) P_{\hat{n}}(\mu)\ \mathrm{d}\mu = \frac{1}{2^{m+n} m! n!} (-1)^n \int_{-1}^{1} \left(\mu^2-1\right)^n \frac{\mathrm{d}^{m+n}}{\mathrm{d}\mu^{m+n}}\left(\mu^2-1\right)^m\ \mathrm{d}\mu. \tag{5.4}$$

Finally, the highest order term in $\left(x^2-1\right)^m$ is of order $2m$, so since $n > m$ the differential must be zero, hence (5.2) holds [21].

It can further be derived from (5.1) that

$$\int_{-1}^{1} P_n(\mu) P_{\hat{n}}(\mu)\ \mathrm{d}\mu = \frac{2\delta_{n,\hat{n}}}{2n+1}, \tag{5.5}$$

where $\delta_{n,\hat{n}}$ is the *Kronecker delta* function defined as

$$\delta_{n,\hat{n}} \equiv \left\{\begin{array}{ll} 1, & n = \hat{n} \\ 0, & n \neq \hat{n} \end{array}\right. . \tag{5.6}$$

Hence, they have the normalisation

$$\int_{-1}^{1} \left(P_n(\mu)\right)^2\ \mathrm{d}\mu = \frac{2}{2n+1},$$

and once normalised they form a complete orthonormal sequence [21]. This means we can write any member $v \in [-1, 1]$ as

$$\begin{aligned} v &= \sum_{n=0}^{\infty} \frac{\langle v, P_n\rangle}{\langle P_n, P_n\rangle} P_n \\ &= \sum_{n=0}^{\infty} \frac{2n+1}{2} \langle v, P_n\rangle P_n. \end{aligned} \tag{5.7}$$

Consequently we can write the neutron flux, Ψ, as

$$\begin{aligned}\Psi(x,\mu) &= \sum_{n=0}^{\infty}\frac{2n+1}{2}\left(\int_{-1}^{1}\Psi(x,\mu')\,P_n(\mu')\,\mathrm{d}\mu'\right)P_n(\mu)\\ &= \sum_{n=0}^{\infty}(2n+1)\,\phi_n(x)\,P_n(\mu),\end{aligned} \tag{5.8}$$

where

$$\phi_n(x) \equiv \frac{1}{2}\int_{-1}^{1}\Psi(x,\mu')\,P_n(\mu')\,\mathrm{d}\mu'.$$

Lastly, the associated Legendre functions are defined by

$$P_n^m(x) = (-1)^m\left(1-x^2\right)^{m/2}\frac{\mathrm{d}^m}{\mathrm{d}x^m}\left(P_n(x)\right), \quad \text{for } m,n = 0,1,2\ldots. \tag{5.9}$$

5.2 Spherical Harmonics

Spherical harmonics are the angular portion of the solution to Laplace's equation

$$\nabla^2\psi = 0,$$

where ψ is a scalar function. Laplace's spherical harmonics, [22] [19], are a specific set of these, denoted Y_n^m, where m and n are integers, with $-n \leq m \leq n$ and defined as

$$Y_n^m(\theta,\varphi) \equiv a_n^m P_n^{|m|}(\cos(\theta))\,e^{im\varphi}, \tag{5.10}$$

where the $P_n^{|m|}$ are associated Legendre functions, and the a_n^m are normalisation constants. They are a product of trigonometric functions, which in this definition are included as a complex exponential function. With the correct constants, a, the Laplace spherical harmonics form a complete orthonormal sequence, and so form an orthonormal basis of the L^2, or square-integrable, functions. This means that any square-integrable function, f, can be written as a linear expansion

$$f(\theta,\varphi) \equiv \sum_{n=0}^{\infty}\sum_{m=-n}^{n} f_n^m Y_n^m(\theta,\varphi). \tag{5.11}$$

5.3 The P_N Approximation

Theorem 5.1:
The one dimensional, mono-energetic, steady-state, linear Boltzmannn Transport equation with zero fission in slab geometry with P_n scattering is given by

$$\mu\frac{\partial}{\partial x}\Psi\left(x,\mu\right)+\sigma_t\left(x\right)\Psi\left(x,\mu\right)=\sum_{n=0}^{N}\left(2n+1\right)\sigma_{s,n}\left(x\right)P_n\left(\mu\right)\phi_n\left(x\right)+q\left(x,\mu\right).$$
$$\phi_n\left(x\right)\equiv\frac{1}{2}\int_{-1}^{1}\Psi\left(x,\mu'\right)P_n\left(\mu'\right)\ d\mu' \tag{5.12}$$

where

$$\sigma_{s,n}\left(x\right)\equiv 2\pi\int_{-1}^{1}\sigma_s\left(x,\hat{\mu}\right)P_n\left(\hat{\mu}\right)\ d\hat{\mu}. \tag{5.13}$$

Proof.
We work from the 3D Boltzmann transport equation as given in [5]

$$\Omega\cdot\nabla\Psi\left(r,\Omega\right)+\sigma_t\left(r\right)\Psi\left(r,\Omega\right)=\int_{\mathbb{S}^2}\Psi\left(r,\Omega'\right)\sigma_s\left(r,\Omega\cdot\Omega'\right)\ \mathrm{d}\Omega'+q\left(r,\Omega\right). \tag{5.14}$$

This was also given in chapter 1. We expand the flux in spherical harmonics (as defined in section 5.2) to obtain

$$\Psi\left(r,\Omega\right)=\sum_{n=0}^{\infty}\sum_{m=-n}^{n}\phi_n^m\left(r\right)Y_n^m\left(\Omega\right). \tag{5.15}$$

In this, Y_n^m is a spherical harmonic defined by

$$Y_n^m\left(\Omega\right)\equiv a_n^m P_n^{|m|}\left(\mu\right)e^{im\varphi}, \tag{5.16}$$

where $P_n^{|m|}$ is the associated Legendre function and $\Omega=(\sin\theta\cos\varphi,\sin\theta\sin\varphi,\cos\theta)$ with $\mu=\cos\theta$. The constants a_n^m are present to scale the surface harmonics to have unit norm in $L_2\left(\mathbb{S}^2\right)$. This simplifies the expansion by avoiding weighting terms, and they are defined by

$$a_n^m\equiv\left(\frac{\left(2n+1\right)\left(n-|m|\right)!}{2\left(1+\delta_{m,0}\right)\pi\left(n+|m|\right)!}\right)^{1/2}, \tag{5.17}$$

where $\delta_{a,b}$ is again the Kronecker delta function (5.6). Further

$$\phi_n^m\left(r\right)\equiv\int_{\mathbb{S}^2}\Psi\left(r,\Omega\right)Y_n^m\left(\Omega\right)\ \mathrm{d}\Omega \tag{5.18}$$

is the (n,m)th moment of Ψ.

Now, we reduce $\sigma_s\left(r,\Omega\cdot\Omega'\right)$ to 1D cartesian coordinates as in chapter 3 equation (3.6) and expand via Legendre polynomials, in the same way as (5.15). This gives

$$\sigma_{s,n}(r) \equiv 2\pi \int_{-1}^{1} \sigma_s(r,\hat{\mu}) P_n(\hat{\mu}) \, \mathrm{d}\hat{\mu}, \tag{5.19}$$

where $\hat{\mu} \equiv \cos\theta$, the cosine of the scattering angle. With Ψ in the form of (5.15) we can use this to rewrite the scattering integral in (5.14) as

$$\int_{\mathbb{S}^2} \Psi(r,\Omega') \sigma_s(r,\Omega\cdot\Omega') \, \mathrm{d}\Omega' = \sum_{n=0}^{\infty} \sigma_{s,n}(r) \sum_{m=-n}^{n} \phi_n^m(r) Y_n^m(\Omega). \tag{5.20}$$

Since we are aiming for a 1D equation we do not need to work with surface integrals on the unit sphere so we can reduce the scattering to only one angular direction. This effectively means we set $m = 0$, so we are working just with Legendre polynomials and the coefficients a_n^m become

$$a_n^0 \equiv \frac{1}{2}\sqrt{\frac{2n+1}{\pi}}. \tag{5.21}$$

Thus

$$Y_n^0(\Omega) = \frac{1}{2}\sqrt{\frac{2n+1}{\pi}} P_n(\mu) e^0, \tag{5.22}$$

and so

$$\phi_n^0(r) \equiv \int_{\mathbb{S}^2} \Psi(r,\Omega) \frac{1}{2}\sqrt{\frac{2n+1}{\pi}} P_n(\mu) \, \mathrm{d}\Omega. \tag{5.23}$$

Substituting (5.22) and (5.23) into the right hand side of (5.20) with $m = 0$ gives

$$\begin{aligned} &\sum_{n=0}^{\infty} \sigma_{s,n}(r) \frac{1}{2}\sqrt{\frac{2n+1}{\pi}} P_n(\mu) \int_{\mathbb{S}^2} \Psi(r,\Omega) \frac{1}{2}\sqrt{\frac{2n+1}{\pi}} P_n(\mu) \, \mathrm{d}\Omega \\ &\equiv \sum_{n=0}^{\infty} \sigma_{s,n}(r) P_n(\mu)(2n+1) \frac{1}{4\pi} \int_{\mathbb{S}^2} \Psi(r,\Omega) P_n(\mu) \, \mathrm{d}\Omega \end{aligned} \tag{5.24}$$

The using polar coordinates as in chapter 3 we can simplify this integral to

$$\sum_{n=0}^{\infty} (2n+1) \sigma_{s,n}(r) P_n(\mu) \frac{1}{2} \int_{-1}^{1} \Psi(r,\mu) P_n(\mu) \, \mathrm{d}\mu. \tag{5.25}$$

This final expression is the 1D scattering term using P_n expansion for the linear Boltzmann Transport equation. If we limit the summation to a finite upper value of $n = N$ and insert this into the Transport equation, we get the required equation. □

Theorem 5.2:
The P_N, or Legendre, approximation in slab geometry for the Transport equation, con-

sisting of $N+1$ coupled differential equations is given by

$$\frac{n}{2n+1}\frac{d\phi_{n-1}}{dx}(x) \;+\; \frac{n+1}{2n+1}\frac{d\phi_{n+1}}{dx}(x) \;+\; (\sigma_t(x)-\sigma_n(x))\,\phi_n(x) \;=\; q_n \qquad \textit{for} \quad n=0,\dots,N-1$$

and

$$\frac{N}{2N+1}\frac{d\phi_{N-1}}{dx}(x) \;+\; (\sigma_t-\sigma_N)\,\phi_N \;=\; q_N, \tag{5.26}$$

where $q_n \equiv \frac{1}{2}\int_{-1}^{1} P_n(\mu)\,q(x,\mu)\;d\mu$ *and we denote* $\sigma_{s,n}\equiv\sigma_n$.

Proof. ([6] Appendix D)
Equation (5.8) in section 5.1 shows how the neutron flux can be written as an infinite series expansion. If we substitute this into (5.12) we obtain

$$\begin{aligned}\mu\frac{\partial}{\partial x}\left(\sum_{n=0}^{\infty}(2n+1)\,P_n(\mu)\,\phi_n(x)\right)+\sigma_t(x)\left(\sum_{n=0}^{\infty}(2n+1)\,P_n(\mu)\,\phi_n(x)\right)\\ =\sum_{n=0}^{\infty}(2n+1)\,P_n(\mu)\,\sigma_n(x)\,\phi_n(x)+q(x,\mu).\end{aligned} \tag{5.27}$$

Now, premultiplying by $\frac{1}{2}P_{\hat{n}}(\mu)$ and integrating over $-1\le\mu\le 1$ yields

$$\begin{aligned}&\sum_{n=0}^{\infty}\frac{(2n+1)}{2}\int_{-1}^{1}\mu P_{\hat{n}}(\mu)\,P_n(\mu)\,\frac{\mathrm{d}\phi_n}{\mathrm{d}x}(x)\;\mathrm{d}\mu\\ &\qquad+\quad\sigma_t(x)\sum_{n=0}^{\infty}\frac{(2n+1)}{2}\int_{-1}^{1}P_{\hat{n}}(\mu)\,P_n(\mu)\,\phi_n(x)\;\mathrm{d}\mu\\ &=\quad\sum_{n=0}^{\infty}\frac{(2n+1)}{2}\int_{-1}^{1}P_{\hat{n}}(\mu)\,P_n(\mu)\,\sigma_n(x)\,\phi_n(x)\;\mathrm{d}\mu\\ &\qquad+\quad\frac{1}{2}\int_{-1}^{1}P_{\hat{n}}(\mu)\,q(x,\mu)\;\mathrm{d}\mu.\end{aligned} \tag{5.28}$$

This looks quite complicated, however by applying the orthogonality of the Legendre polynomials, (5.2), the summation can be simplified in both the collision and scattering terms. To simplify notation, we also define $q_{\hat{n}}\equiv\frac{1}{2}\int_{-1}^{1}P_{\hat{n}}(\mu)\,q(x,\mu)\;\mathrm{d}\mu$. This gives

$$\begin{aligned}&\sum_{n=0}^{\infty}\frac{(2n+1)}{2}\int_{-1}^{1}\mu P_{\hat{n}}(\mu)\,P_n(\mu)\,\frac{\mathrm{d}\phi_n}{\mathrm{d}x}(x)\;\mathrm{d}\mu\\ &\qquad+\quad\left[\frac{2\hat{n}+1}{2\hat{n}+1}\right]\sigma_t(x)\,\phi_{\hat{n}}(x)\\ &=\quad\left[\frac{2\hat{n}+1}{2\hat{n}+1}\right]\sigma_{\hat{n}}(x)\,\phi_{\hat{n}}(x)\quad+\quad q_{\hat{n}}.\end{aligned} \tag{5.29}$$

It is harder to simplify the remaining summation due to the μ multiple within the integral. To move on this must be written solely in terms of Legendre polynomials so that the property of orthogonality may once more be applied. This is achieved through the use of the recurrence relation (5.1) defined in section 5.1. Substituting this in to the summation term on the left of (5.29) we get

$$\begin{aligned}\sum_{n=0}^{\infty}(2n+1)\frac{\mathrm{d}\phi_n}{\mathrm{d}x}(x)\Bigg\{\frac{1}{2}\int_{-1}^{1}\frac{\hat{n}}{2\hat{n}+1}P_{\hat{n}-1}(\mu)\,P_n(\mu)\ \mathrm{d}\mu \\ +\ \frac{1}{2}\int_{-1}^{1}\frac{\hat{n}+1}{2\hat{n}+1}P_{\hat{n}+1}(\mu)\,P_n(\mu)\ \mathrm{d}\mu\Bigg\},\end{aligned} \tag{5.30}$$

which using the orthogonality of Legendre polynomials once more becomes

$$\left[\frac{2\hat{n}-1}{2\hat{n}-1}\right]\left(\frac{\hat{n}}{2\hat{n}+1}\right)\frac{\mathrm{d}\phi_{\hat{n}-1}}{\mathrm{d}x}(x)\ +\ \left[\frac{2\hat{n}+3}{2\hat{n}+3}\right]\left(\frac{\hat{n}+1}{2\hat{n}+1}\right)\frac{\mathrm{d}\phi_{\hat{n}+1}}{\mathrm{d}x}(x)\,. \tag{5.31}$$

Substituting (5.31) into (5.29) gives

$$\begin{aligned}\left(\frac{\hat{n}}{2\hat{n}+1}\right)\frac{\mathrm{d}\phi_{\hat{n}-1}}{\mathrm{d}x}(x)\ +\ \left(\frac{\hat{n}+1}{2\hat{n}+1}\right)\frac{\mathrm{d}\phi_{\hat{n}+1}}{\mathrm{d}x}(x) \\ +\ \sigma_t(x)\,\phi_{\hat{n}}(x)\ =\ \sigma_{\hat{n}}(x)\,\phi_{\hat{n}}(x)\ +\ q_{\hat{n}}.\end{aligned} \tag{5.32}$$

From this we can obtain the desired system of $N+1$ coupled 1st-order differential equations by setting $\phi_{N+1}=0$. □

Finally, to move from P_N to P_1 we take $N=1$ in (5.26), leaving the following two coupled differential equations:

$$\frac{\mathrm{d}\phi_1}{\mathrm{d}x}\ +\ (\sigma_t-\sigma_0)\,\phi_0\ =\ q_0, \tag{5.33}$$

$$\frac{1}{3}\frac{\mathrm{d}\phi_0}{\mathrm{d}x}\ +\ (\sigma_t-\sigma_1)\,\phi_1\ =\ q_1. \tag{5.34}$$

To eliminate ϕ_1 we let $\sigma_1=0$ and rearrange, so (5.34) becomes

$$\phi_1\ =\ \frac{1}{\sigma_t}q_1\ -\ \frac{1}{3\sigma_t}\frac{\mathrm{d}\phi_0}{\mathrm{d}x}, \tag{5.35}$$

which when substituted into (5.33) results in

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{1}{3\sigma_t}\frac{\mathrm{d}\phi_0}{\mathrm{d}x}\right)\ +\ \sigma_t\left(1-\frac{\sigma_0}{\sigma_t}\right)\phi_0\ =\ q_0\ -\ \frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{q_1}{\sigma_t}\right). \tag{5.36}$$

The total cross section σ_t is defined as

$$\begin{aligned}\sigma_t(x) &\equiv \sigma_c(x)+\sigma_n(x) \\ &= \sigma_c(x)+2\pi\int_{-1}^{1}\sigma_s(r,\hat{\mu})\,P_n(\hat{\mu})\ \mathrm{d}\hat{\mu} \\ &= \sigma_c(x)+\sigma_0(x)\end{aligned} \tag{5.37}$$

which leaves

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{1}{3\sigma_t}\frac{\mathrm{d}\phi_0}{\mathrm{d}x}\right)\ +\ \sigma_c\phi_0\ =\ q_0\ -\ \frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{q_1}{\sigma_t}\right). \tag{5.38}$$

This is the approximation to the diffusion equation known as the P_1 approximation. For further information on where this approximation comes from see [23] [6] [5] and [7]. In chapter 6 we will see how it is used in preconditioning the Transport equation to enable a more efficient solution method (higher rate of convergence). This method is known as *Diffusion Synthetic Acceleration*, or *DSA*.

CHAPTER 6

DIFFUSION SYNTHETIC ACCELERATION (DSA)

In chapter 4 we saw that preconditioning a system can greatly reduce the number of iterations required for convergence. In this chapter we introduce an efficient preconditioner for the Transport equation, and demonstrate its effectiveness using an example. Firstly however we must cover a few preliminary concepts that are required.

6.1 Source Iteration

As before, the steady-state (time independent) Transport equation is given by

$$\mu\frac{\partial\Psi}{\partial x} + \sigma_t(x)\Psi(x,\mu) = \frac{1}{2}\left(\sigma_s(x) + \nu\sigma_f(x)\right)\int_{-1}^{1}\Psi \,\mathrm{d}\mu' + q(x), \tag{6.1}$$

with certain boundary conditions. The most basic iterative method for solving the Transport equation is *source iteration*, which is equivalent to simple iteration introduced in chapter 2 [3]. If we define

$$\begin{aligned} \mathcal{T}(\cdot) &\equiv \mu\frac{\partial\cdot}{\partial x} + \sigma_t(x)(\cdot), \\ \mathcal{S}(\cdot) &\equiv \frac{1}{2}\left(\sigma_s(x) + \nu\sigma_f(x)\right)\int_{-1}^{1}(\cdot)\,\mathrm{d}\mu', \end{aligned} \tag{6.2}$$

then (6.1) can be written as

$$\mathcal{T}\Psi = \mathcal{S}\Psi + q. \tag{6.3}$$

From this, source iteration can be defined as

$$\mathcal{T}\Psi^{(l+1)} = \mathcal{S}\Psi^{(l)} + q, \quad l \geq 0, \tag{6.4}$$

where $\Psi^{(0)}$ is an initial guess. An iterative step is carried out by substituting in an 'old' estimate for Ψ, say $\Psi^{(l)}$, and then solving the system for $\Psi^{(l+1)}$. This process is repeated until the difference between consecutive estimates is lower than some desired tolerance.

The following theorem proves the convergence of source iteration for the case where the cross sections are constant. This result could not be found upon searching known literature, and we include both its statement and proof here. It relies upon a result proved in [1].

Theorem 6.1:
If σ_s and σ_t are constant functions of x, then source iteration (6.4) *converges to the true solution Ψ as $l \to \infty$.*

Proof.
Let $E^{(l)} \equiv \Psi - \Psi^{(l)}$, where Ψ is the true solution to (6.4), then

$$\mathcal{T}E^{(l+1)} = \mathcal{S}E^{(l)}. \tag{6.5}$$

Defining $e^{(l)} \equiv \frac{1}{2}\int_{-1}^{1} E^{(l)}\,\mathrm{d}\mu$, then by the definition of $\mathcal{S}$ we have that

$$\mathcal{T}E^{(l+1)} = \sigma_s e^{(l)}. \tag{6.6}$$

Now, let $\mathcal{K}$ be the appropriate integral operator [1] such that

$$\begin{aligned} e^{(l+1)} &= \mathcal{K}\mathcal{T}E^{(l+1)} \\ &= \sigma_s\mathcal{K}e^{(l)} \\ &= (\sigma_s\mathcal{K})^2 e^{(l-1)} \\ &= \ldots \\ &= (\sigma_s\mathcal{K})^{l+1} e^{(0)}. \end{aligned} \tag{6.7}$$

Now, by taking norms we have that

$$\begin{aligned} \|e^{(l+1)}\| &\leq \|(\sigma_s\mathcal{K})^{l+1}\|\|e^{(0)}\| \\ &\leq \|\sigma_s\mathcal{K}\|^{l+1}\|e^{(0)}\|, \end{aligned} \tag{6.8}$$

and so we have that $\|e^{(l+1)}\| \to 0$ as $l \to \infty$ if $\|\sigma_s\mathcal{K}\| < 1$. By [1] (page 29 theorem 2.9) and the fact that $\sigma_t = \sigma_c + \sigma_s + \sigma_f$, we get that

$$\|\sigma_s\mathcal{K}\| \leq \frac{\sigma_s}{\sigma_t} < 1. \tag{6.9}$$

Thus $\|e^{(l+1)}\| \to 0$ and so $\|E^{(l+1)}\| \to 0$, hence $\Psi^{(l+1)} \to \Psi$ as $l \to \infty$ as required. $\square$

From this proof, it can be seen that the closer $\|\sigma_s \mathcal{K}\|$ is to zero, the faster the error decreases and so the faster the convergence. By the bound (6.9) this means that the smaller σ_s is compared to σ_t, the faster source iteration will converge. We will return to this property later in chapter 7.

This method converges rapidly for problems in small systems in which particles are likely to exit through the outer boundary after a few collisions ('leaky' systems), or in larger systems in which particles are likely to be captured after a few collisions. Unfortunately for systems with low probabilities of leakage or capture, most particles will collide many times before either leaking or being captured and source iteration converges very slowly [7]. These problems are of great practical importance, and so it would be very beneficial to be able to speed up, or *accelerate*, the convergence.

6.2 Synthetic Acceleration

The original idea of a *synthetic* acceleration scheme was introduced by Kopp [24] in 1962. It works by treating one source iteration as the first step of a two (or more) step process. To incorporate this into our notation, we shall denote by $\Psi^{(l+1/2)}$ the estimate obtained after one source iteration. Thus

$$\mathcal{T}\Psi^{(l+1/2)} = \mathcal{S}\Psi^{(l)} + q, \quad l \geq 0. \tag{6.10}$$

The next step is to obtain an estimate $\Psi^{(l+1)}$ that is a far better approximation to Ψ than $\Psi^{(l+1/2)}$ is. Under a synthetic scheme, this is achieved by subtracting (6.10) from (6.3):

$$\begin{aligned} \mathcal{T}\left(\Psi - \Psi^{(l+1/2)}\right) &= \mathcal{S}\left(\Psi - \Psi^{(l)}\right) \\ &= \mathcal{S}\left(\Psi - \Psi^{(l+1/2)}\right) + \mathcal{S}\left(\Psi^{(l+1/2)} - \Psi^{(l)}\right). \end{aligned} \tag{6.11}$$

From this an equation for the *additive correction* $\Psi - \Psi^{(l+1/2)}$ can be found as

$$\begin{aligned} (\mathcal{T} - \mathcal{S})\left(\Psi - \Psi^{(l+1/2)}\right) &= \mathcal{S}\left(\Psi^{(l+1/2)} - \Psi^{(l)}\right), \\ \Rightarrow \quad \Psi &= \Psi^{(l+1/2)} + (\mathcal{T} - \mathcal{S})^{-1}\mathcal{S}\left(\Psi^{(l+1/2)} - \Psi^{(l)}\right). \end{aligned} \tag{6.12}$$

This precisely defines the exact solution Ψ in terms of known variables, however it also involves the inverse of the transport operator $\mathcal{T} - \mathcal{S}$. If this could be found or approximated efficiently, then the solution could be found quickly and efficiently too. This is the idea of synthetic acceleration: to replace $(\mathcal{T} - \mathcal{S})^{-1}$ in (6.12) with an approximation M,

$$M \approx (\mathcal{T} - \mathcal{S})^{-1} \tag{6.13}$$

where $M\mathcal{S}\left(\Psi^{(l+1/2)} - \Psi^{(l)}\right)$ is easier to calculate than $(\mathcal{T} - \mathcal{S})^{-1}\,\mathcal{S}\left(\Psi^{(l+1/2)} - \Psi^{(l)}\right)$.

This allows the iterative scheme for the second step to be obtained from (6.12):

$$\Psi^{(l+1)} \;=\; \Psi^{(l+1/2)} + M\mathcal{S}\left(\Psi^{(l+1/2)} - \Psi^{(l)}\right). \tag{6.14}$$

The full synthetic scheme can be defined by (6.10) and (6.14). If the scheme converges, then the solution must satisfy the original Transport equation. To see this, we subtract $\Psi^{(l)}$ from both sides of (6.14)

$$\begin{aligned}\Psi^{(l+1)} - \Psi^{(l)} &= \Psi^{(l+1/2)} + M\mathcal{S}\left(\Psi^{(l+1/2)} - \Psi^{(l)}\right) - \Psi^{(l)},\\ \Psi^{(l+1)} - \Psi^{(l)} &= (M\mathcal{S} + I)\left(\Psi^{(l+1/2)} - \Psi^{(l)}\right).\end{aligned} \tag{6.15}$$

If the method has converged, then $\Psi^{l+1} - \Psi^{l} \approx 0$, so

$$0 = (M\mathcal{S} + 1)\left(\Psi^{(l+1/2)} - \Psi^{(l)}\right), \tag{6.16}$$

which implies that $\left(\Psi^{l+1/2} - \Psi^{l}\right) \approx 0$, thus the original Transport problem has converged also.

The convergence and efficiency of the scheme vary depending upon the choice of M. If M is very close to $(\mathcal{T} - \mathcal{S})^{-1}$ then (6.13) is almost exactly true and convergence will be very fast. However, there is a trade-off between M being closer to $(\mathcal{T} - \mathcal{S})^{-1}$ and M being easier to calculate. So a good choice of M should be as close as possible to $(\mathcal{T} - \mathcal{S})^{-1}$ while remaining easy to calculate.

6.2.1 Acceleration and Preconditioning

The concept of an acceleration scheme can in fact be shown to be equivalent to the concept of preconditioning, introduced in chapter 4. In fact, source iteration and synthetic acceleration schemes can be written as *Richardson* [25] and *preconditioned Richardson* [26] matrix iteration schemes, respectively. We can show this just using equations already defined in this section.

Applying the inverse of the operator $\mathcal{T}$ to (6.3) and (6.4) yields

$$\begin{aligned}\Psi &\equiv \mathcal{T}^{-1}\mathcal{S}\Psi + \mathcal{T}^{-1}q,\\ \Psi^{(l+1)} &\equiv \mathcal{T}^{-1}\mathcal{S}\Psi^{(l)} + \mathcal{T}^{-1}q.\end{aligned} \tag{6.17}$$

Defining $\mathcal{A} \equiv I - \mathcal{T}^{-1}\mathcal{S}$ and $\hat{q} \equiv \mathcal{T}^{-1}q$, we can write (6.17) as

$$\mathcal{A}\Psi \equiv \hat{q}, \tag{6.18}$$

$$\Psi^{(l+1)} \equiv (I - \mathcal{A})\,\Psi^{(l)} + \hat{q}. \tag{6.19}$$

(6.19) is a Richardson scheme for (6.18). Next we write (6.10) as

$$\Psi^{(l+1/2)} = (I - \mathcal{A})\,\Psi^{(l)} + \hat{q}, \tag{6.20}$$

and substitute this into (6.14) giving

$$\Psi^{(l+1)} = \Psi^{(l)} + (I + M\mathcal{S})\left(-\mathcal{A}\Psi^{(l)} + \hat{q}\right). \tag{6.21}$$

Defining $\mathcal{P} \equiv I + M\mathcal{S}$, (6.21) becomes

$$\Psi^{(l+1)} = (I - \mathcal{P}\mathcal{A})\,\Psi^{(l)} + \mathcal{P}\hat{q}. \tag{6.22}$$

This final equation, (6.22), is the preconditioned Richardson scheme for (6.18) with preconditioner $\mathcal{P}$.

6.3 Diffusion Synthetic Acceleration

We now have a sufficient basis to be able to describe the iterative process known as *Diffusion Synthetic Acceleration* or *DSA*. DSA is a type of synthetic acceleration method (as described in section 6.2) that uses the P_1 diffusion approximation as its preconditioner (see chapter 5). This is particularly appropriate because the first step source iteration of a synthetic acceleration method suppresses error modes that have strong angular and spatial dependence [7]. Therefore, the most efficient scheme would involve a preconditioner which suppresses error modes with *weak* angular and spatial dependence. This is a property satisfied by the diffusion approximation, making it a good candidate for an effective preconditioner.

For the completeness of this section, we will restate source iteration and then demonstrate the use of the P_1 diffusion approximation. Firstly, the source iteration is defined as

$$\mu\frac{\partial\Psi^{(l+1/2)}}{\partial x}(x,\mu) + \sigma_t(x)\Psi^{(l+1/2)}(x,\mu) = \frac{\sigma_s(x)}{2}\int_{-1}^{1}\Psi^{(l)}\left(x,\mu'\right)\,\mathrm{d}\mu' + q, \quad l \geq 0, \tag{6.23}$$

for $x \in [a,b]$ and $-1 \leq \mu \leq 1$, with boundary conditions

$$\Psi^{(l+1/2)}(b,\mu) = \Psi_b(\mu), \quad -1 \leq \mu < 0,$$
$$\Psi^{(l+1/2)}(a,\mu) = \Psi^{(l+1/2)}(a,-\mu), \quad 0 < \mu \leq 1.$$

Next, defining the linear correction as

$$\boldsymbol{e}^{(l+1/2)}(x,\mu) \equiv \Psi(x,\mu) - \Psi^{(l+1/2)}(x,\mu), \tag{6.24}$$

we can use (6.12) to obtain

$$\begin{aligned}\mu\frac{\partial \boldsymbol{e}^{(l+1/2)}}{\partial x}(x,\mu) + \sigma_t(x)\,\boldsymbol{e}^{(l+1/2)}(x,\mu) - \frac{\sigma_s(x)}{2}\int_{-1}^{1}\boldsymbol{e}^{(l+1/2)}\left(x,\mu'\right)\,\mathrm{d}\mu' \\ = \frac{\sigma_s(x)}{2}\int_{-1}^{1}\Psi^{(l+1/2)}\left(x,\mu'\right) - \Psi^{(l)}\left(x,\mu'\right)\,\mathrm{d}\mu',\end{aligned} \tag{6.25}$$

with boundary conditions

$$\begin{gathered}\boldsymbol{e}^{(l+1/2)}(b,\mu) = 0, \quad -1 \le \mu < 0, \\ \boldsymbol{e}^{(l+1/2)}(a,\mu) = \boldsymbol{e}^{(l+1/2)}(a,-\mu), \quad 0 < \mu \le 1.\end{gathered}$$

This is notationally equivalent to

$$(\mathcal{T} - \mathcal{S})\left(\boldsymbol{e}^{(l+1/2)}\right) = \mathcal{S}\left(\Psi^{l+1/2} - \Psi^{l}\right). \tag{6.26}$$

In section 6.2 we observed that solving this formulation was no less complex than the original problem, but that using an easily calculable *approximation* to $(\mathcal{T} - \mathcal{S})^{-1}$ could lead to an efficient solution. In the DSA scheme this is achieved via the P_1 diffusion approximation. In order to apply this approximation however, we need an arguement that is a function of spatial variable x only. To this end, we define

$$F^{(l+1)}(x) \approx \int_{-1}^{1}\boldsymbol{e}^{(l+1/2)}(x,\mu)\,\mathrm{d}\mu, \tag{6.27}$$

which is an approximation to the scalar flux component of $\boldsymbol{e}^{(l+1/2)}(x,\mu)$. For simplicity we also note that (5.12) implies

$$\phi_0^{(l)}(x) \equiv \frac{1}{2}\int_{-1}^{1}\Psi^{(l)}(x,\mu)\,\mathrm{d}\mu. \tag{6.28}$$

With these in hand, we can now apply the diffusion approximation to (6.25) giving

$$\begin{aligned}-\frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{1}{3\sigma_t}\frac{\mathrm{d}F^{(l+1)}}{\mathrm{d}x}(x)\right) + \sigma_c(x)\,F^{(l+1)}(x) \\ = \sigma_s(x)\left[\phi_0^{(l+1/2)}(x) - \phi_0^{(l)}(x)\right],\end{aligned} \tag{6.29}$$

with boundary conditions

$$F^{(l+1)}(b) - \frac{2}{3\sigma_t(b)}\frac{\mathrm{d}F^{(l+1)}}{\mathrm{d}x}(b) = 0,$$
$$\frac{\mathrm{d}F^{(l+1)}}{\mathrm{d}x}(a) = 0.$$

To obtain $\Psi^{(l+1)}$ we now integrate (6.24) over μ and rearrange as follows

$$\frac{1}{2}\int_{-1}^{1} \boldsymbol{e}^{(l+1/2)}(x,\mu)\ \mathrm{d}\mu = \phi_0(x) - \phi_0^{(l+1/2)}(x)$$

$\iff$

$$\phi_0^{(l+1)}(x) \approx \frac{1}{2}F^{(l+1)}(x) + \phi_0^{(l+1/2)}(x). \tag{6.30}$$

This final equation puts us in a position to fully define the DSA scheme using (6.23), (6.29) and (6.30):

At the start of an iteration $\phi_0^{(l)}$ is known, either from the previous iteration or with $l = 0$ for an initial guess.

The first step of the iteration (known as the *transport sweep*) is to substitute this into (6.23) and solve for $\Psi^{(l+1/2)}$, then integrate to get $\phi_0^{(l+1/2)}$.

In the second step of the iteration, (6.29) is solved for $F^{(l+1)}$, then (6.30) is used to obtain $\phi_0^{(l+1)}$ as required.

In chapter 7 we will explore how to apply this method computationally.

CHAPTER 7

IMPLEMENTATION OF ITERATIVE METHODS

In this chapter, we will describe how the discrete ordinates method for discretising the Transport equation defined in chapter 3 may be implemented computationally to set up a solveable system of linear equations. We will discuss briefly how each of the iterative techniques defined in chapter 2 can be applied to the system once it is set up, making particular reference to how the results in chapter 2 relating to convergence influence our choices. We will further focus on how the method of DSA, derived and defined in chapters 5 and 6, may be discretised and applied to the system as well. Finally, we will investigate the convergence of the various methods by applying them to a model problem given in A. Greenbaum [3]. We will examine the infinity norm of the error of the approximate solution given at each step by each method, and compare a plot of these values to those obtained by A. Greenbaum. Code to carry out this test is included for reference in appendix A, and was written to run in Matlab.

7.1 Discrete Ordinates Matrix Construction

Applying the method of discrete ordinates to solve the Transport equation as discussed in chapter 3 involves setting up a very large and sparse matrix (3.30). It would be beneficial to exploit the sparsity of this matrix in order to reduce the amount of storage required. To this end, we first look at the structure of the matrix. Figure 7-1 illustrates this structure by representing the matrix graphically, plotting blue points only where there are non-zero entries in the matrix. The matrix is heavily diagonal and the black lines show clearly that it can be naturally split into four ‘blocks’. This block format is also visible in the definition (3.30). In order to exploit the sparsity we will need to work on each block individually. By doing this we can store the non-zero diagonals for each block as vectors, whilst keeping track of where each diagonal should be placed. This is a very common technique for storing sparse diagonal matrices and there are built in functions in Matlab for handling such data.

These vectors could be found via the formulae in chapter 3. For each block we would obtain a vector of values for each non-zero diagonal, along with a vector of indices describing on which diagonals to place these values to obtain the desired matrix. Having achieved this, a Matlab function called `spdiags` can be used to generate each block as a fully formed sparse matrix. Finally concatenating these would yield the final matrix, but without the opposing coefficients described by the matrices Γ_j in (3.29). Now, each Γ_j only contains one non-zero entry, γ_j, so we can just focus on entering these values. To do this we need to know the specific coordinates of each γ_j. These can be found via

$$\begin{array}{lll} ((n_x+1)j & , (n_x+1)(n_\mu - j + 1)) & \text{for } j < n_\mu/2, \\ ((n_x+1)(j-1)+1 & , (n_x+1)(n_\mu - j) + 1) & \text{for } j > n_\mu/2. \end{array} \tag{7.1}$$

We can now fully construct the coefficient matrix defined in (3.30), so we have only to find the right hand side vector. This presents far fewer difficulties, and in fact can be calculated directly using formulae from chapter 3. With this, the discrete ordinates system is completely formed, and can be solved by an appropriate iterative technique.

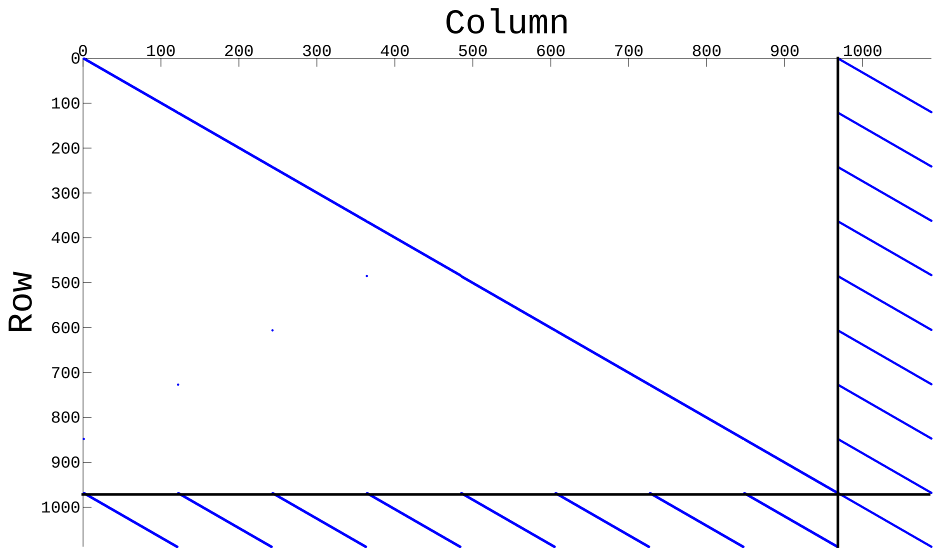

Figure 7-1: *Graphical representation of the coefficient matrix obtained using the method of discrete ordinates* (3.30)*. The upper end of the domain has a reflecting boundary condition and the lower end has a vacuum boundary condition.*

7.2 Implementing Iterative Solution Methods

In this section, we look briefly at how each of the methods defined in chapter 2 could be applied to the linear system found above. To begin with we look at simple iteration, then the Orthomin(1) method and finally GMRES.

For simple iteration (SI) the requirements are very straightforward with only an initial guess needed to get started. However, lemma 2.1 asserts that SI will converge to the correct solution if and only if

$$\lim_{k\to\infty} \| (I - A)^k \| = 0, \tag{7.2}$$

where A is the coefficient matrix. Unfortunately, due to the highly diagonal nature of the coefficient matrix obtained via the method of discrete ordinates, this condition fails and so simple iteration would not converge on the full system. This is not an issue if it is applied to the Schur complement system defined in chapter 3 section 3.4. The biggest difference is that this system solves for the scalar flux term $\boldsymbol{\phi}$, from which the angular flux can then be computed if required [3].

The Orthomin(1) method (OM-1) will converge if applied to the full system, and so does not need to use the Schur complement system. This is because the convergence condition of theorem 2.2 holds. The actual implementation of OM-1 is almost identical to that of SI, only needing one further vector calculation and storage per step.

To apply the GMRES method is far more complicated, requiring the calculation and storage of successive rotation matrices as well the use of Arnoldi's algorithm. Usually, the impracticality of the storage demands necessitates the use of the restarted GMRES method mentioned at the end of chapter 2. In either case we will not fully explore this application here since it is possible to use the built in Matlab function `gmres`, which gives an efficient implementation of GMRES. For more information regarding programming the GMRES method, see [13].

7.3 Implementing DSA

In this section we look at how to implement DSA, defined in chapter 6, in Matlab. We will work on applying it to SI, though most other iterative methods would also work with it. Since we are working with SI, we will once again be solving the Schur complement system in order to ensure convergence. Using the ideas from the previous section, we can carry out the transport sweep of DSA already. This leaves us with solving the diffusion approximation.

The first step is to discretise the diffusion approximation (6.29) so that it may be solved numerically. For simplicity, we restate (6.29) in the following general form

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left(c\left(x\right)\frac{\mathrm{d}F}{\mathrm{d}x}\right)+d\left(x\right)F\left(x\right)=g\left(x\right), \tag{7.3}$$

for $x \in [a,b]$ with boundary conditions

$$F\left(b\right)-e\left(b\right)\frac{\mathrm{d}F}{\mathrm{d}x}\left(b\right)=0, \tag{7.4}$$

$$\frac{\mathrm{d}F}{\mathrm{d}x}\left(a\right)=0. \tag{7.5}$$

This assumes vacuum boundary conditions on the upper end of the domain and reflecting conditions on the lower end. Since we will know the right hand side of this equation already from previous calculations, we focus on discretising the left. If we discretise the domain as $a \equiv x_0 < x_1 < \cdots < x_{n_x} \equiv b$, then standard Taylor expansion gives

$$\left\{-\left(cF'\right)'+dF\right\}\Big|_{x_i} \approx \frac{-1}{h}\left[\left(cF'\right)\left(x_{i+1/2}\right)-\left(cF'\right)\left(x_{i-1/2}\right)\right]+d\left(x_i\right)F\left(x_i\right). \tag{7.6}$$

for $i = 0,\ldots,n_x$, where $F' \equiv \frac{\mathrm{d}F}{\mathrm{d}x}$ and $h \equiv (b-a)/n_x$ is the width of each discrete interval. By a further expansion we can say that

$$\begin{aligned}\left(cF'\right)\left(x_{i+1/2}\right) &= c\left(x_{i+1/2}\right)F'\left(x_{i+1/2}\right)\\ &\approx c\left(x_{i+1/2}\right)\left(\frac{F\left(x_{i+1}\right)-F\left(x_i\right)}{h}\right),\\ \left(cF'\right)\left(x_{i-1/2}\right) &= c\left(x_{i-1/2}\right)F'\left(x_{i-1/2}\right)\\ &\approx c\left(x_{i-1/2}\right)\left(\frac{F\left(x_i\right)-F\left(x_{i-1}\right)}{h}\right).\end{aligned} \tag{7.7}$$

Substituting this into (7.6) gives that

$$\begin{aligned}\left\{-\left(cF'\right)'+dF\right\}\Big|_{x_i} \approx \frac{-1}{h^2}\Big[c\left(x_{i+1/2}\right)\left\{F\left(x_{i+1}\right)-F\left(x_i\right)\right\}-&\\ c\left(x_{i-1/2}\right)\left\{F\left(x_i\right)-F\left(x_{i-1}\right)\right\}\Big]+d\left(x_i\right)F\left(x_i\right),&\end{aligned} \tag{7.8}$$

thus

$$\begin{aligned}\frac{-1}{h^2}\Big[c\left(x_{i+1/2}\right)\left\{F\left(x_{i+1}\right)-F\left(x_i\right)\right\}-c\left(x_{i-1/2}\right)\left\{F\left(x_i\right)-F\left(x_{i-1}\right)\right\}\Big]+&\\ d\left(x_i\right)F\left(x_i\right)=g\left(x_i\right).&\end{aligned} \tag{7.9}$$

Now, for $i = 0$ or n_x this results in terms being evaluated at points outside of the domain. This means we cannot evaluate our system at the end points of the domain, leaving us with two less equations than unknowns. These points are often referred to as *ghost points* and can be eliminated by evaluating them at the boundary to establish a relationship between them and known points [27]. Using this relationship we can then solve (7.9) at the boundary to obtain a new equation for the system. So looking first at the lower end of the domain, via Taylor expansion on (7.5) we obtain

$$0 = F'(x_0) \approx \frac{F(x_1) - F(x_{-1})}{2h}, \tag{7.10}$$

which implies

$$F(x_{-1}) = F(x_1). \tag{7.11}$$

Substituting this into (7.9) gives

$$\begin{aligned}\frac{-1}{h^2}\left[c\left(x_{1/2}\right)\{F(x_1) - F(x_0)\} - c\left(x_{-1/2}\right)\{F(x_0) - F(x_1)\}\right] + \\ d(x_0)F(x_0) = g(x_0).\end{aligned} \tag{7.12}$$

This includes $c\left(x_{-1/2}\right)$ which is unknown, however this can be set equal to $c\left(x_{1/2}\right)$ without much loss of accuracy, resulting in

$$\frac{-2}{h^2}\,c\left(x_{1/2}\right)[F(x_1) - F(x_0)] + d(x_0)F(x_0) = g(x_0),$$

$\Rightarrow$

$$\left\{d(x_0) + \frac{2}{h^2}\,c\left(x_{1/2}\right)\right\}F(x_0) - \frac{2}{h^2}\,c\left(x_{1/2}\right)F(x_1) = g(x_0). \tag{7.13}$$

Thus we obtain one further equation from the lower boundary condition. Next, through Taylor expansion on (7.4) we have

$$F(x_{n_x}) - e(x_{n_x})\left[\frac{F(x_{n_x+1}) - F(x_{n_x-1})}{2h}\right] = 0. \tag{7.14}$$

Rearranging gives us

$$F(x_{n_x+1}) = F(x_{n_x-1}) + \frac{2h}{e(x_{n_x})}F(x_{n_x}), \tag{7.15}$$

which as before, we substitute into (7.9). Setting $c\left(x_{n_x+1/2}\right) = c\left(x_{n_x-1/2}\right)$ and rearranging yields

$$\left[\left(\frac{1}{h}-\frac{1}{e\left(x_{n_x}\right)}\right)\frac{2}{h}c\left(x_{n_x-1/2}\right)+d\left(x_{n_x}\right)\right]F\left(x_{n_x}\right)-\frac{2}{h^2}c\left(x_{n_x-1/2}\right)F\left(x_{n_x-1}\right)=g\left(x_{n_x}\right). \tag{7.16}$$

In this way we obtain another equation from the upper boundary condition. This gives us the same number of equations as unknowns and thus a solveable system consisting of (7.9) for $i=1,\ldots,n_x-1$, together with (7.13) and (7.16).

Now, applying this derivation to the specific case of the diffusion approximation for preconditioning the Transport equation, we are solving for $F^{(l+1)}$ and have

$$\begin{aligned}
c(x_i)&\equiv\frac{1}{3\sigma_t\left(x_i\right)},\\
d(x_i)&\equiv\sigma_c\left(x_i\right),\\
g(x_i)&\equiv\left\{\sigma_s\left[\phi^{(l+1/2)}-\phi^{(l)}\right]\right\}(x_i),\\
e(x_i)&\equiv\frac{2}{3\sigma_t\left(x_i\right)}.
\end{aligned} \tag{7.17}$$

Substituting these into (7.9), (7.13) and (7.16), we are solving the discretised system

$$\frac{-1}{h^2}\left[\frac{F^{(l+1)}\left(x_{i+1}\right)-F^{(l+1)}\left(x_i\right)}{3\sigma_t\left(x_{i+1/2}\right)}-\frac{F^{(l+1)}\left(x_i\right)-F^{(l+1)}\left(x_{i-1}\right)}{3\sigma_t\left(x_{i-1/2}\right)}\right]+\sigma_c\left(x_i\right)F^{(l+1)}\left(x_i\right)=\left\{\sigma_s\left[\phi^{(l+1/2)}-\phi^{(l)}\right]\right\}\left(x_i\right), \tag{7.18}$$

for $i=1,\ldots,n_x-1$, with

$$\left\{\sigma_c\left(x_0\right)+\frac{2}{3h^2\sigma_t\left(x_{1/2}\right)}\right\}F^{(l+1)}\left(x_0\right)-\frac{2F^{(l+1)}\left(x_1\right)}{3h^2\sigma_t\left(x_{1/2}\right)}=\left\{\sigma_s\left[\phi^{(l+1/2)}-\phi^{(l)}\right]\right\}\left(x_0\right), \tag{7.19}$$

and

$$\left[\left(\frac{2}{h^2}-\frac{3\sigma_t\left(x_{n_x}\right)}{h}\right)\frac{1}{3\sigma_t\left(x_{n_x-1/2}\right)}+\sigma_c\left(x_{n_x}\right)\right]F^{(l+1)}\left(x_{n_x}\right)-\frac{2F^{(l+1)}\left(x_{n_x-1}\right)}{3h^2\sigma_t\left(x_{n_x-1/2}\right)}=\left\{\sigma_s\left[\phi^{(l+1/2)}-\phi^{(l)}\right]\right\}\left(x_{n_x}\right), \tag{7.20}$$

at the end points of the domain. To state this in a more useful form we first define three vectors, p, q and r. These will contain the coefficients of $F^{(l+1)}\left(x_{i-1}\right)$, $F^{(l+1)}\left(x_i\right)$ and $F^{(l+1)}\left(x_{i+1}\right)$ respectively as follows

$$p\left(i\right) = \begin{cases} 0 & \text{if } i = 0, \\ \frac{-1}{3h^2\sigma_t\left(x_{i-1/2}\right)} & \text{if } i = 1, \ldots, n_x - 1, \\ \frac{-2}{3h^2\sigma_t\left(x_{i-1/2}\right)} & \text{if } i = n_x, \end{cases}$$

$$q\left(i\right) = \begin{cases} \sigma_c\left(x_i\right) + \frac{2}{3h^2\sigma_t\left(x_{i+1/2}\right)} & \text{if } i = 0, \\ \frac{1}{3h^2}\left(\frac{1}{\sigma_t\left(x_{i+1/2}\right)} + \frac{1}{\sigma_t\left(x_{i-1/2}\right)}\right) + \sigma_c\left(x_i\right) & \text{if } i = 1, \ldots, n_x - 1, \\ \left(\frac{2}{h^2} - \frac{3\sigma_t(x_i)}{h}\right)\frac{1}{3\sigma_t\left(x_{i-1/2}\right)} + \sigma_c\left(x_i\right) & \text{if } i = n_x, \end{cases}$$

$$r\left(i\right) = \begin{cases} \frac{-2}{3h^2\sigma_t\left(x_{i+1/2}\right)} & \text{if } i = 0, \\ \frac{-1}{3h^2\sigma_t\left(x_{i+1/2}\right)} & \text{if } i = 1, \ldots, n_x - 1, \\ 0 & \text{if } i = n_x. \end{cases}$$

Under these definitions, the above discrete system (7.18)-(7.20) can be simply written as

$$p\left(i\right)F^{(l+1)}\left(x_{i-1}\right) + q\left(i\right)F^{(l+1)}\left(x_i\right) + r\left(i\right)F^{(l+1)}\left(x_{i+1}\right) = \left\{\sigma_s\left[\phi^{(l+1/2)} - \phi^{(l)}\right]\right\}\left(x_i\right), \tag{7.21}$$

for $i = 0, \ldots, n_x$.

From this format it is simple to write the system as a tridiagonal matrix - vector equation by placing the vectors p, q and r along the three main diagonals of the matrix. Denoting $p\left(i\right)$, $q\left(i\right)$ and $r\left(i\right)$ as p_i, q_i and r_i, this results in

$$\begin{pmatrix} q_0 & r_0 & & & & \\ p_1 & q_1 & r_1 & & & \\ & p_2 & q_2 & r_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & p_{n_x-1} & q_{n_x-1} & r_{n_x-1} \\ & & & & p_{n_x} & q_{n_x} \end{pmatrix}\begin{pmatrix} F^{(l+1)}\left(x_0\right) \\ F^{(l+1)}\left(x_1\right) \\ \vdots \\ F^{(l+1)}\left(x_{n_x-1}\right) \\ F^{(l+1)}\left(x_{n_x}\right) \end{pmatrix} = \begin{pmatrix} \left\{\sigma_s\left[\phi^{(l+1/2)} - \phi^{(l)}\right]\right\}\left(x_0\right) \\ \left\{\sigma_s\left[\phi^{(l+1/2)} - \phi^{(l)}\right]\right\}\left(x_1\right) \\ \vdots \\ \left\{\sigma_s\left[\phi^{(l+1/2)} - \phi^{(l)}\right]\right\}\left(x_{n_x-1}\right) \\ \left\{\sigma_s\left[\phi^{(l+1/2)} - \phi^{(l)}\right]\right\}\left(x_{n_x}\right) \end{pmatrix}. \tag{7.22}$$

In this form, the diffusion approximation is easily solveable using the Thomas algorithm (or Tridiagonal Matrix Algorithm) [28].

Water $0 \leq x \leq 12$	Water $12 \leq x \leq 15$	Iron $15 \leq x \leq 21$	Water $21 \leq x \leq 30$
$\sigma_t = 3.3333$ $\sigma_s = 3.3136$ $f = 1$	$\sigma_t = 3.3333$ $\sigma_s = 3.3136$ $f = 0$	$\sigma_t = 1.3333$ $\sigma_s = 1.1077$ $f = 0$	$\sigma_t = 3.3333$ $\sigma_s = 3.3136$ $f = 0$

Table 1: Cross section data for Transport problem.

7.4 Investigating Convergence

To demonstrate the differences in efficiency of each of the methods introduced we used each of them to solve a model problem. This problem was chosen to be sufficiently small that the solution could be found relatively quickly by other means, allowing us to compare the true error of each solution method. For this purpose, the programs listed in appendix A were written.

The main program `MatrixSetUp.m`, section A.1, constructs the coefficient matrix described by the discrete ordinates method. It also converts this into the Schur complement system for use in simple iteration. Finally it applies each of the iterative methods discussed in chapter 2, and plots the error of each method at each iterative step. It also plots the error curve for the restarted GMRES(5) method, mentioned at the end of chapter 2. `MatrixSetUp.m` makes use of several other subfunctions. Firstly `Greenbaum_p143_SetUp.m`, section A.2, which builds the basis vectors as well as vectors containing the corresponding values of σ_f, σ_t and σ_s depending upon the data for a specific problem. Also `TDMAsolver.m`, section A.3, [8] invokes the Thomas algorithm to solve the tridiagonal system required by DSA. Finally, `gjacobi.m`, section A.4, [1] finds the points required by Gauss quadrature used in the discrete ordinates method.

The problem we used is defined in Greenbaum [3] p143, but was originally from [29]. It is the Transport problem for nuclear fission, with cross sections corresponding to water and iron in different regions. The data for this problem is given in Table 1.

The slab thicknesses, x, are given in cm and the cross sections are in cm^{-1}. Also, the right hand end of the domain has a vacuum boundary condition and the left end has a reflecting boundary condition. Using this data, the Matlab programs given in appendix A were used to produce figure 7-2.

The corresponding graph given by Greenbaum is shown in figure 7-3. One difficulty faced in reproducing Greenbaum's error curves was that the initial guess was not known. We used a random initial guess to produce figure 7-2, however this inevitably leads to differences in results. Despite this, very similar convergence behaviour is seen and in fact the curves for the error in simple iteration and GMRES are almost identical. The error curve for GMRES(5) shows very fast convergence behaviour, beyond that of GMRES. The convergence of Orthomin(1) is more sensitive to the initial guess,

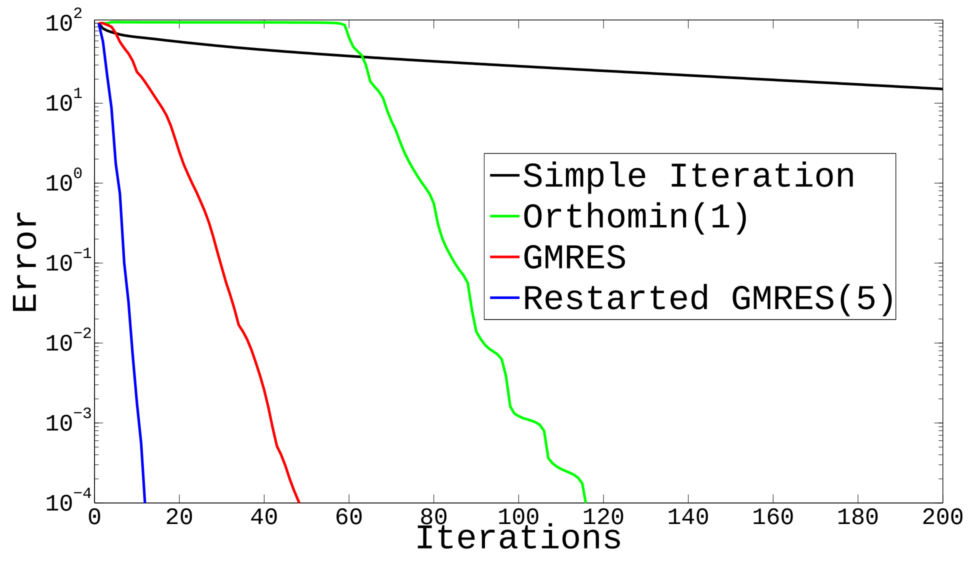

Figure 7-2: *Error curves for three iterative solution methods applied to the model Transport problem.*

however it displays the same step-like behaviour shown by the corresponding curve in figure 7-3. The further curve in figure 7-3 is for a solution using simple iteration with DSA preconditioning. Unfortunately, the code we wrote for carrying out this solution method did not converge and so the corresponding curve is not included in figure 7-2. It is still included within `MatrixSetUp.m` in appendix A for reference, however it should be noted that it contains an as yet unresolved error.

From figure 7-2 it is clear that for the model problem simple iteration is not efficient enough to be a practical solution method. Orthomin(1) provides a significant improvement to this with comparatively little extra work. Its sensitive dependence upon the initial guess means that its convergence can easily range between that of

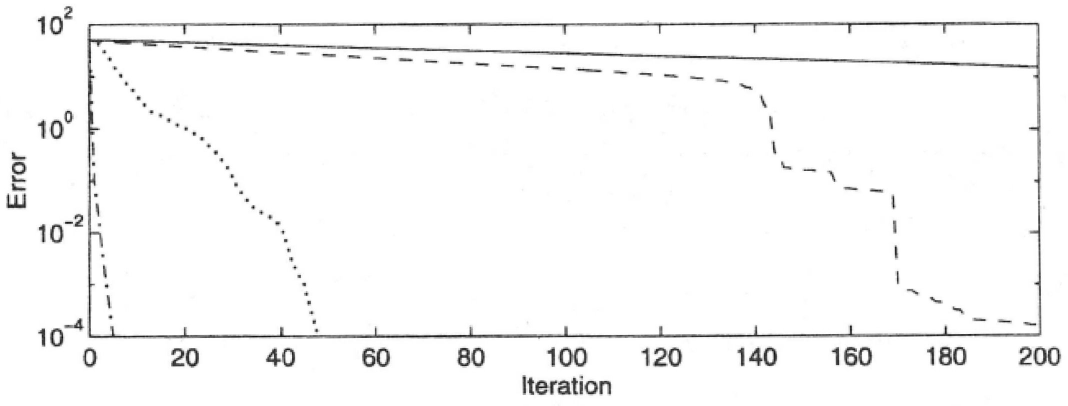

Figure 7-3: *Error curves for Simple iteration (solid), Orthomin(1) (dashed), full GMRES (dotted) and DSA-preconditioned simple iteration (dash-dot), taken from Greenbaum p143 [3].*

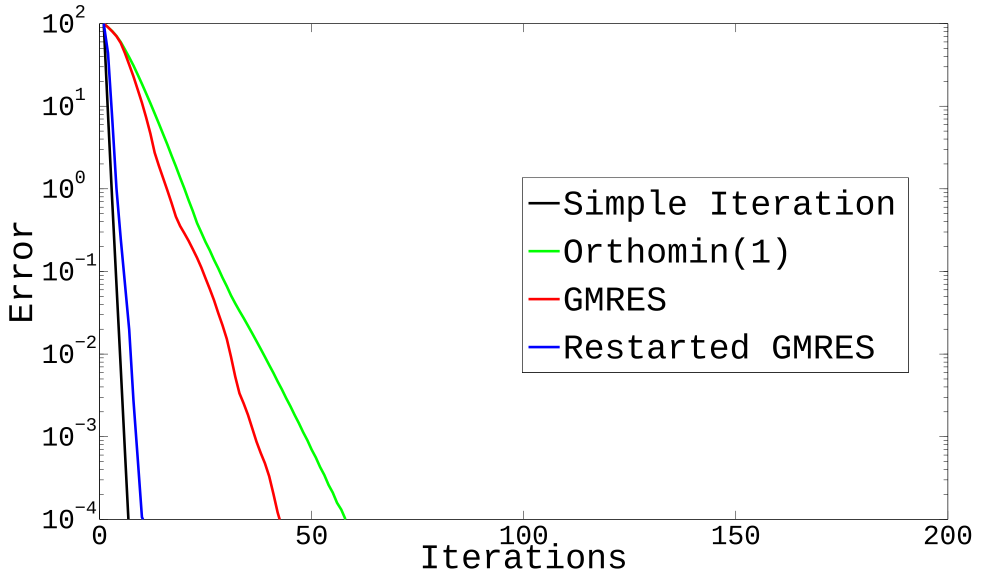

Figure 7-4: *Error curves for three iterative solution methods applied to the model Transport problem with σ_s scaled by* 0.1.

GMRES and simple iteration. However with an appropriate choice of initial guess this could possibly be minimised. GMRES provides a robust and fast convergence to the true solution, outperforming simple iteration and Orthomin(1). Its downside is that it requires a large amount of storage and so for higher dimensional problems may be impractical. One way around this issue is to use the restarted GMRES method and here we implemented GMRES(5). This showed faster rates of convergence and would largely reduce storage requirements. The theory behind the convergence of both GMRES and restarted GMRES is explored in [3] and further in [15]. The error curve for simple iteration with DSA preconditioning in figure 7-3 shows the highest rate of convergence and is the most effective solution method for this problem.

Figure 7-4 shows the error plot for the model problem but with the values of σ_s scaled by a factor of $\frac{1}{10}$. In this case, simple iteration is actually the fastest converging of all the methods. This can be understood by considering the remark made in chapter 6, after theorem 6.1. Here, we noted that the smaller σ_s is compared to σ_t, the faster source iteration will converge. This is demonstrated clearly by figure 7-4, where σ_s has been made considerably smaller than σ_t. In problems with such a low σ_s/σ_t ratio, source iteration can provide an efficient and less computationally demanding solution method. In these cases, even if DSA preconditioning can improve this rate of convergence, it may not be beneficial to do so as it will be adding an unnecessary level of complexity, having to conduct both a Transport sweep and solve the diffusion approximation. It becomes practical only when the time saved in reducing the required iterations outweighs the extra computation time required for each iteration.

REFERENCES

[1] Fynn Scheben. *Iterative Methods for Criticality Computations in Neutron Transport Theory.* PhD thesis, University of Bath, 2011.

[2] Fynn Scheben and Ivan G. Graham. Iterative methods for neutron transport eigenvalue problems. *To appear in SIAM J. Sci.Comp*, Accepted in 2011.

[3] Anne Greenbaum. *Iterative Methods for Solving Linear Systems.* SIAM, 1997.

[4] G.C. Pomraning. *Equations of Radiation Hydrodynamics (Monographs in Natural Philosophy).* Elsevier, 1973.

[5] Peter N. Brown. A linear algebraic development of diffusion synthetic acceleration for three-dimensional transport equations. *SIAM J. Numer. Anal.*, 32:179–214, 1995.

[6] E. E. Lewis and W. F. Miller. *Computational Methods of Neutron Transport.* Wiley-Interscience, 1993.

[7] M. L. Adams and E. W. Larsen. Fast iterative methods for discrete-ordinates particle transport calculations. *Progress in Nuclear Energy*, 40(1):3 – 159, 2002.

[8] Wikipedia article: Tridiagonal matrix algorithm. Retrieved September 8, 2011 from http://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm.

[9] J. Schroder, U. Trottenberg, and K. Witsch. On fast poisson solvers and applications. *Lecture Notes in Mathematics, Springer, Berlin*, 631:153 – 187, 1978.

[10] A. N. Krylov. On the numerical solution of the equation by which in technical questions frequencies of small oscillations of material systems are determined. *Izvestija AN SSSR. Otdel. mat. i estest. nauk, VII*, 4:491–539, 1931.

[11] James W. Longley. Modified gram-schmidt process vs. classical gram-schmidt. *Communications in Statistics - Simulation and Computation*, 10(5):517–527, 1981.

[12] B. N. Datta. *Numerical methods for linear control systems.* Elsevier Academic Press San Diego, London, 2004.

[13] Valérie Frayssé, Luc Giraud, Serge Gratton, and Julien Langou. A set of gmres routines for real and complex arithmetics on high performance computers. *CERFACS Technical Report TR/PA/03/3*, 2003.

[14] Y. Saad and M. H. Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. and Stat. Comput.*, 7(3):856–869, 1986.

[15] Jörg Liesen and Petr Tichý. Convergence analysis of krylov subspace methods. *Retrieved on September 9th, 2011 from http://www.cs.cas.cz/tichy/download/public/LiTiGAMM2005.pdf*, 2000.

[16] F. Zhang. *The Schur Complement and Its Applications (Numerical Methods and Algorithms).* Springer, 2005.

[17] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial.* SIAM, 2000.

[18] Olinde Rodrigues. De l'attraction des sphéroïdes. *Correspondence sur l'École Impériale Polytechnique*, 3(3):361 – 385, 1916.

[19] E. T. Whittaker and G. N. Watson. *A Course of Modern Analysis.* Cambridge University Press, 1935.

[20] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions.* Dover, 1970.

[21] Michael Stone and Paul Goldbart. *Mathematics for Physics: A Guided Tour for Graduate Students.* Cambridge University Press, 2009.

[22] R. Courant and D. Hilbert. *Methods of Mathematical Physics, Volume 1.* John Wiley & Sons, 1966.

[23] S. F. Ashby, P. N. Brown, M. R. Dorr, and A. C. Hindmarsh. A linear algebraic analysis of diffusion synthetic acceleration for the boltzmann transport equation. *SIAM J. Numer. Anal.*, 32:128–178, 1995.

[24] H. J. Kopp. Synthetic method solution of the transport equation. *NucL Sci. Eng.*, 17:65, 1962.

[25] L. F. Richardson. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 210:pp. 307–357, 1911.

[26] Louis A. Hageman and David M. Young. *Applied Iterative Methods.* Academic Press, inc., 1981.

[27] Mark H. Holmes. *Introduction to Numerical Methods in Differential Equations (Texts in Applied Mathematics).* Springer, 2006.

[28] Forman S. Acton. *Numerical Methods that Work (Spectrum).* The Mathematical Association of America, 1997.

[29] D. R. McCoy and E. W. Larsen. Unconditionally stable diffusion-synthetic acceleration methods for the slab geometry discrete ordinates equations. part ii: Numerical results. *Nuclear Sci. Engrg.*, 82:64–70, 1982.

APPENDIX A

MATLAB CODE

A.1 MatrixSetUp.m

```
function [A] = MatrixSetUp(M,N)
% MatrixSetUp(M,N)
% Solves a discretised form of the Boltzmann Transport equation, obtained
% using the method of discrete ordinates, using simple iteration, the
% Orthomin(1) method, GMRES, restarted GMRES(5), and optionally
% simple iteration with DSA preconditioning. This option is specified
% using the following variable: 1 = 'yes', 0 = 'no'
    do_DSA = 0;
% This is left optional since the current code for implementing this
% method contains unresolved errors.
%
% This function is structured as follows:
%
% - Set up the coefficient matrix obtained via discrete ordinates, using
%   relevant functions Greenbaum_p143_SetUp.m and gjacobi.m
% - Construct the Schur complement system for use with simple iteration
% - initialise solution vector, phi
% - use 'slash' operator to obtain the true solution. Will be used to
%   find the error norms at each iteration.
% - carry out all iterative methods, storing vectors of the error norms
%   from each iteration
%   plot vectors of error norms for each method, using log scaled error


close all
```

```
% Set up sigmas for Greenbaum problem, p143
[sig_t_halfbasis,sig_s_halfbasis,f_vector,delt_x]=Greenbaum_p143_SetUp(M);

% Gauss Quadrature points for diamond differencing method
[pts,wts] = gjacobi(2*N,0);
mu       = pts(2*N:-1:1);

% initialise diagonals of the blocks of the final matrix
D_TL    = [zeros((M+1)*2*N,1) ...
           ones((M+1)*2*N,1)  ...
           zeros((M+1)*2*N,1) ];
diag_TL = [-1 0 1];

D_TR    = zeros(M,2*N);
diag_TR = zeros(1,2*N);

D_BL    = ones(M,4*N);
diag_BL = zeros(1,4*N);

% Also want to create RHS, F, so initialise
F       = zeros((M+1)*2*N+M,1);

% loop form the diagonals of the blocks of the final matrix, and
% specify where they are to be placed, i.e. upon which diagonals
%
%   D_**    = the entries for a specific block of the coeff matrix
%   diag_** = the diagonals upon which the entries should be placed
%
for j = 1:2*N

    % evaluate top right diagonal entries
    D_TR(:,j)      = sig_s_halfbasis;

    % placement of bottom left diagonal entries
    diag_BL(2*j-1)  = (j-1)*(M+1);
    diag_BL(2*j)    = (j-1)*(M+1)+1;

    % evaluate bottom left diagonal entries
    D_BL(:,2*j-1)   = -wts(j)/4;
    D_BL(:,2*j)     = -wts(j)/4;

    if j<=N % i.e. for j <= (n_mu)/2

        % evaluate top right diagonal entries
        D_TL((j-1)*(M+1)+1:j*(M+1)-1,2) =  abs(mu(j))./delt_x+...
                                            1/2*sig_t_halfbasis;
        D_TL((j-1)*(M+1)+2:j*(M+1),3)   = -abs(mu(j))./delt_x+...
```

```
                                                      1/2*sig_t_halfbasis;

        % placement of TR diag entries
        diag_TR(j)                       = -(j-1)*(M+1);

        % evaluate f_vector, with boundary condition for j <= (n_mu)/2
        F((j-1)*(M+1)+1:j*(M+1)-1)       = f_vector;
        F(j*(M+1))                       = 0; % phi_b: vacuum BC

    else % if  j > N,  i.e. for j >= (n_mu)/2

        % evaluate top left diagonal entries
        D_TL((j-1)*(M+1)+2:j*(M+1),2)    =  abs(mu(j))./delt_x+...
                                                      1/2*sig_t_halfbasis;
        D_TL((j-1)*(M+1)+2:j*(M+1),1)    = -abs(mu(j))./delt_x+...
                                                      1/2*sig_t_halfbasis;

        % placement of TR diag entries
        diag_TR(j)                       = -(j-1)*(M+1)-1;

        % evaluate f_vector, with boundary condition for j >= (n_mu)/2
        F((j-1)*(M+1)+2:j*(M+1))         = f_vector;
        F((j-1)*(M+1)+1)                 = 0; % reflecting boundary cond.

    end % if

end % for


% form each block in sparse diagonal matrix form
Block_TL    = spdiags(D_TL,diag_TL,(M+1)*2*N,(M+1)*2*N);
Block_TR    = spdiags(D_TR,diag_TR,(M+1)*2*N,M);
Block_BL    = spdiags(D_BL,diag_BL,M,(M+1)*2*N);
Block_BR    = eye(M,M);

% add opposing coefficients for reflecive BC
for j = N+1:2*N
    Block_TL((M+1)*(j-1)+1,(M+1)*(2*N-j)+1) = -1;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       FORM THE COEFFICIENT MATRIX FOR THE FULL SYSTEM
A = [Block_TL Block_TR ;...
     Block_BL Block_BR  ];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       BUILD THE SCHUR MATRICES
len_TL = length(Block_TL);
A_sch = Block_BR - Block_BL * (Block_TL\eye(len_TL)) * Block_TR;
F_sch = -Block_BL * (Block_TL\eye(len_TL)) * F(1:len_TL);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%% Calculate 'exact' solution for use in error calculations
xsol_full   = A\F;                          % FULL system
xsol_sch    = A_sch\F_sch;                  % SCHUR system


%%%% Set up initial guess for both systems
initial_full    = rand(length(A),1)*100;    % FULL system
initial_sch     = initial_full(end-M+1:end);% SCHUR system

%%%% Define a convergence tolerance
tol = 1e-5;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simple Iteration solution, using method as defined on p138 of Greenbaum
% solves the SCHUR COMPLEMENT system (cf. p138)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
resvec_simple = zeros(201,1);
phi           = initial_sch;

for i = 1:200

    % calculate residual vector
    r_k              = F_sch - A_sch*phi;

    % save current residual norm value
    resvec_simple(i) = norm(xsol_sch-phi,inf);

    % update solution approximation
    phi              = r_k + phi;

    if norm(r_k) < tol
        break
    end

end
```

```
fprintf(['Simple method had an error of %g'...
    ' after %g iterations\n'],resvec_simple(i),i)

clear r_k phi


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simple Iteration WITH ORTHOMIN(1) solution, using method as defined on
% p138 and p29 of Greenbaum
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
resvec_orthomin   = zeros(201,1);
phi               = initial_full;

for i = 1:200

    %find or update residual using one of two methods
    if i == 1
        % calculate residual vector
        r_k = F - A*phi;
    else
        r_k = r_k - a_k*Ar_k;
    end

    % save current residual norm value
    resvec_orthomin(i) = norm(xsol_full-phi,inf);

    %resvec_orthomin(i) = norm(phi - xsol);

    % find A*r_k
    Ar_k = A*r_k;

    % a_k coefficient calculation
    a_k     = r_k'*Ar_k/norm(Ar_k)^2;

    % update solution approximation
    phi     = phi + a_k*r_k;


    if norm(r_k) < tol
        break
    end

end

fprintf(['Orthomin method had an error of '...
                '%g after %g iterations\n'],resvec_orthomin(i),i)

clear r_k phi a_k Ar_k
```

```
if do_DSA == 1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simple Iteration WITH DSA PRECONDITIONING, using method as defined on
% p138 of Greenbaum. Solves the SCHUR COMPLEMENT system (cf. p138)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
resvec_DSA = zeros(201,1);
phi1       = initial_sch;

Length     = 30;
h          = Length/M;
x          = (0:h:Length)';

% initialise sigma vectors
sig_t      = zeros(length(x),1);
sig_s      = zeros(length(x),1);

% define sigma vectors (for x_i's, true grid points)
for i = 1:length(x)
    if x(i)<15 || x(i)>=21
        sig_t(i) = 3.3333;
        sig_s(i) = 3.3136*0.1;
    else
        sig_t(i) = 1.3333;
        sig_s(i) = 1.1077*0.1;
    end
end

% constuct sig_c
sig_c = sig_t - sig_s;

% define p and r for use in the thomas algorithm
% these are the diagonals below and above leading
% (respectively) in the coefficient matrix
p = -[0 ; 1./(3*sig_t_halfbasis(1:M-1)) ; 0] / (h^2);
r = -[0 ; 1./(3*sig_t_halfbasis(2:M))   ; 0] / (h^2);

% define q for use in the thomas algorithm
% this is the leading diagonal in the coefficient matrix
q =  [0 ; -(p(2:M)+r(2:M))/(h^2) + sig_c(2:M) ; 0];

%%%% ADD BOUNDARY CONDITIONS %%%%%%%%%%%%%%%%%%%%
% Left end,  x = a
sig_t_onehalf = sig_t_halfbasis(1);
q(1) = (sig_c(1)+2/(3*sig_t_onehalf*h^2));
r(1) = -2/(3*sig_t_onehalf*h^2);
```

```
% Right end, x = b
sig_t_nxhalf = sig_t_halfbasis(end);
p(end) = -2/(3*sig_t_nxhalf*h^2);
q(end) = (2/(h^2)-3*sig_t(end)/h)/(3*sig_t_nxhalf)+sig_c(end);

for i = 1:200

    %%%%%%% SIMPLE ITERATION PART %%%%%%%%%%%%%
    % calculate residual vector
    r_k = F_sch - A_sch*phi1;

    % save current residual norm value
    resvec_DSA(i) = norm(xsol_sch-phi1,inf);

    % update solution approximation
    phi2 = r_k + phi1;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Convert from 'averaged' format
    % to 'actual grid points' format
    phi1long = zeros(M+1,1);
    phi2long = zeros(M+1,1);

    for k = 2:M
        phi1long(k) = (phi1(k-1)+phi1(k))/2;
        phi2long(k) = (phi2(k-1)+phi2(k))/2;
    end

    phi2long(1)   = phi2(1);
    phi2long(end) = phi2(end);
    phi1long(1)   = phi1(1);
    phi1long(end) = phi1(end);


    % find right hand side vector for DSA
    RHS = phi2long-phi1long;
    RHS = RHS.*sig_s;

    %%%%%%% DSA ITERATION PART %%%%%%%%%%%%%%%%
    % Thomas Algorithm to solve tri-diag matrix system
    Flp1 = TDMAsolver(p,q,r,RHS);

    % calculate value of phi at next iteration
    temp = phi2long + (Flp1')/2;

    for k = 1:M
        phi1(k) = (temp(k)+temp(k+1))/2;
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if norm(r_k) < tol
        break
    end

end

fprintf(['Simple method with DSA had an error of %g '...
    ' after %g iterations\n'],resvec_DSA(i),i)

clear r_k phi

end % do_DSA ?

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Full GMRES solution, no preconditioner %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

resvec_GMRES    = zeros(201,1);
resvec_GMRES(1) = norm(xsol_full - initial_full,inf);

for k = 1:200
    [tempsol,flag] = gmres(A,F,[],tol,k,[],[],initial_full);
    resvec_GMRES(k+1) = norm(xsol_full-tempsol,inf);
    if flag == 0
        break
    end
end

fprintf(['GMRES had an error of %g '...
    ' after %g iterations\n'],resvec_GMRES(k),k)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Restarted GMRES solution, no preconditioner, restarts every 5 iterations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

resvec_RestGMRES    = zeros(201,1);
resvec_RestGMRES(1) = norm(xsol_full - initial_full,inf);

for k = 1:200
    [tempsol,flag] = gmres(A,F,5,tol,k,[],[],initial_full);
    resvec_RestGMRES(k+1) = norm(xsol_full-tempsol,inf);
    if flag == 0
```

```
        break
    end
end

fprintf(['Restarted GMRES had an error of %g '...
    ' after %g iterations\n'],resvec_RestGMRES(k),k)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



% Plot residual vectors for each solution method
hold off

semilogy(resvec_simple,'k-','linewidth',3),hold on
semilogy(resvec_orthomin,'g-','linewidth',3),hold on
if do_DSA == 1, semilogy(resvec_DSA,'c.-','linewidth',3),hold on, end
semilogy(resvec_GMRES,'r-','linewidth',3),hold on
semilogy(resvec_RestGMRES,'b-','linewidth',3),hold on

axis([0 200 10^-4 10^2])

if do_DSA == 1,
    leg = legend('Simple Iteration','Orthomin(1)','DSA','GMRES',...
                                        'Restarted GMRES');
else
    leg = legend('Simple Iteration','Orthomin(1)','GMRES',...
                                        'Restarted GMRES');
end

xlabel('Iterations','fontsize',30)
ylabel('Error','fontsize',30)

set(gca,'fontsize',20)
set(leg,'fontsize',20)





end
```

A.2 Greenbaum_p143_SetUp.m

```
function [sig_t_halfbasis,sig_s_halfbasis,f_vector,delt_x] =...
                                                    Greenbaum_p143_SetUp(M)
% Function to set up the sigma vectors, the f vector and the
% discretisations needed for the discrete ordinates method. Data used is
% given in:     Anne Greenbaum − Iterative Methods − p143

% interval length calculation
Length = 30;
h = Length/M;

% DATA
sigma_t_water = 3.3333; %
sigma_t_iron  = 1.3333; %
sigma_s_water = 3.3136*0.1; %
sigma_s_iron  = 1.1077*0.1; %
sigma_f       = 1;      %

% domain discretisation
x       = (0:h:Length)';
mid_x   = 1/2*(x(2:end) + x(1:end−1));
delt_x  = x(2:end) − x(1:end−1);

% initialise sigma and f vectors
sig_t_halfbasis  = zeros(length(mid_x),1);
sig_s_halfbasis  = zeros(length(mid_x),1);
f_vector         = zeros(length(mid_x),1);

% form sigma and f vectors
for j = 1:length(mid_x)

    if mid_x(j)<15 || mid_x(j)>=21
        sig_t_halfbasis(j) = sigma_t_water;
        sig_s_halfbasis(j) = sigma_s_water;
    else
        sig_t_halfbasis(j) = sigma_t_iron;
        sig_s_halfbasis(j) = sigma_s_iron;
    end

    if mid_x(j)<12
        f_vector(j) = sigma_f;
    end
end

end
```

A.3 TDMAsolver.m

```
function x = TDMAsolver(a,b,c,d)
% a, b, c are the column vectors for the compressed tridiagonal
% matrix, d is the right vector

n = length(b); % n is the number of rows

% Modify the first-row coefficients
c(1) = c(1) / b(1);    % Division by zero risk.
d(1) = d(1) / b(1);    % Division by zero would imply a singular matrix.

for i = 2:n
    id = 1 / (b(i) - c(i-1) * a(i));  % Division by zero risk.
    c(i) = c(i)* id;                 % Last value calculated is redundant.
    d(i) = (d(i) - d(i-1) * a(i)) * id;
end

% Now back substitute.
x(n) = d(n);
for i = n-1:-1:1
    x(i) = d(i) - c(i) * x(i + 1);
end
```

A.4 gjacobi.m

```
function [pts,wts] = gjacobi(n,p)

% Computes the points and weights of the n point Gauss−Jacobi rule
% on (−1,1)  with  weight function (1−y)ˆp.
% These are computed by the Golub−Welsch eigenproblem trick
% described in Davis and Rabinowitz (2nd Edition) Section 2.7.5
% The appropriate eigenvalue problems are computed by the built−in
% Matlab function eig.
%
% Assemble the appropriate tridiagonal matrix described in D&R page 119:
%


if(n==1)

 pts = 1−2*((p+1)/(p+2));    % The calculation of the tridiagonal will
 wts = (2ˆ(p+1))/(p+1);      % fail if n = 1. We do it manually. x

else     % if n > 1

 scale = (1:n)';
 alpha = (2*(1:n−1).*((1:n−1)+p))./...
        ((2*(1:n−1)+p).*sqrt((2*(1:n−1)+p−1).*(2*(1:n−1)+p+1)));


 if(p==0)
   beta = zeros(1,n);
 else
  beta = −pˆ2./((2*(1:n)−2+p).*(2*(1:n)+p));
  end

 [V,D] = eig(diag(beta) + diag(alpha,−1) + diag(alpha,1));

 [pts,ind] = sort(diag(D));

 wts = (V(1,1:n)./norm(V(:,1:n))).ˆ2; % The first entries of the
                                     % normalised eigenvectors

 wts = ((2ˆ(p+1))/(p+1))*wts(ind)';   % multiply by the integral of the
                                     % weight function and
                                     % pick out the weights corresponding to
                                     % the reordered points


 end  %if
```