Solving Linear Systems

The physical relevance of computations based on the model problems arising from the electrical activity in the heart depends on high accuracy of the solution. High accuracy implies the solution of large linear or nonlinear systems of ODEs and PDEs. This chapter deals with solution algorithms for the discretization of (linear) PDEs, which is a huge research field around the world. A lot of the research in this field has been centered around simple model problems such as the Poisson problem, where a solid theoretical framework has been developed. We will briefly review this theory in the simplest possible manner. Then, in the end of the chapter, we explain how the powerful concept of (block) preconditioning extends these algorithms also to systems of PDEs arising from the discretization of the Bidomain model.

1 Overview

As discussed in the previous chapter, the computation of the electrical activity in the heart and body requires the solution of large linear systems. With today's extremely fast computers, when even desktop computers are capable of performing more than 10^9 floating point operations per second, this might not seem as a problem. But it is. In our quest for more accurate solutions to models of physical problems, we tend to solve larger and larger linear systems. This trend is unavoidable, because the accuracy of the solution is proportional to some power of the number of unknowns.

Let us for the moment consider a linear system arising from a finite element discretization of a partial differential equation of the form

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

equipped with suitable boundary conditions. The linear system is then becomes

$$Au_h = b, \tag{2}$$

where A typically is large and very sparse. We have used the subscript h to emphasize that u_h is an approximation of u and that the accuracy depends on h, which is the typical distance between the vertices in the grid. Because u_h approaches u as h approaches zero, we want to be able to chose h as small as possible. The smallest possible h is dictated by the largest number of unknowns that can be handled by the computer. On a modern desktop computer we can solve (2) with $N \sim 10^7$ unknowns provided that we use an optimal solution algorithm. However, we may want even larger values of N and, as we have already seen, we want to solve systems of partial differential equations leading to even larger linear systems. Also, the linear systems arising from the Bidomain equations are more challenging to solve that the system (2) generated by the Poisson equation (1).

The matrices arising from low order discretization of PDEs are typically very sparse, that is, relatively few entries are non-zeroes. In fact, the number of non-zeroes in each row is usually a fixed small number, e.g., between five and ten and the matrices therefore require $\mathcal{O}(N)$ floating point numbers of storage. For such matrices, it is essential that only the nonzero entries are stored. The matrix may be banded or completely unstructured depending on the structure of the underlying grid. For banded matrices it is possible to develop an unpivoted factorization procedure using $\mathcal{O}(Nb^2)$ floating point operations, where b is the bandwidth of the system. However, the bandwidth is usually at least $\mathcal{O}(N^{1/2})$, which makes the total solution algorithm $\mathcal{O}(N^2)$ in number of floating point operations. The storage requirement is usually somewhat smaller, $\mathcal{O}(N^{3/2})$ floating points. The case is even worse for unstructured matrices.

The improvement of the computers have for the last 30 years been remarkably well predicted by Moore's law, which states that the number of transistors per area will double every 18 month[1]. A consequence of this is that the speed of the CPU and the amount of RAM memory double in the same period. In other words, the speed of the CPU has improved roughly by a factor 10^5 the last 30 years. In recent years, the amount of RAM has increased at a similar rate. Assuming that this law will be valid also for the next 10 years, we expect a 50 times faster CPU with 50 times more RAM memory. Still, it may very well be that the accuracy is questionable and that we need as many unknowns as possible. Hence, for simplicity we assume that it is possible to make the matrix with $50N$ unknowns. It would then not be possible to solve the system, because the banded Gaussian elimination is $\mathcal{O}(N^{3/2}))$ in storage, which in this concrete case means $\approx 350N$. However, the situation is even worse for the number of floating point operations, which is $\mathcal{O}(N^2)$, or $2500N$, in this concrete case. Because the CPU is able to perform $50N$ operations per second, we would then need to wait 50 times longer for the answer than today. This is not acceptable and we therefore have to search for faster methods requiring less storage. More precisely, we shall seek optimal methods requiring $\mathcal{O}(N)$ operations and $\mathcal{O}(N)$ memory allocations for a system with N unknowns.

2 Iterative Methods

As discussed above, direct methods have two main problems:

- the required storage is $\mathcal{O}(N^\alpha)$ floating point numbers, and
- the required floating points operations are $\mathcal{O}(N^\beta)$,

[1] Moore never formulated the law clearly, but this is commonly known as Moore's law. However, the doubling period may vary.

where $\alpha, \beta > 1$ and the matrix to be solved is very sparse, in fact the required storage is $\mathcal{O}(N)$. These observations have lead to the development of algorithms that take advantage of the particular structures of the matrices. This field of research is huge and successful [5]. As seen in Table 2, the computational time needed to solve the problem with a highly efficient multigrid algorithm is about a 1% of the banded Gaussian elimination, when using 257^2 unknowns. Moreover, banded Gaussian elimination runs out of memory at this point and can not be compared with the iterative methods. Still, it is worth noting that the multigrid algorithms using 1025^2 unknowns outperforms banded Gaussian elimination with 257^2 unknowns.

Table 1. The CPU time (in seconds) required to solve a Poisson problem in 2D, using different types of solution algorithms, with respect to the number of unknowns. For the iterative algorithms, the stopping criterion requires that the L_2-norm of the residual is reduced by a factor of 10^8. The measurements are obtained on an Itanium2 1.3 GHz processor.

Unknowns	Gauss Elim.	CG	CG/MILU	MG	CG/MG
65^2	0.29	0.12	0.04	0.04	0.04
129^2	4.30	1.07	0.28	0.15	0.18
257^2	68.49	12.30	2.77	0.64	0.92
513^2	-	123.06	18.65	2.89	4.08
1025^2	-	969.21	111.31	12.07	16.90

2.1 The Richardson Iteration

Here we will briefly introduce iterative methods designed to solve linear systems. We will begin with classical schemes and then move on to more advanced methods. Our ultimate goal is to derive an optimal method for the linear system arising from discretizations of the Bidomain model. We want to solve the linear system,

$$Au = b.$$

The matrix, A, is sparse, but A^{-1} is in general full and require the storage of N^2 floating points. When $N \sim 10^7$, which is quite common, the inverse can neither be computed nor stored. However, the matrix-vector product, $w = Av$, requires only $\mathcal{O}(N)$ operations. Therefore, a first attempt for a more memory efficient algorithm might be a fix point iteration,

$$u^n = u^{n-1} - \tau(Au^{n-1} - b). \tag{3}$$

This algorithm is commonly called the *Richardson iteration* or the *simple iteration.* The obvious question is whether the iteration converges to the solution or not. A standard approach to analyze iterative methods is to assume that the solution u is known, such that we can investigate how the error behaves. The error in the n'th iteration is defined as

$$e^n = u^n - u. \tag{4}$$

Hence, by subtracting u from both sides of (3) and using the relation, $Au = b$, we obtain a recursion for the error,

$$e^n = e^{n-1} - \tau A e^{n-1}. \tag{5}$$

The error at the n'th iteration, e^n, should then be smaller, in some sense, than the previous ones. To quantify the error behavior in "some sense" we introduce the discrete L_2-norm,

$$\|e\|_{L_2} = (\frac{1}{N}\sum_{i=1}^{N} e_i^2)^{1/2}, \tag{6}$$

and the discrete L_2 inner product,

$$(e,f)_{L_2} = (\frac{1}{N}\sum_{i=1}^{N} e_i f_i). \tag{7}$$

The corresponding matrix norm is defined by

$$\|A\| = \sup_v \frac{(Av,v)_{L_2}}{(v,v)_{L_2}}.$$

We will in the following drop the subscript L_2.

A necessary and sufficient condition for convergence is that the error decreases during the iteration,

$$\|e^n\| \le \rho \|e^{n-1}\|,$$

where $0 \le \rho < 1$. This imply that the Richardson iteration is a contraction and convergent if and only if

$$\|I - \tau A\| \le \rho < 1, \tag{8}$$

or

$$0 < (1-\rho) \le \|\tau A\| \le (1+\rho) < 2, \tag{9}$$

From (5) and (8) the error will decrease

$$\|e^n\| = \|(I-\tau A)e^{n-1}\| \le \|I-\tau A\|\|e^{n-1}\| \le \rho\|e^{n-1}\|.$$

A more detailed mathematical description of this methods and the corresponding convergence proofs can be found in Chapter 3 in [13].

The parameter τ can be chosen and a possible choice is $\tau = \frac{c}{\|A\|}$, with $c < 2$. Hence, $\|A\|$ has to be computed or estimated. In the general case, the computation of $\|A\|$ is not easy, but for the applications considered in this chapter it is sufficient to assume that it is on the form $\tau_2 N^{2/d}$, where d is the number of space dimensions, and manually tune τ_2. The estimate does not need to be very accurate, but a too small τ_2 will lead to divergence.

This iteration has some of the basic characteristics we are seeking. The matrix requires $\mathcal{O}(N)$ floating points in storage, which imply that

– Each iteration involves $\mathcal{O}(N)$ floating point operations.
– The entire method requires only the storage of $\mathcal{O}(N)$ floating points.

This is a potential advantage over direct methods, but we will see that it is not enough. The Richardson iteration has poor performance for matrices that come from the discretization of PDEs. To explain this we introduce a FDM discretization of the Poisson problem in 1D. This example is very simple, but it still has the basic features of the problems that we want to address.

2.2 The FDM Discretization Poisson Equation in 1D

Efficient algorithms take advantage of particular properties of the matrix. The classical iterations; Jacobi, Gauss-Seidel, SOR, and SSOR are usually more efficient than the Richardson iteration, but they are not general purpose algorithms. Some properties are required. This also apply to the Conjugate-Gradient method. Optimal solution algorithms like multigrid and domain decomposition need even more properties to be present. Since multigrid was introduced in 1960 and domain decomposition was introduced already in 1870, these algorithms have been extensively studied by many researchers and the theoretical framework has reached a high degree of sophistication and abstraction [4] and [13]. We will start by explaining the underlying ideas, in the simplest possible fashion. In this respect, the exposition is similar to [6].

First, the basic properties of the FDM discretization of the Poisson problem in 1D are reviewed. This model problem reads

$$-u''(x) = f(x),\ x = (0,1),\ u(0) = 0,\ u(1) = 0. \tag{10}$$

We are not interested in any particular solution, but rather the class of problems on this form. We therefore introduce the differential operator $L = -\frac{\partial^2}{\partial x^2}$, such that the problem can be written as

$$Lu = -u''(x) = f(x),\ x = (0,1),\ u(0) = 0,\ u(1) = 0. \tag{11}$$

The operator L has the following properties:

– It is *positive definite*; this means that $(Lu, u) > 0$ for all relevant functions u.

- It is *symmetric*; this means that $(Lu, v) = (u, Lv)$ for all relevant functions u.
- It is *invertible* for any $f \in C(0,1)$; this means that the problem (10) has a unique solution for $f \in C(0,1)$.

These properties can be derived directly from the definition of L, see, e.g., [26].

Later we will see that a lot of the intuition concerning the solution of PDEs and the solution algorithms is closely connected to eigenvalues and eigenfunctions for the differential operators. The definition of eigenvalues and eigenfunctions for differential operators is equivalent to the definition for matrices in linear algebra, i.e., λ is an eigenvalue of L if

$$Lw = \lambda w, \quad \lambda \in \mathbb{C}, \tag{12}$$

and w is the corresponding eigenfunction. Because the operator is symmetric and positive definite, the eigenvalues are real and positive, and the eigenfunctions are orthogonal. In fact, it is known that the eigenvalues for L in (11) are

$$\mu_k = \pi^2 k^2, \quad \text{for } k = 1, \ldots, \infty,$$

and the eigenfunctions are

$$w_k(x) = \sin(k\pi x), \quad \text{for } k = 1, \ldots, \infty.$$

These properties can be derived directly from the definition of L, see, e.g., [26].

The first four eigenfunctions are shown in Figure 1. Notice that for k small, the eigenvalues are small and the corresponding eigenfunctions are low frequency functions. However, as $k \to \infty$, $\mu_k \to \infty$ and the eigenfunctions oscillates between 1 and -1 with a period $\frac{\pi}{k}$ which approaches zero. This relation between the smoothness of the eigenfunctions and the magnitude of the eigenvalues is typical for elliptic PDEs, regardless of dimensions, domains and boundary conditions.

This 1D problem can be solved analytically, but this is not feasible for the Poisson problem on general domains in 2D or 3D. However, discretization techniques like the finite element method (FEM)[2] or the finite difference method (FDM)[3] are general purpose strategies. The next step is to make an approximation of the problem such that we arrive at a linear system that can be solved. Let the grid consist of grid points $x_i = ih$. A FDM discretization of (10) reads,

$$-\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} = f_i, \quad i = 1, \ldots N, \tag{13}$$

$$u_0 = 0, \tag{14}$$

$$u_{N+1} = 0. \tag{15}$$

[2] FEM requires that it is possible to generate a reasonable grid.
[3] Also FDM is hard to apply in general geometries.

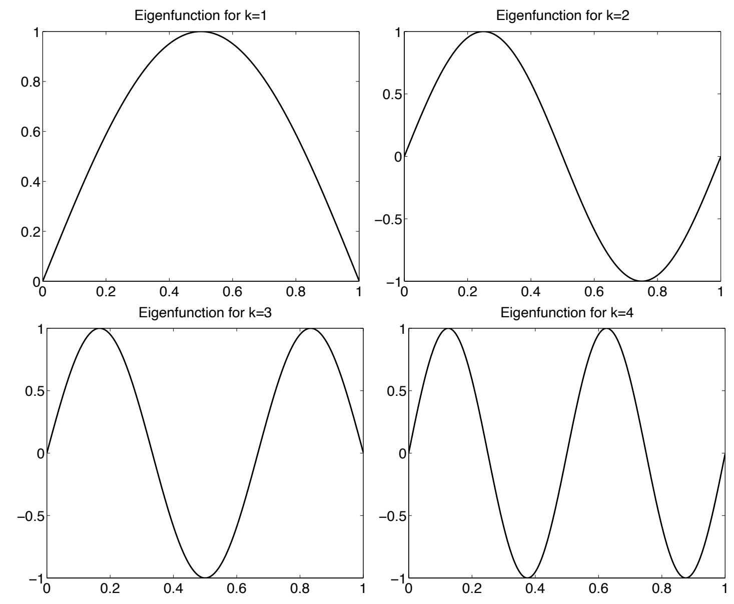

Fig. 1. The first four eigenfunctions.

These equations can be written as the linear system,

$$Au_h = f, \tag{16}$$

where

$$A = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}, \quad u = \begin{pmatrix} u_1 \\ \cdot \\ \cdot \\ \cdot \\ u_N \end{pmatrix}, \quad f = \begin{pmatrix} f_1 \\ \cdot \\ \cdot \\ \cdot \\ f_N \end{pmatrix}. \tag{17}$$

The unknowns u_i are pointwise approximations of u, $u_i \approx u(x_i)$ and $f_i = f(x_i)$. This matrix will be used extensively to explain the behavior of the various algorithms throughout this chapter. Although this matrix is very simple, it has the characteristic properties that will be studied and used throughout most of this chapter.

The eigenvalues of this system are

$$\mu_k = \frac{4}{h^2}\sin^2(\frac{k\pi h}{2}), \quad k = 1, \ldots, N, \tag{18}$$

and the corresponding eigenfunctions are

$$v_{k,i} = v_k(x_i) = \sin(k\pi x_i), \quad k = 1, \ldots, N, \tag{19}$$

where $v_{k,i}$ is the value associated with the node i at the point x_i and the k'th eigenfunction. Similar to the continuous case, the large eigenvalues correspond to high frequency eigenvectors, whereas smooth eigenfunctions are associated with small eigenvalues. This property is exactly what is exploited in the multigrid algorithm which we will discuss below.

A more detailed description of the continuous and discrete Poisson problem can be found in [26].

2.3 The Richardson Iteration Revisited

We saw in Section 2.1 that the Richardson iteration is a memory efficient method to solve a general matrix equation, given a reasonable estimate on $\|A\|$. In order to estimate the computational work needed by the method, we have to estimate the number of floating point operations the CPU has to perform. To this end, we have to estimate how many iterations that will be needed.

The first thing to consider is a *stopping criterion*, designed to prevent endless iterations. We are seeking a numerical approximation u_h of the actual unknown u and have introduced an error, e_h, inherited from the numerical method. In the previous Chapter **??** we saw that the error could be estimated,

$$\|e_h\| = \|u - u_h\| \leq ch^{\alpha},$$

where h is a characteristic grid size parameter directly depending on the number of unknowns. This discretization error e_h determine the level of accuracy needed by the iterative method. Let e_h^n be the error at the n'th iteration. We can split this error into two parts, $e_h^n = e^n + e_h$, where e^n is the part induced by the iterative method and e_h and is the discretization error. It is then reasonable to require that both contributions are equally sized, $e^n \approx e_h$. Hence, we will stop the iteration when $\|e^n\| \leq \|e_h\|$. The concretization of such a stopping criteria is not an easy task. The discretization error, e_h, is of course in general not available. However, the residual can be computed,

$$r^n = b - Au^n.$$

From the residual-error equation

$$Ae^n = r^n,$$

we obtain

$$\|e^n\| \leq \|A^{-1}\|\|r^n\|.$$

Another possible technique is to check $u^n - u^{n-1}$. We will not go deeply into a discussion about various stopping criteria here. However, a frequently used criterion stops the iterating when

$$\|r^n\| < \gamma,$$

where γ is a small number which is usually found by numerical experiments.

The second parameter that determines the number of iterations is the *convergence rate* or minimal error reduction per iteration, which will be explained below. The error at the n'th iteration is governed by

$$e^n = (I - \tau A)e^{n-1}. \tag{20}$$

Given that $\|I - \tau A\| = \rho < 1$, we saw that the iteration was convergent, but we have still not estimated the number of iteration to reach a given stopping criterion. Let the convergence rate ρ be defined as

$$\rho = \|I - \tau A\|.$$

Then

$$\|e^n\| \leq \rho \|e^{n-1}\|. \tag{21}$$

Moreover,

$$\|e^n\| \leq \rho^n \|e^0\|. \tag{22}$$

Assuming that ϵ is the discretization error, we can tolerate an iteration error,

$$\|e^n\| \leq \epsilon \tag{23}$$

If we assume equality in (22) and (23) we get

$$\|e^n\| = \rho^n \|e^0\| = \epsilon \tag{24}$$

Hence, the number of iterations can be estimated as

$$n = \frac{\log \frac{\epsilon}{\|e^0\|}}{\log \rho}. \tag{25}$$

The goal of this chapter is to present algorithms where $\rho \leq c$, where $c < 1$ is a constant independent of the grid size. If the convergence criterion, ϵ is fixed, independent of the grid size, then n will be bounded independent of the grid size. This is refereed to as an *optimal algorithm.*

Finally, we will consider some additional properties that are typical for the matrices we are working with. The matrices are symmetric positive definite and for these matrices it is known that,

$$\|A\| = \lambda_{\max}(A), \quad \|A^{-1}\| = \lambda_{\min}^{-1}(A),$$

where $\lambda_{\max}$ and $\lambda_{\min}$ are the largest and smallest eigenvalues, respectively. From the eigenvalues for the 1D discretization of the Poisson equation (18) we have,

$$\lambda_{\max}(A) = \frac{4}{h^2}, \quad \lambda_{\min}(A) = \pi^2.$$

The ratio between the largest and the smallest eigenvalues is commonly called the condition number of the matrix, $\kappa(A)$,

$$\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}. \tag{26}$$

Hence, the condition number of the matrix A in (17) is $\mathcal{O}(h^{-2})$. Notice that this condition number is typical for elliptic PDEs regardless of the dimension, the boundary conditions and the domain.

These eigenvalues are now used to gain more insight in what will happen during the iteration. Let the initial error be expanded in terms of eigenvectors,

$$e_0 = \sum_{k=1}^{N} c_k w_k,$$

where, by orthonormality, the coefficients are determined by

$$c_k = (e_0, w_k).$$

Lets chose $\tau = \frac{c}{\lambda_{\max}}$. The error at the n'th iteration is then from (20),

$$e^n = \sum_{k=1}^{N} (1 - \frac{c\lambda_k}{\lambda_{\max}})^n c_k w_k.$$

If $0 < c < 2$, then the iteration is convergent in the sense that,

$$\|e^n\| \to 0, \qquad \text{as } n \to \infty.$$

To see this, we note that

$$\begin{aligned}\|e^n\| = (e^n, e^n)^{1/2} &= (\frac{1}{N}\sum_{k=1}^{N}(1 - \frac{c\lambda_k}{\lambda_{\max}})^n c_k w_k, (1 - \frac{c\lambda_k}{\lambda_{\max}})^n c_k w_k) \\ &= (\frac{1}{N}\sum_{k=1}^{N}(1 - \frac{c\lambda_k}{\lambda_{\max}})^{2n} c_k^2) \le (1-c)^{2n}\|e^0\|,\end{aligned}$$

where we have used that $\{w_k\}_{k=1}^N$ are orthogonal.

Still, different parts of the error will decrease at different speed. This is clarified in the following, by considering the error components corresponding to the high and low eigenvalues. Let the initial error, e^0, be dw_N, where d is a constant and w_N is the N'th eigenvector, corresponding to the largest eigenvalue. The error after the first iteration will then be

$$e^1 = (1 - \frac{c\lambda_{\max}}{\lambda_{\max}})w_N = (1-c)e^0.$$

For instance, the choice $c = 0.9$ implies an error reduction by a factor 0.1 for the error component associated with w_N. We also notice that choosing a smaller c leads to slower convergence. Similarly, all the high frequency parts of the error are removed rather efficiently.

However, the situation is quite different for the low frequency parts of the error. Let e^0 be dw_1, where d is a constant and w_1 is the eigenvector corresponding to the lowest eigenvalue. Then we have

$$e^1 = (1 - \frac{c\lambda_{\min}}{\lambda_{\max}})w_1 = (1 - \frac{c}{K(A)})w_1 \approx e^0,$$

where $\kappa(A)$ is the condition number of the matrix. As observed for the 1D discretization of the Poisson equation, the condition number is $\sim h^{-2}$, which is really bad. Moreover, reducing c only makes things worse.

The second disadvantage with the Richardson iteration is that it relies on the estimation of the largest eigenvalue. The iteration will diverge if τ is not properly chosen.

Even if this method is primitive and unusable for large linear systems, it captures the basics of the classical iterations. Some parts of the error is efficiently dealt with, while other remain essentially unchanged. In particular, the smooth components are troublesome. This property will be exploited later, when we consider the (relaxed) Jacobi and Gauss-Seidel methods.

2.4 Preconditioning

An "obvious" generalization of the Richardson iteration is to include a matrix B, usually called a preconditioner, in the following way:

$$u^n = u^{n-1} - \tau B(Au^{n-1} - b). \tag{27}$$

The error iteration will then be

$$e^n = e^{n-1} - \tau BAe^{n-1}, \tag{28}$$

which is convergent if and only if

$$\|I - \tau BA\| < 1. \tag{29}$$

Although this idea seems quite simple, it is remarkably powerful. In fact, nearly all the methods we will consider in the following, multigrid and domain decomposition as well as Jacobi and Gauss-Seidel fit into this framework. It is the general form of any *linear iteration.* The only exception in this chapter is the Conjugate Gradient method.

Notice that we will often refer to B as an operator or matrix. The reason is that if B is a linear iteration it can, in principle, always be represented as a matrix. Still, it is usually easier and more efficient to implement B as an algorithm. To be more specific, only the action of B on a vector, $u = Bv$, has to be implemented.

We will now go briefly go through the classical iterations. These methods are generally not usable to solve matrices with more than 1000 unknowns. Still, they deserve attention, at least for the following four reasons:

- They are used as smoothers in multigrid algorithms.
- They serve to illustrate important aspects of the multigrid method.
- Domain decomposition algorithms are generalizations of the block versions of these algorithms.
- Our final optimal block preconditioner is an inexact variant of the block Jacobi method.

2.5 Jacobi

The simplest algebraic operator splitting technique is the Jacobi method. This algorithm is easily explained by considering the i'th equation in the matrix equation,

$$\sum_{j=1}^{N} a_{ij} u_j = b_i.$$

Rearranging the i'th equation, we get

$$u_i = \frac{1}{a_{ii}}(b_i - \sum_{i \neq j} a_{ij} u_j).$$

The problem is of course that the other unknowns, u_j, are not yet computed. However, we can assume that we have either initial guesses or the values from the previous iteration, which suggests the Jacobi iteration:

$$u_i^n = \frac{1}{a_{ii}}(b_i - \sum_{j \neq i} a_{ij} u_j^{n-1}). \tag{30}$$

Notice that all the variables in (30) can be updated independently. This is important when we use parallel computers. On such computers the unknowns may be distributed on different processors and the update can then be done in parallel. This issue is discussed in Chapter **??**.

The strength of the Richardson iteration (27) now becomes apparent because the Jacobi iteration is included. To see this, consider

$$u^n = u^{n-1} - D^{-1}(Au^{n-1} - b),$$

with $B = D = \mathrm{diag}(A)$ and $\tau = 1$. In the next section this framework will be used to analyze the convergence properties and the corresponding problems with this iteration.

Properties and Problems. To understand when the Jacobi method is convergent we consider the error iteration,

$$e^n = e^{n-1} - D^{-1}Ae^{n-1}. \tag{31}$$

Hence, the iteration is convergent if and only if

$$\rho = \|I - D^{-1}A\| < 1,$$

or

$$0 < \|D^{-1}A\| < 2.$$

It is not easy to understand the behavior of this algorithm directly from these inequalities. We therefore go briefly through the 1D discretization of the Poisson equation. The Jacobi iteration matrix associated with (17) is

$$J = I - D^{-1}A = \frac{1}{2}\begin{pmatrix} 0 & 1 & & & \\ 1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 0 & 1 \\ & & & 1 & 0 \end{pmatrix}.$$

Inserting the eigenfunctions of (17),

$$w_k = \sin(k\pi x_j), \quad k = 1, \dots N. \tag{32}$$

where $x_j = jh$, gives,

$$(I - D^{-1}A)w_k = w_k - D^{-1}\mu_k w_k = (1 - \frac{h^2}{2}\mu_k)w_k,$$

where μ_k is given in (18). Therefore, w_k is an eigenfunction of J, with the corresponding eigenvalue

$$\lambda_k = 1 - \frac{h^2}{2}\mu_k = 1 - \frac{h^2}{2}\frac{4}{h^2}\sin^2(\frac{k\pi h}{2}) = cos(k\pi h), \quad k = 1, \dots, N. \tag{33}$$

The eigenvalues are shown in Figure 2.

Notice that from Figure 2 it seems like the eigenvalues $\lambda_k \in [-1, 1]$. In fact, $\lambda_k \in (-1, 1)$ and the iteration is convergent. However, the convergence rate, determined by $\lambda_{\max}$ and $\lambda_{\min}$ depends strongly on the number of unknowns. To see this we use the Taylor expansion of $\cos(x)$ around $x = \pi$,

$$\cos(x) = \cos(\pi) - \sin(\pi)(x - \pi) - \cos(\pi)(x - \pi)^2 + \dots \tag{34}$$

Letting $x = \frac{N\pi}{N+1}$ in (34), corresponding to the N'th eigenvector in (33), we get

$$\lambda_N = \cos(\frac{N\pi}{N+1}) \approx \cos(\pi) + \cos(\pi)(\frac{\pi N}{N+1} - \pi)^2 = 1 - (\frac{\pi}{N+1})^2 = 1 - (\pi h)^2.$$

The same estimate is obtained for the lowest eigenvalue, with $N = 1$, and using the Taylor expansion around $x = 0$. This means that the error corresponding to the most high frequent eigenfunction, $k = N$, and the most

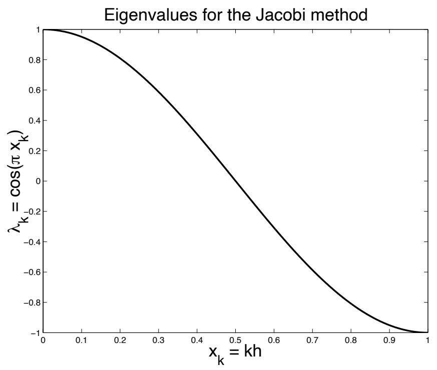

Fig. 2. Eigenvalues of the Jacobi Matrix.

Table 2. Number of iterations for the Jacobi method to achieve an error reduction by a factor 10^{-4} for the 1D discretization of the Poisson problem.

Unknowns	Convergence rate	Iterations
10	$1 - 8.2e^{-2}$	108
100	$1 - 9.7e^{-4}$	9514
1000	$1 - 9.8e^{-6}$	$9.4 \cdot 10^5$
10000	$1 - 9.9e^{-8}$	$9.3 \cdot 10^7$

low frequent eigenfunction, $k = 1$, only decreased by a factor $1 - (\pi h)^2$. The number of Jacobi iterations needed to reduce the error by a factor 10^{-4} is shown in Table 2.5.

Still, for a subset of the eigenfunctions, $\frac{N}{4} \leq k \leq \frac{3N}{4}$, the convergence is fast, in the sense that $\rho < 0.71$. Hence, the Jacobi method performs quite well for at least half of the error components. The iteration would therefore be efficient if the initial error only contained these parts. Naturally, it is hard to construct such an initial guess.

2.6 Relaxed Jacobi

By looking at the Jacobi iteration (30) we see that u_i is computed based on u_j for $j \neq i$. A natural extension is to include information from the previous iteration,

$$u_i^n = (1 - \omega)u_i^n + \frac{\omega}{a_{ii}}(b_i - \sum_{j \neq i} a_{ij} u_j^{n-1}), \tag{35}$$

where ω can be chosen. The inclusion of the data from the previous iteration is commonly called relaxation. As earlier we can express this iteration in terms of the Richardson iteration (3)

$$u^n = u^{n-1} - \omega D^{-1}(Au^{n-1} - b),$$

where $B = D = \operatorname{diag}(A)$ and $\omega = \tau$. The necessary condition for convergence is that

$$0 < \|\omega D^{-1} A\| < 2.$$

Again, we briefly review the properties of this iteration in terms of eigenvalues and eigenfunctions. The eigenfunctions are the same as for Jacobi (32), and a simple calculation shows that the eigenvalues are

$$\lambda_k(\omega) = 1 + \omega(\cos(k\pi h) - 1), \quad k = 1, \dots, n. \tag{36}$$

The eigenvalues for $\omega = 2/3$ are shown in Figure 3. The spectrum has changed from $(-1, 1)$ for $\omega = 1$ to $(-\frac{1}{3}, 1)$ for $\omega = 2/3$. Hence, changing the parameter ω dramatically changes the behavior of the relaxed Jacobi for the high frequency parts of the error. However, low frequency or smooth parts of the error are inefficiently handled for any ω. Still, this is good news, because the high frequency parts of the error are most problematic. A simple idea might be to compute the solution on a coarser grid and use this as the initial guess. Multigrid methods are generalizations of this idea.

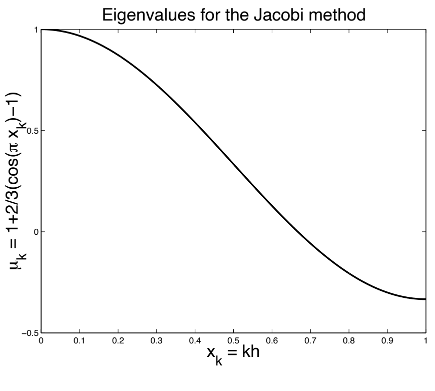

Fig. 3. Eigenvalues of the Relaxed Jacobi iteration Matrix.

2.7 Exact and Inexact Block Jacobi

Another natural extension of the pointwise Jacobi algorithm expressed above constitutes what we will call the block Jacobi method. This iteration arise when replacing the numbers a_{ij}, u_j, and b_j in (30) with the matrices A_{ij}, and the vectors u_j and b_j, respectively. Consider the linear system

$$Au = b,$$

where

$$A = \begin{pmatrix} A_{11} & A_{12} & \cdots & & A_{1N} \\ A_{21} & A_{22} & & & \\ . & \ddots & \ddots & \ddots & \\ . & & & & \\ A_{N1} & & & & A_{NN} \end{pmatrix}, \quad u = \begin{pmatrix} u_1 \\ . \\ . \\ . \\ u_N \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ . \\ . \\ . \\ b_N \end{pmatrix}. \tag{37}$$

Here, A_{ij} are matrices, and x_i and b_i are vectors. The algorithm takes the form,

$$u_i^n = A_{ii}^{-1}(b_i - \sum_{j \neq i} A_{ij} u_j^{n-1}). \tag{38}$$

In order for this method to be applicable we have to assume that the block matrices, A_{ii}, are invertible.

As for the previous Jacobi variants, this iteration can be written in terms of the Richardson iteration (3),

$$u^n = u^{n-1} - D^{-1}(Au^{n-1} - b),$$

where D contains the diagonal blocks matrices A_{ii}. The iteration is convergent if

$$0 < \|D^{-1}A\| < 2.$$

Of course, A_{ii} may be large blocks and in such cases we do not necessarily want to invert A_{ii}, but rather make an approximation of D^{-1}, i.e.,

$$u^n = u^{n-1} - \hat{D}^{-1}(Au^{n-1} - b),$$

The convergence is determined by $\|\hat{D}^{-1}A\|$.

We will postpone the discussion of the merits of this algorithm. However, notice that this framework covers both the optimal preconditioner for the Bidomain model and the domain decomposition method of additive Schwarz type.

2.8 Gauss-Seidel

Looking at the Jacobi iteration, we observe that when computing the unknown u_i^n the unknowns u_j^n for $j < i$ has already been computed. Moreover, u_j^n should be "closer" to the actual solution than u_j^{n-1}. Therefore, it seems natural to modify the Jacobi iteration to use the new values instead. This is the Gauss-Seidel iteration,

$$u_i^n = \frac{1}{a_{ii}}(b_i - \sum_{j<i} a_{ij} u_j^n - \sum_{j>i} a_{ij} u_j^{n-1}). \tag{39}$$

Let

$$A = D + U + L,$$

where D is the diagonal, U and L are the strictly upper and lower diagonal parts of A, respectively. Then the Gauss-Seidel iteration can be written in terms of the Richardson iteration (27),

$$u^n = u^{n-1} - (D + L)^{-1}(Au^{n-1} - b),$$

where $B = (D + L)^{-1}$ and $\tau = 1$.

2.9 Relaxed Gauss-Seidel

The Gauss-Seidel method can be relaxed in the same way as the Jacobi method,

$$u_i^n = (1 - \omega)u^{n-1} + \omega(\frac{1}{a_{ii}}(b_i - \sum_{j<i} a_{ij} u_j^n - \sum_{j>i} a_{ij} u_j^{n-1})). \tag{40}$$

Written in terms of the Richardson iteration, we obtain

$$u^n = u^{n-1} - \omega(D + wL)^{-1}(Au^{n-1} - b).$$

2.10 Symmetric Gauss-Seidel

The Gauss-Seidel method is not symmetric (unless A is a diagonal matrix). However, symmetry is an important property for some solution algorithms such as the Conjugate-Gradient method described later. If the matrix is symmetric, it is rather straightforward to derive a symmetric version of the Gauss-Seidel method. The symmetric Gauss-Seidel simply consist of one standard Gauss-Seidel sweep followed by an additional sweep using the unknowns numbered backwards.

$$\begin{aligned} u_i^{n-1/2} &= \frac{1}{a_{ii}}(b_i - \sum_{j<i} a_{ij} u_j^{n-1/2} - \sum_{j>i} a_{ij} u_j^{n-1}), \text{ for } i = 1, \dots n, \\ u_i^n &= \frac{1}{a_{ii}}(b_i - \sum_{j>i} a_{ij} u_j^n - \sum_{j<i} a_{ij} u_j^{n-1/2}), \text{ for } i = n, \dots 1. \end{aligned}$$

Written in terms of the Richardson iteration,

$$u^n = u^{n-1} - (D + L)^{-1} D (D + U)^{-1}(Au^{n-1} - b).$$

2.11 Block Gauss-Seidel

The block version of Gauss-Seidel method is a straightforward extension of the pointwise Gauss-Seidel. This is similar to the extension we did with block Jacobi. The algorithm is as follows.

$$u_i^n = A_{ii}^{-1}(b_i - \sum_{j<i} A_{ij} u_j^n - \sum_{j>i} A_{ij} u_j^{n-1}). \tag{41}$$

A necessary condition is that A_{ii} is invertible, which was also the case with block Jacobi.

3 The Conjugate Gradient Method

So far we have considered classical iterative methods for solving linear systems of the form

$$Au = f. \tag{42}$$

The classical schemes are easily derived and their motivation is rather simple. In 1952, Hestenes and Stiefel [14] broke this tradition and published a completely different algorithm. Initially, they viewed their scheme as an alternative to Gaussian elimination; i.e. they derived a direct method that would, in exact arithmetics, give the exact solution in at most N iterations. Here N is the number of unknowns in (42) and the matrix A is supposed to be symmetric and positive definite. Later on, it was realized that the approximate solution obtained after much less than N iterations was actually quite good. Soon it became common to use the Conjugate Gradient (CG) method of Hestenes and Stiefel as an iterative method rather than as a direct method.

One of the most important ideas in numerical analysis is that of “best approximation”. Given a large space of functions V and a subspace with fewer functions V_N; how can you find the best approximations $v_N \in V_N$ of the “true” solution $v \in V$. This approach is the basis of the finite element method and, as we shall see, the fundamental idea of the CG method as well. In this text, we shall merely sketch the development of the method. For a full story we refer the reader to e.g. Golub and vanLoan [12] or to Stoer and Bulirsch [21].

3.1 The CG Algorithm

As mentioned above, the CG method seeks the best possible approximation in a certain subspace. To this end, the method consists of two basic ingredients; computing a proper subspace and computing the best approximation in this subspace.

In the CG method, we use two inner products; the standard one

$$(u, v) = \frac{1}{N} \sum_{i=1}^{N} u_i v_i, \tag{43}$$

and the A-inner product

$$(u, v)_A = (Au, v). \tag{44}$$

Since A is symmetric and positive definite, (44) defines an inner product. Similarly, we define the associated norms

$$\|u\| = (u, u)^{1/2}, \tag{45}$$

and

$$\|u\|_A = (Au, u)^{1/2}. \tag{46}$$

The algorithm computes the best solution measured in the A-norm. Furthermore, we will derive a set of search vectors that are A-orthogonal, thus spanning the subset in which we seek an approximate solution. Suppose the subspace of $\mathbb{R}^N$ is denoted by W, and let w be the best approximation of u measured in the A-norm, i.e.

$$\|u - w\|_A \leq \|u - v\|_A, \quad \forall\, v \in W. \tag{47}$$

Then it is generally known that

$$(u - w, v)_A = 0, \quad \forall\, v \in W, \tag{48}$$

i.e. the error is orthogonal to the subspace. This result is fundamental in the derivation of the CG-method.

Let us now assume that we have already computed k search vectors

$$p_0, p_1, \ldots, p_{k-1},$$

which are mutually A-orthogonal, i.e.

$$(p_i, p_j)_A = 0 \quad i \neq j. \tag{49}$$

Let

$$W_k = \text{span}\{p_0, \ldots, p_{k-1}\}, \tag{50}$$

and note that

$$\dim(W_k) = k. \tag{51}$$

We assume that

$$u_k \in W_k, \tag{52}$$

satisfies

$$\|u - u_k\|_A \le \|u - v\|_A, \quad \forall\, v \in W_k, \tag{53}$$

such that $u_k \in W_k$ is the best approximation of the exact solution u measured in the A-norm. It is rather obvious that if W_k spans all of $\mathbb{R}^N$, then $u_k = u$ so we will have the exact solution in at most N iterations.

Let us also define the residual

$$r_k = f - Au_k, \tag{54}$$

which of course is zero if $u_k = u$. The residual has a very interesting property that can be derived from the best approximation property. It follows from (48) and (53) that

$$(u - u_k, v)_A = 0, \quad \forall\, v \in W_k, \tag{55}$$

By using the definition of the A-norm, we get

$$(A(u - u_k), v) = 0, \quad \forall\, v \in W_k, \tag{56}$$

and since $Au = f$, we have

$$(r_k, v) = 0, \quad \forall\, v \in W_k. \tag{57}$$

So r_k is orthogonal to all the vectors in W_k, and in particular

$$(r_k, p_j) = 0, \quad j = 0, 1, \ldots, k-1. \tag{58}$$

We now want to step from iteration k to iteration $k+1$, and in order to do so, we need to increase the dimension of $W_k = \mathrm{span}\{p_0, \ldots, p_{k-1}\}$ and then compute the best approximation in the new and larger subspace. In order to increase the dimension of W_k, we will apply the Gram-Schmidt algorithm. This is an algorithm that computes an orthogonal basis based on linearly independent vectors. Now, we already have k linearly independent, and in fact also orthogonal, vectors. In order to apply the Gram-Schmidt algorithm, we will need a vector that is linearly independent of all vectors in W_k. Since we have already seen that r_k orthogonal to all vectors in W_k we can use this vector to increase the dimension to $k+1$, and then use the Gram-Schmidt orthogonalization process to generate an orthogonal basis. Using this algorithm, we find that

$$p_k = r_k + \beta_{k-1} p_{k-1}, \tag{59}$$

where

$$\beta_{k-1} = -\frac{(r_k, p_{k-1})_A}{(p_{k-1}, p_{k-1})_A}. \tag{60}$$

Now we have $W_{k+1} = \mathrm{span}\{p_0, p_1, \ldots, p_k\}$ and we want to find the best approximation $u_{k+1} \in W_{k+1}$. We seek $u_{k+1} \in W_{k+1}$ of the form

$$u_{k+1} = u_k + \alpha_k p_k, \tag{61}$$

and the task is to determine α_k such that

$$\|u - u_{k+1}\|_A \leq \|u - v\|_A, \quad \forall\, v \in W_{k+1}. \tag{62}$$

Because of (48), we require that

$$(u - u_{k+1}, v)_A = 0, \quad \forall\, v \in W_{k+1}, \tag{63}$$

such that, in particular, we have

$$(u - u_{k+1}, p_k)_A = 0. \tag{64}$$

Using (61), we have

$$\alpha_k = \frac{(u - u_k, p_k)_A}{(p_k, p_k)_A}. \tag{65}$$

Here we notice that

$$(u - u_k, p_k)_A = (A(u - u_k), p_k)_A = (r_k, p_k)_A,$$

such that

$$\alpha_k = \frac{(r_k, p_k)_A}{(p_k, p_k)_A}. \tag{66}$$

By (61) we have

$$u_{k+1} = u_k + \alpha_k p_k, \tag{67}$$

and the derivation is complete.

The CG-method can be formulated in many variants using the orthogonality properties discussed above. All these variants are mathematically equivalent but may behave differently on a computer due to round-off issues. The version we give here has been successfully applied in many practical computations.

The Conjugate Gradient Algorithm

Let $A \in \mathbb{R}^{N,N}$, $f \in \mathbb{R}^N$,$u^0 \in \mathbb{R}^N$, and $0 < \varepsilon < 1$ be given. The matrix A is supposed to be symmetric and positive definite.

```
$u = u^0$
$r = f - Au$
$p = r$
$\rho_0 = (r, r)$
$k = 0$
**While** $\rho_k/\rho_0 > \varepsilon$ **do**
        $z = Ap$                         (68)
        $\gamma = (p, z)$                (69)
        $\alpha = \rho_k/\gamma$         (70)
        $u = u + \alpha p$               (71)
        $r = r - \alpha z$               (72)
        $\rho_{k+1} = (r, r)$            (73)
        $\beta = \rho_{k+1}/\rho_k$      (74)
        $p = r + \beta p$                (75)
        $k = k + 1$                      (76)
**end**
```

The update of r_k is worth noting. Recall that

$$r_k = f - Au_k, \tag{77}$$

since, according to (67),

$$u_{k+1} = u_k + \alpha_k p_k, \tag{78}$$

and we have

$$\begin{aligned} r_{k+1} &= f - Au_{k+1} \\ &= f - A(u_k + \alpha_k p_k) \\ &= f - Au_k - \alpha_k Ap_k \\ &= r_k - \alpha_k Ap_k. \end{aligned} \tag{79}$$

Since we have already computed Ap_k (see (68)), we can avoid an extra matrix-vector multiplication by using (79).

3.2 Convergence Theory

As mentioned above, the CG method is now considered as an iterative scheme and it is important to study the convergence behavior of the method. It turns out that the convergence can be studied in terms of the condition number of the matrix A in (42). Recall that

$$K = K(A) = \frac{\lambda_{\max}}{\lambda_{\min}}, \tag{80}$$

where $\lambda_{\max}$ and $\lambda_{\min}$ are the largest and smallest eigenvalue of A respectively. It is well known, see e.g. Knabner and Angermann [**?**], that the error after k iterations with the CG method can be bounded as follows,

$$\frac{\|e_k\|_A}{\|e_0\|_A} \leq 2 \left(\frac{\sqrt{K}-1}{\sqrt{K}+1} \right)^k, \tag{81}$$

where the error is given by

$$e_k = u - u_k. \tag{82}$$

We observe from (81) that if K is small (close to one), the convergence is very fast, and if K is large the convergence may be very slow. As discussed above, linear systems of the form (42) arising from the discretization of partial difference equations are often poorly conditioned, i.e. K is very large.

Suppose K is large and we want to compute the number of iterations k such that

$$\frac{\|e_k\|_A}{\|e_0\|_A} \leq \varepsilon, \tag{83}$$

for a given ε, $0 < \varepsilon < 1$. Then, from (81) we need

$$k \geq \frac{\ln(\varepsilon/2)}{\ln\left(\frac{\sqrt{K}-1}{\sqrt{K}+1}\right)} \tag{84}$$

iterations. Since K is large, we have

$$\frac{\sqrt{K}-1}{\sqrt{K}+1} \approx 1 - \frac{1}{\sqrt{K}},$$

and since

$$\ln(1+x) = x + O(x^2),$$

we have

$$\ln\left(\frac{\sqrt{K}-1}{\sqrt{K}+1}\right) \approx -1/\sqrt{K},$$

and thus

$$k \gtrsim \ln(2/\varepsilon)\sqrt{K}. \tag{85}$$

This formula explains the importance of the condition number for the convergence of the CG method. We will return to this issue several times below. It is clear that if (85) is sharp estimate, then the number of iterations is $O(\sqrt{K})$ and thus increases quite rapidly as the condition number increases.

3.3 Numerical Experiments

In this section we will present some numerical experiments using the CG method. To this end, we consider the following two problems:

$$-\Delta u_2 = f_2 \quad (x, y) \in \Omega = [0,1]^2, \quad u_2 = 0 \text{ at } \partial\Omega, \tag{86}$$
$$-\Delta u_3 = f_3 \quad (x, y) \in \Omega = [0,1]^3, \quad u_3 = 0 \text{ at } \partial\Omega. \tag{87}$$

We use

$$f_2 = e^{xy}, \tag{88}$$
$$f_3 = e^{xyz}, \tag{89}$$

and discretize (86) and (87) using straightforward finite differences. In a 1D finite difference discretization we use n internal nodes leading to a grid spacing of

$$h = \frac{1}{1+n}. \tag{90}$$

In 2D we use n^2 internal nodes and n^3 internal nodes in 3D. Based on the PDEs above, this leads to linear systems of the form

$$A_2 u_2 = g_2, \tag{91}$$
$$A_3 u_3 = g_3. \tag{92}$$

It can be shown that the condition numbers of A_2 and A_3 are $O(h^{-2})$, see e.g. [**?**]. Since the number of iterations is given by (85), we have $k = O(\sqrt{K}) = O(h^{-1})$. In 2D, the number of nodes is $N \approx 1/h^2$ and in 3D, we have $N \approx 1/h^3$, hence $k_2 \approx c_2 N^{1/2}$ in 2D, and $k_3 \approx c_3 N^{1/3}$ in 3D. Using $\varepsilon = 10^{-7}$, we have applied the CG-method to the systems (91) and (92). The number of iterations are given in Table 3.3 and Table 3.3.

We observe from the tables that the number of iterations is about

$$k_2 \approx 3N^{1/2}$$

in 2D and

$$k_3 \approx 3.5N^{1/3}$$

Table 3. The table shows the number of nodes (N), the number of iterations (k_2), and $c_2 = k_2/N^{1/2}$.

N	k_2	$c_2 = k_2/N^{1/2}$
10 404	287	2.81
40 804	579	2.87
161 604	1167	2.90
643 204	2361	2.94
2 566 404	4770	2.98

Table 4. The table shows the number of nodes (N), the number of iterations (k_3), and $c_3 = k_3/N^{1/3}$.

N	k_3	$c_3 = k_3/N^{1/3}$
1 728	34	2.83
10 648	69	3.14
74 088	138	3.29
551 368	278	3.39
4 251 528	558	3.44

in 3D. Since the amount of work in each iteration is $O(N)$, we have that the solution process requires $O(N^{3/2})$ in 2D and $O(N^{4/3})$ in 3D. The optimal result would be $O(N)$ and we will derive methods that are optimal in this sense later in the text.

4 Multigrid

4.1 Idea

In this section we will first try to motivate multigrid methods with the intuition derived from the classical iterations in the previous section, before we present the polished and abstract framework. Readers interested in the theory of multigrid methods can consult [4] and [13]. More practical introductions to multigrid methods are given in [6] and [24]. An overview of multigrid methods and generalizations thereof can be found in [5] and the references therein.

In the previous section we observed that both the standard and the weighted Jacobi iterations had problems for certain eigenfunctions, but were very good for others. The Jacobi method showed poor performance for eigenfunctions corresponding to either the larger or smaller eigenvalues, whereas weighted Jacobi with $\omega = \frac{2}{3}$ was rather effective for all the higher frequencies. We recall that the error iteration associated with the weighted Jacobi method can be written

$$e^n = e^{n-1} - \omega D^{-1} A e^{n-1}. \tag{93}$$

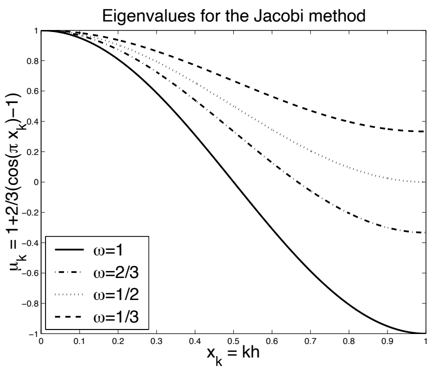

Fig. 4. Eigenvalues of the Jacobi iteration Matrix.

Using the eigenfunction expansion of e^n and (93), we arrived at an estimate for the error after n iterations, e^n, in terms of the eigenvalues for the weighted Jacobi method in (33). We restate the eigenvalues here for convenience,

$$\lambda_k(\omega) = 1 + \omega(\cos(k\pi h) - 1), \quad k = 1, \dots, N. \tag{94}$$

For the high frequencies this results in

$$\max_{k \geq \frac{n}{2}} |\lambda(\frac{2}{3})| \leq \frac{1}{3}. \tag{95}$$

However, the performance is bad for the smooth components. How can the performance be improved ? A natural idea is to combine the strengths of Jacobi with some other method. Earlier we suggested that a coarse grid solution could be used as an initial guess. The initial error would then only contain the high frequency components and the relaxation would be highly efficient. In this section we will develop this idea further, and eventually end up with the multigrid algorithm.

Lets assume that we have performed a number, m, of relaxed Jacobi iterations. The error is then smooth and the continuing iterations barely alter u^m. It is clear that we need another strategy. The first step is to observe that instead of solving

$$Au = b.$$

We can solve

$$Ae_m = r_m,$$

where

$$e_m = u - u_m,$$

and

$$r_m = Au_m - b.$$

The solution u is then obtained as

$$u = u_m + e_m.$$

The multigrid idea is based on the observation that while u can be any function, the error is smooth after a number of relaxations. The error can therefore be well represented on a coarse grid. Moreover, the computations on a coarse grid is of course cheaper than on a fine grid. To clarify this idea, we again consider the 1D discretization of the Poisson problem. The generalization is done afterwards.

In our 1D example it is easy to define a coarse grid. Let the fine grid be

$$\Omega_h = \{x_j = jh, j = 0, 1, \dots, N+1\}.$$

Then a coarser grid can be defined analogously,

$$\Omega_H = \{x_j = jH, j = 0, 1, \dots, M+1\}.$$

An obvious choice is $H = 2h$, which is assumed in the following for simplicity. Hence, instead of solving

$$A_h e_h = r_h,$$

we solve

$$A_H e_H = r_H,$$

because we know that the Jacobi iterations have removed the high frequency error. Given the grid Ω_H we are able to define the coarse grid matrix A_H. But a coarse grid representation of the residual on the fine grid, r_h, is also needed. This can be done by using a restriction operator such that

$$r_H = I_h^H r_h.$$

This notation indicates that I_h^H generates an approximation r_H on the coarse scale of the fine scale version represented by r_h. A suitable choice, in this case, is the weighted restriction operator defined by

$$u_j^H = (I_h^H u^h)_j = \frac{1}{4}(u_{2j-1} + 2u_{2j} + u_{2j+1}), \quad j = 1, \dots, M.$$

With this restriction operator we are able to compute the coarse error

$$Ae_H = r_H = I_h^H r_h.$$

Finally, it is necessary to represent the coarse error on the fine grid. A suitable interpolation is

$$I_H^h = (I_h^H)^T,$$

which, componentwise, reads

$$\begin{aligned} u_{2j}^h &= u_j^H, \quad j = 1, \dots, M, \\ u_{2j+1}^h &= \frac{1}{2}(u_j^H + u_{j+1}^H) \quad j = 0, \dots, M. \end{aligned}$$

4.2 Theoretical Framework

It is now time to generalize the ideas and present the more polished theoretical framework of multigrid methods. Let $\Omega_0 \subset \Omega_1 \subset \ldots \subset \Omega_L$ be a nested sequence of quasi-uniform grids, where Ω_J is the finest grid and Ω_0 is the coarsest. We assume that the coarsest grid is significantly coarser than the finest, such that the cost of the solution of the coarse grid problem can be neglected when compared to a smoothing operation on the finest grid. Furthermore, let V_J be a finite element space on Ω_J consisting of piecewise continuous polynomials of degree r, typically r is 1 or 2. Then $V_0 \subset V_1 \subset \ldots \subset V_L \subset H^1_0$. Notice that this assumption about nested grids and finite element spaces is not essential, but that it simplifies the analysis significantly.

The next thing to notice is that the restriction and interpolation operators come for free when using a finite element discretization. The natural restriction operator is the L_2 projection, $Q_J : L_2 \to V_J$, defined by

$$(Q_J f, N) = (f, N), \quad \forall N \in V_J \text{ and } f \in L_2$$

where $(\cdot,\cdot)$ is the continuous L_2 inner product defined by

$$(f,g) = \int_\Omega fg \, \partial\Omega.$$

This inner product is the natural extension of the discrete L_2 inner product in (7). The interpolation operator is implicitly defined because $V_{J-1} \subset V_J$. In other words, a basis function N_j^{J-1} on the coarse grid Ω_{J-1} can be expressed by a sum of basis functions $\{N_i^J\}$ on the fine grid Ω_J,

$$N_j^{J-1} = \sum_i \alpha_i N_i^J.$$

The next thing to do is to define the linear systems on the different grids. In the previous Chapter **??** we defined the weak formulation of a general elliptic problem,

$$-\nabla \cdot (a(x)\nabla u(x)) + c(x)u(x) = f(x), \text{ in } \Omega, \tag{96}$$
$$u = 0, \text{ on } \partial\Omega. \tag{97}$$

where

$$\inf_{x\in\Omega} a(x) > 0, \quad \inf_{x\in\Omega} c(x) \geq 0,$$

and there exists a real number M such that

$$\sup_{x\in\Omega} a(x) < M, \quad \sup_{x\in\Omega} c(x) < M.$$

We saw in Chapter **??** that the problem (96)-(97) could formally be written as

$$Au = f,$$

with $f \in H^{-1}$ and $A : H_0^1 \to H^{-1}$ was defined by the weak formulation of (96)-(97),

$$(Au, v) = (a\nabla u, \nabla v) + (cu, v), \quad \forall u, v \in H_0^1.$$

The solution $u \in H_0^1$ and is unique.

We are now able to define the linear systems to be solved approximately on the different grids,

$$A_J u_J = f_J,$$

where

$$f_J = Q_J f, \quad A_J = Q_J A.$$

An explicit expression can be found. Let $\{N_j^J\}$ be the finite element basis functions that span V_J then

$$f_j = (f, N_j^J), \quad A_{ij}^J = (AN_i^J, N_j^J).$$

Finally, we need to specify sufficient conditions for the approximate solvers (smoothers) on the different grids to ensure convergence. Let S_J denote a smoother, for instance the weighted Jacobi method would correspond to $S_J = \omega D_J$, with D_J being the diagonal of A_J. As before these smoothers need to be convergent, that is

$$\rho(S_J A_J) \leq \sigma, \tag{98}$$

where $\sigma \subset (0, 2)$. The other condition basically says that the smoother has to be close to A_J^{-1} for the high frequency components. This can be stated as

$$(S_J^{-1} v, v) \leq \alpha(A_J v, v), \quad \forall v \in (I - Q_{J-1})V_J, \tag{99}$$

where $\alpha > 0$. This condition is a generalization of the eigenvalue result for weighted Jacobi (95).

Finally, we state the V-cycle multigrid algorithm. Notice that there are a number of generalizations of this algorithm, e.g., W-cycle and full multigrid, see e.g., [6] and [24].

```
The Multigrid V-cycle Algorithm

I: $B_1 = A_1^{-1}$
II: $B_j g = v^3$ where
    $v^0 = 0$
    $v^1 = v^0 - S_j(A_j v^0 - f)$
    $v^2 = v^1 - B_{j-1} Q_{j-1}(A_j v^1 - f)$
    $v^3 = v^2 - S_j(A_j v^2 - f)$
```

4.3 Convergence Theory

In Section 2.3 we described how the efficiency of iterative methods can be stated in terms of the convergence rate,

$$\rho = \|I - \tau BA\|,$$

and,

$$\|e^n\| \leq \rho\|e^{n-1}\|.$$

For optimal algorithms $\rho \leq c < 1$ independent of h and this leads to a bounded number of iterations independent of h. In the following we set $\tau = 1$ for simplicity.

We used the norm $\|\cdot\|$ for convenience, but in fact, any norm can be used, see [13], Chapter 3. The most common norm in the convergence analysis of multigrid methods (and domain decomposition methods) is the $\|\cdot\|_A$-norm. The A-norm of v is defined as,

$$\|v\|_A = (Av, v)^{1/2},$$

and the corresponding A-norm of a matrix C is

$$\|C\|_A = \sup_v \frac{(Cv, v)_A}{(v, v)_A}.$$

With these norms the convergence rate is

$$\rho_A = \|I - BA\|_A,$$

and the error estimate is

$$\|e^n\|_A \leq \rho_A\|e^{n-1}\|_A.$$

It is shown in [1] and [27] that given that the smoothers S_J satisfy the assumptions (98) and (99), we have $\rho_A < 1$ independent of h. In fact,

$$\rho_A \leq \frac{\alpha}{\alpha + 2}.$$

As a rule of the thumb, the convergence rate $\rho_A \leq \frac{1}{10}$ in the case of a discretization of the Poisson equation on a simple geometry.

Notice that there are many different ways to prove the optimality of multigrid, see for instance [4] and [13], and the references therein.

4.4 Experiments

We will now show an experiment with multigrid. Let the model problem be

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

where the solution is $u = e^{xy}$ and $\Omega = [0,1] \times [0,1]$. The initial guess is a highly oscillating random function. This random function is used to make the stress test for multigrid as hard as possible. It should contain "every possible" error. We use $\frac{\|r_k\|}{\|r_0\|} \leq 10^{-8}$ as the stopping criterion. The number of iterations needed to achieve convergence is shown in Table 4.4. The number of iterations seems to be independent of h, as it should.

Table 5. The number of iterations, n, to achieve convergence with for a Poisson problem with respect to the grid size h.

h	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
n	5	6	6	6	6	6

In Figure 5 the dramatic improvement of the solution during one V-cycle is displayed. It should be clear to anyone that the combination of smoothing and coarse grid correction is powerful.

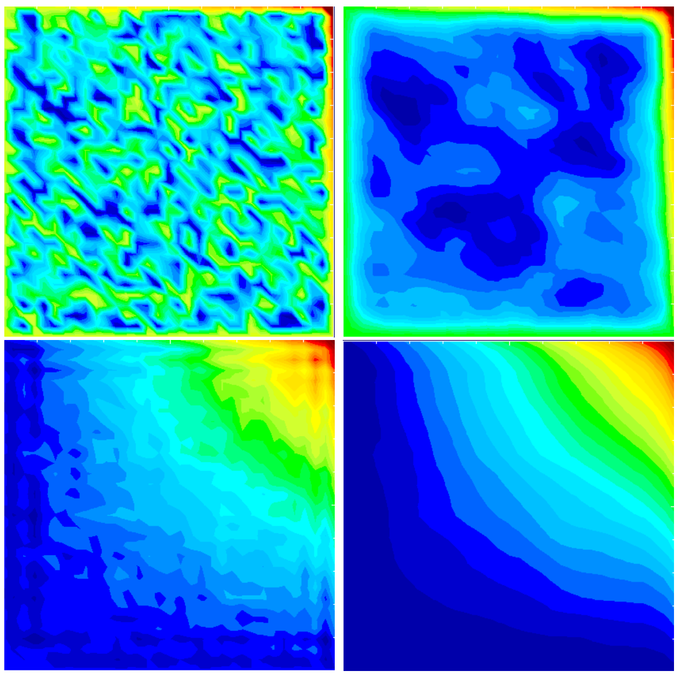

Fig. 5. The upper left picture shows the initial vector. It is a random vector that should contain "all possible" errors. In the upper right picture the solution after one symmetric Gauss-Seidel sweep is displayed. It is clear that the high frequency random behavior in the initial solution has been effectively removed. The picture down to the left shows the solution after the coarse grid correction. The smooth components of the solution have improved dramatically. In the last picture the solution after the post smoothing is displayed. The solution is now very close to the actual solution.

5 Domain Decomposition

Domain decomposition methods have become very popular, in particular during the last two decades, due to the development of parallel computers. Many researchers have worked on this subject and the result is an abstract and mature theoretical foundation, at the same level as multigrid methods. The idea was introduced already in 1870 by Schwarz, who made an algorithm for computing the solution on a compound domain $\Omega = \Omega_1 \cup \Omega_2$, by successively solving similar problems on the simpler domains Ω_1 and Ω_2. Later, in particular from the 1980s and on, researchers discovered that the technique was a powerful algorithm for the efficient solution of discretizations of various PDEs, in particular on parallel computers. For a more detailed description of domain decomposition methods we refer to [9] and [20].

We start this section by reviewing the results from Schwarz's early paper [19], because it is a very good illustration of the method. Schwarz was interested in finding the solution to the following Poisson problem[4]

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

where the domain $\Omega = \Omega_1 \cup \Omega_2$ is depictured in Figure 6.

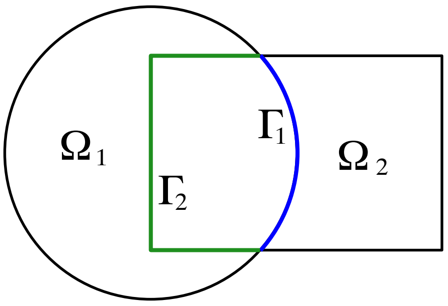

Fig. 6. Schwarz problem domain.

The solution of the Poisson problem on the simpler domains Ω_1 and Ω_2 was known. Schwarz's powerful idea was to make an iteration that repeatedly reuse the solutions in Ω_1 and Ω_2 to get the proper boundary conditions on Γ_1 and Γ_2, and thereby the solution in the compound domain Ω. The algorithm can be summarized as follows. First, the solution in Ω_1 is computed as the solution of the following problem,

$$\begin{aligned} -\Delta u_1^n &= f, \quad && \text{in } \Omega_1, \\ u_1^n &= g, \quad && \text{on } \partial\Omega_1 \backslash \Gamma_1, \\ u_1^n &= u_2^{n-1}, \quad && \text{on } \Gamma_1. \end{aligned}$$

The unknown boundary condition on Γ_1 is based on the previous solution in the domain Ω_2 or an initial guess. Once u_1^n is computed, it is used on the unknown boundary Γ_2 such that u_2^n can be computed by

$$\begin{aligned} -\Delta u_2^n &= f, \quad && \text{in } \Omega_2, \\ u_2^n &= g, \quad && \text{on } \partial\Omega_2 \backslash \Gamma_2, \\ u_2^n &= u_1^n, \quad && \text{on } \Gamma_2. \end{aligned}$$

[4] Schwarz actually studied the Laplace equation, i.e., the Poisson problem with a homogeneous right-hand side, $f = 0$. However, the algorithm works for any f.

After the new solution, u_2^n, is computed, the next iteration starts by the computation of u_1^{n+1} with new boundary conditions based on u_2^n and the iteration continues. Schwarz was able to prove that this iteration converges to the actual solution in Ω.

We will now generalize the alternating Schwarz algorithm to handle many subdomains and make it suitable for parallel computers. However, we will not go into the parallelization here. This is a large field of its own and is covered in Chapter **??**. Instead we will focus on the mathematical properties of the algorithm. Let Ω_h be a triangulation of Ω and furthermore let $\Omega^1, \ldots, \Omega^p$ be an overlapping subdivision of Ω_h, where p will typically equal the number of processors on the parallel computer. The subdomains are usually constructed as follows. Let $\hat{\Omega}^1, \ldots, \hat{\Omega}^p$ be a partition of non-overlapping domains such that $\Omega = \hat{\Omega}^1 \cup \ldots \cup \hat{\Omega}^p$ and $\hat{\Omega}^i \cap \hat{\Omega}^j = 0$ for $i \neq j$. Each of the subdomains $\hat{\Omega}^i$ is then extended with a distance βH such that the subdomains become overlapping. We will come back to the parameters β and H later.

A convenient assumption is that the subdomains are nested in the sense that $\Omega^i \subset \Omega_h$. This assumption is not necessary, but simplifies the exposition. Inherited from the nestedness of the grids there exists a restriction matrix $R_i : \Omega_h \to \Omega^i$, such that

$$R_i(x_j) = 1, \quad \text{if } x_j \in \Omega^i, \tag{100}$$

$$R_i(x_j) = 0, \quad \text{elsewhere.} \tag{101}$$

Here, x_j are the vertices (or nodal points) in the grid Ω_h. The corresponding interpolation operator is simply the transposed of the restriction matrix, R_i^T. Finally, the subdomain matrices are defined by

$$A_i = R_i A R_i^T.$$

With the above definitions of A_i, R_i and R_i^T we can now state the additive Schwarz algorithm.

The Additive Schwarz Algorithm

```
for $i = 1, \ldots, p$:
    $u^{n+1} = u^n - \tau R_i^T A_i^{-1} R_i (A u^n - b)$
```

The additive Schwarz algorithm can be expressed as a preconditioner, B_a, for the Richardson iteration (27), as follows

$$B_a = \sum_{i=1}^{p} R_i^T A_i^{-1} R_i. \tag{102}$$

Notice also that the previously computed u^n is used in each step. This means that the algorithm is parallel by nature. Furthermore, if A is symmetric then so is B_a. It may be necessary to adjustment τ to ensure convergence.

The multiplicative Schwarz algorithm is very similar to the additive Schwarz algorithm. The only difference is that the most recently computed values of u are always used. Also, multiplicative Schwarz is convergent with $\tau = 1$ and therefore the τ parameter is usually avoided.

```
The Multiplicative Schwarz Algorithm

for $i = 1, \dots, p$:
    $u^{n+\frac{i}{p}} = u^{n+\frac{i-1}{p}} - R_i^T A_i^{-1} R_i (A u^{n+\frac{i-1}{p}} - b)$
```

The multiplicative Schwarz can also be represented in terms of a preconditioned Richardson iteration. We have the identity

$$I - B_m A = (I - R_p^T A_p^{-1} R_p) \cdots (I - R_1^T A_1^{-1} R_1), \tag{103}$$

where B_m is the preconditioner in the Richardson iteration (27).

The multiplicative Schwarz algorithm is generally more efficient than the additive Schwarz algorithm on a scalar computer. However, multiplicative Schwarz is a sequential algorithm and is not suitable for parallel computers. In its present shape, the algorithm is not symmetric. But a symmetric version can be made by one standard multiplicative Schwarz iteration followed by an additional iteration with the domains numbered backwards. These algorithms are generalizations of block Jacobi (additive Schwarz) and block Gauss-Seidel (multiplicative Schwarz) to overlapping blocks.

There is one major problem with the algorithms as stated above. The efficiency of the algorithms depends on the number of subdomains p. In fact, the efficiency deteriorates quite rapidly as p increases. A cheap, but efficient solution to the problem of the p dependency is to introduce a coarse grid Ω_H, with characteristic grid size H. This grid can be very coarse (in contrast to multigrid methods). We refer to the matrix on the coarse grid as A_H and the restriction operator is R_H. Notice that the restriction operator is not on the form (100)-(101), but is instead made similar to the restriction operator used for multigrid methods.

The above definition of the Schwarz algorithms can easily employ coarse grid correction. Instead of the previous numbering where $\Omega^1, \dots, \Omega^p$ is a overlapping partition of Ω_h, we number the overlapping domains as $\Omega^2, \dots, \Omega^{p'}$, where $p' = p + 1$. In addition, the coarse grid is used, let $\Omega^1 = \Omega_H$. With this numbering of the domains and the corresponding matrices A_i and R_i, the above algorithms extend directly to the case with a coarse grid, given that p is replaced with p'.

The following results are known for the additive and multiplicative Schwarz preconditioners. For the additive Schwarz preconditioner with a coarse grid correction, B_a, the condition number of $B_a A$ is independent of h. In fact,

$$K(B_a A) \leq C(1 + \beta^{-1}).$$

Notice that this does not mean that the additive Schwarz algorithm with coarse grid correction is necessarily convergent. However, τ can be chosen such that the convergence rate,

$$\rho_A = \|I - \tau B_a A\|_A < 1,$$

independently of h and H. The choice of τ is dealt with in more detail later in Section 6.

On the other hand the multiplicative Schwarz method with coarse grid correction is convergent and the convergence rate, $\rho_A < 1$, is independent of h and H, where

$$\rho_A = \|I - B_m A\|_A.$$

The proof can be found in e.g, [9] and [20]. Without a coarse grid correction the convergence rate typically deteriorates as $\mathcal{O}(H^{-2})$.

There are also non-overlapping domain decomposition algorithms where the underlying domains are not overlapping. Moreover, the domains may not even be motivated geometrically. We will not go into these algorithms here, instead we refer to e.g., [9] and [20].

6 Preconditioning Revisited

6.1 Idea

We have now introduced the classical iterations, the multigrid method, domain decomposition and the Conjugate Gradient method as separate methods. It is now time to glue them together to a framework suitable for handling the equations of the Bidomain model. The gluing concept is *preconditioning.*

As before, we want to solve the linear system,

$$Au = b. \tag{104}$$

We have seen earlier that the efficiency of both the Richardson and the Conjugate Gradient method depends on the condition number of A, which was typically of order h^{-2}. The idea of preconditioning is simply to multiply (104) with a matrix or operator B to obtain an equivalent system,

$$BAu = Bb, \tag{105}$$

where B is usually called the *preconditioner.* The system (105) has the same solution as (104) provided that B has full rank[5]. The hope is then that (105) is easier to solve than (104). The preconditioner B can be any matrix or operator that in some sense resembles A^{-1}. For instance, it may be a sweep with the Jacobi or the Gauss-Seidel iterations or it may be based on an

[5] A matrix has full rank if it is invertible.

approximate factorization of A. What is crucial is that B should be a good approximation of A^{-1} and also cheap in storage and evaluation.

Earlier we considered optimal solution algorithms, and we will therefore describe *optimal preconditioners*. First, we need the concept of spectral equivalence, which describes what a "good" approximation of A^{-1} is.

6.2 Spectral Equivalence

Notice that A and B are families of matrices with respect to the triangulations Ω_h, where h approaches zero in the limit. It is therefore common to let A and B have the subscript h, A_h and B_h.

The two operators or matrices that are assumed to be symmetric and positive definite[6], A_h^{-1} and B_h, are spectrally equivalent (independent of h) if there exist constants c_1 and c_2, independent of h such that

$$c_1(A_h^{-1}v, v) \leq (B_h v, v) \leq c_2(A_h^{-1}v, v), \quad \forall v. \tag{106}$$

Alternatively, we may express this property as

$$c_1(A_h v, v) \leq (A_h B_h A_h v, v) \leq c_2(A_h v, v), \quad \forall v. \tag{107}$$

It is known that the condition number of $B_h A_h$ is bounded by the constants c_1 and c_2,

$$K(B_h A_h) \leq \frac{c_2}{c_1}. \tag{108}$$

It is common to denote spectral equivalence of A_h and B_h^{-1} as $A_h \sim B_h^{-1}$. With the definition of spectral equivalence we can now state how a well designed preconditioner should be:

- B_h should be spectrally equivalent to A_h^{-1},
- the evaluation of B_h on a vector v, $B_h v$, should cost $\mathcal{O}(N)$ operations,
- the storage of B_h should be similar to the storage of A_h, $\mathcal{O}(N)$ floating point numbers.

In the following we will see that an optimal preconditioner leads to an optimal solution algorithm, both in the case of the Richardson and the Conjugate Gradient methods.

6.3 The Richardson Iteration Re-Revisited

We can now derive what the spectral equivalence of B and A^{-1} means in the context of the preconditioned Richardson iteration. We remember from (28) that the error at the n iteration, e^n, could be stated in terms of the error at the previous iteration, e^{n-1},

$$e^n = (I - \tau BA)e^{n-1}. \tag{109}$$

[6] If A_h and B_h are symmetric and positive definite then so are A_h^{-1} and B_h^{-1}.

The convergence rate in the A-norm is

$$\rho_A = \|I - \tau BA\|_A, \tag{110}$$

and the error behavior could be estimated as

$$\|e^n\|_A \leq \rho_A \|e^{n-1}\|_A.$$

Because BA is symmetric with respect to the A inner product, ρ_A can be stated in terms of the eigenvalues of BA, μ_i,

$$\rho_A = \|I - \tau BA\|_A = \sup_{\mu_i} |1 - \tau\mu_i|.$$

Hence, ρ_A is a linear polynomial in μ_i and its maximum is obtained at the extreme values μ_i,

$$|1 - \tau\mu_0| \quad \text{or} \quad |1 - \tau\mu_N|,$$

where μ_0 and μ_N are the smallest and largest eigenvalues, respectively. We choose τ as the minimizer of $|1 - \tau\mu_i|$. The minimum is obtained when

$$1 - \tau\mu_0 = \tau\mu_N + 1.$$

which makes

$$\tau = \frac{2}{\mu_0 + \mu_N}$$

the optimal choice. With this choice of τ, ρ_A is

$$\rho_A = 1 - \tau\mu_0 = 1 - \frac{2\mu_0}{\mu_0 + \mu_N} = \frac{\mu_N - \mu_0}{\mu_N + \mu_0} = \frac{K-1}{K+1}, \tag{111}$$

and we have the corresponding error estimate,

$$\|e^n\|_A \leq (\frac{K-1}{K+1})^n \|e^0\|_A.$$

Hence, when A and B^{-1} are spectrally equivalent, the condition number K is independent of h, and therefore the convergence rate, ρ_A, is independent of h.

6.4 Preconditioned Conjugate Gradient Method

In this section we will extend the Conjugate Gradient method with a preconditioner. This is nearly always done in practice, at least when CG is used to compute the numerical solution of PDE problems. First, observe that BA does not need to be symmetric even if A and B are. Symmetry is required to ensure convergence of the Conjugate Gradient method, as we saw in Section 3. However, the algorithm can be stated in any inner product. The inner product of choice when using a preconditioner is $(\cdot,\cdot)_{B^{-1}}$ defined as

$$(u, v)_{B^{-1}} = (B^{-1}u, v).$$

The preconditioner B is positive definite and symmetric, which means that B^{-1} also is positive definite and symmetric. Consequently, also B^{-1} defines an inner product. Moreover, BA is obviously symmetric in the B^{-1} inner product, because

$$(BAu, v)_{B^{-1}} = (Au, v) = (u, Av) = (u, BAv)_{B^{-1}}.$$

Still, we do not want to form neither B^{-1} nor BA. In fact, when using either multigrid or domain decomposition as preconditioner, the only available action is the evaluation on a vector, Bv. Fortunately, the Conjugate Gradient method applied to BA in the B^{-1} inner product can be stated similar to the Conjugate Gradient algorithm at page 22. The only difference is the use of an additional vector and the evaluation of B.

```
*The Preconditioned Conjugate Gradient Algorithm*
Let $A \in \mathbb{R}^{N,N}$, $B : \mathbb{R}^N \to \mathbb{R}^N$, $f \in \mathbb{R}^N$,$u^0 \in \mathbb{R}^N$, and $0 < \varepsilon < 1$
be given. The matrix $A$ and the preconditioner $B$ are supposed to
be symmetric and positive definite.
$u = u^0$
$r = f - Au$
$s = Br$
$p = s$
$\rho_0 = (s, r)$
$k = 0$
**While** $\rho_k/\rho_0 > \varepsilon$ **do**
        $z = Ap$                          (112)
        $t = Bz$                          (113)
        $\gamma = (p, z)$                 (114)
        $\alpha = \rho_k/\gamma$          (115)
        $u = u + \alpha p$                (116)
        $r = r - \alpha z$                (117)
        $s = s - \alpha t$                (118)
        $\rho_{k+1} = (s, r)$             (119)
        $\beta = \rho_{k+1}/\rho_k$       (120)
        $p = s + \beta p$                 (121)
        $k = k + 1$                       (122)
**end**
```

6.5 Convergence Analysis

The convergence analysis in Section 3 extends directly to the case with a preconditioner, but the condition number is $K = K(BA)$ instead of $K(A)$,

$$\|e^n\|_A \leq (\frac{\sqrt{K}-1}{\sqrt{K}+1})^n \|e^0\|_A.$$

In the case where $B \sim A^{-1}$, $K \leq \frac{c_2}{c_1}$ by (108) and is bounded independently of h. This is always an improvement over the preconditioned Richardson iteration,

$$\frac{\sqrt{K}-1}{\sqrt{K}+1} = \frac{K-1}{K+2\sqrt{K}+1} < \frac{K-1}{K+1}.$$

However, if K is small, then the improvement is not large. This is the case for Poisson type equations, because the multigrid methods or domain decomposition methods are highly efficient. In fact, in the case of multigrid preconditioning, an convergence rate of about $\frac{1}{10}$ should be expected. Using the relation (111) the condition number can be estimated to $K \approx 1.2$, which makes $\sqrt{K} \approx 1.1$. Moreover, each iteration with the Conjugate Gradient method is slightly heavier than an Richardson iteration and it is therefore not much to gain by accelerating an efficient multigrid method with the Conjugate Gradient method. However, if robustness is an issue, then the preconditioned Conjugate Gradient method should be considered. This will be exemplified in the next section.

6.6 Variable Coefficients

In this section we will consider the case of elliptic equations with variable coefficients. The reason for this study is that the electrical conductivity in the heart and body vary spatially. In fact, the ratio between the largest and the smallest conductivities may be as large as 100, which is the ratio between blood and bone. Here, we will investigate how such variations affect the multigrid performance. The model problem is

$$-\nabla \cdot (a\nabla u) = f, \quad \text{in } \Omega, \tag{123}$$
$$u = 0, \quad \text{on } \partial\Omega, \tag{124}$$

where $a(x)$ is a matrix that describes the electrical conductivity of the media. Let the corresponding linear system be

$$Au = b.$$

Both multigrid and domain decomposition have problems removing the error associated with the jumps in a. In fact, multigrid may even diverge, depending on the jump and the geometry of a.

On the other hand, the preconditioned Conjugate Gradient method always converge, given that the matrix is positive definite and symmetric, which it is in this application. However, the efficiency of the Conjugate Gradient method depends on the condition number. We know from the sections 4 and 5 that efficient multigrid and domain decomposition algorithms can be made for discretizations of the Poisson problem

$$-\Delta u = f, \text{ in } \Omega, \tag{125}$$

$$u = 0, \text{ on } \partial\Omega. \tag{126}$$

The corresponding linear system is denoted

$$Cu = c.$$

In this case we know that it is possible to construct a preconditioners, B, which is spectrally equivalent with the inverse,

$$c_0(Bv, v) \le (C^{-1}v, v) \le c_1(Bv, v), \quad \forall v.$$

Furthermore, C and A are spectrally equivalent, because

$$a_{\min} \int_\Omega \nabla u \nabla v \, dx \le \int_\Omega a \nabla u \nabla v \, dx \le a_{\max} \int_\Omega \nabla u \nabla v \, dx.$$

Finally, because $A \sim C \sim B^{-1}$, we end up with the following estimate on the condition number,

$$K(BA) = \frac{k_1}{k_0} \frac{c_1}{c_0}.$$

Table 6. The number of iteration with respect to a and h

$h\backslash a$	1	10^2
10^2	-	-
10^3	-	-
10^4	-	-
10^5	-	-
10^6	-	-

7 The Monodomain Model

Recalling the equation for the Monodomain model introduced in Chapter **??**,

$$\begin{aligned} u_t - \nabla \cdot (a \nabla u) &= f(u), \quad && \text{in } \Omega, t > 0, \\ u(0) &= u_0, && \text{in } \Omega, t = 0, \\ u(x) &= 0, && \text{on } \partial\Omega, \end{aligned}$$

where u is the transmebrane potential, a is a scaled conductivity, and $f(u)$ is a nonlinear function. One way to solve the equation is to split it in two parts,

$$u_t = \nabla \cdot (a\nabla u), \quad \text{in } \Omega, t > 0, \tag{127}$$

and

$$u_t = f(u), \quad \text{in } \Omega, t > 0, \tag{128}$$

and assign suitable boundary and initial conditions to these equations.

The main reason for performing the splitting is that efficient solution algorithms exist for both (127) and (128). Solution algorithms for (128) are described in Chapter **??**. Here, we will focus on (127) and consider both multigrid and domain decomposition.

Using either implicit Euler or Crank-Nicholson for the time discretization and either FDM or FEM in space, we need to solve the following linear system at each time step,

$$(I - \Delta t A)u^k = u^{k-1}. \tag{129}$$

Notice, that this linear system corresponds to the discretization of a reaction-diffusion equation,

$$u - \epsilon \Delta u = f, \tag{130}$$

where $\epsilon << 1$. One of the characteristics of this equation is that boundary layers may be present as ϵ approaches zero. This causes the solution u to be less regular than for standard elliptic equations, which again causes a deterioration of the approximation on coarse grids. The quality of the coarse grid correction is important for both multigrid and domain decomposition, but should in the case of (130) be expected to decrease. This is not the case for the system (129), due to the particular right-hand side, which is the solution at the previous time step, u^{k-1}.

In the following sections we will describe multigrid and domain decomposition preconditioners that are independent of both h and Δt, in the sense that,

$$c_0(Bu, u) \leq ((I - \Delta t A)^{-1}u, u) \leq c_1(Bu, u), \quad \forall u,$$

where c_0 and c_1 are independent of h and Δt.

Finally, we remark that the difference between two time steps decreases as $\Delta t \to 0$,

$$\|u^k - u^{k-1}\| \leq C\Delta t.$$

Therefore, the solution from the previous time step, $k-1$, provides a very good start vector at the next time step, k. With this start vector the error reduction required by the linear solvers is only $\sim \Delta t$. Hence, when using iterative methods, the linear system at each time step becomes cheaper to solve as Δt decreases.

7.1 Multigrid

The multigrid methods considered in Section 4 extend to the problem (129) c.f. [3], [16] and [23] and are independent of both Δt and h. This is described in the following.

As earlier, we have a sequence of nested quasi-uniform grids, $\Omega_0 \subset \Omega_1 \subset \ldots \subset \Omega_L$. For each grid the corresponding linear system is

$$(I - \Delta t A_J)u_J^k = u_J^{k-1}, \quad J = 0, \ldots, L. \tag{131}$$

The smoothers, restriction and interpolation operators are defined as in Section 4.

In Section 4 we saw that multigrid is an efficient algorithm for solving the following equation

$$cu - \nabla \cdot (a \nabla u) = f, \tag{132}$$

where c and a are bounded below and above. However, we did not mention the case where $a \to 0$. This is the case here. Let us start by considering the situation when $a = 0$, (In this case the solution has already been found at the previous time step and there is no need to solve a linear system. Look aside from this point.) The solution of (132) reduces to solving a linear system where the matrix is a mass matrix. This can be done optimally with standard classical iterations, because the mass matrix has a condition number independent of h and is diagonally dominant.

Hence, in the limit case of $a = 0$, the coarse grid correction is not needed. This is the general case with this equation. The performance of the smoothers improves as Δt decreases. In fact, it has been shown in [16] that the improved smoothing balance the loss of the approximation on the coarse grid, such that multigrid is an optimal method independently of Δt, even for the harder case of the reaction-diffusion equation. In Table 7.1 we see that the performance improves as Δt decreases for (129).

Table 7. The number of iterations with respect to Δt and h.

$h \backslash \Delta t$	1	10^2	10^4	10^6
10^2	-	-	-	-
10^3	-	-	-	-
10^4	-	-	-	-
10^5	-	-	-	-
10^6	-	-	-	-

7.2 Domain Decomposition

The domain decomposition methods considered in Section 5 extend to the problem (129) c.f. [7], [8] and [9] and are independent of both Δt and h. This is described in the following.

As earlier in Section 5, the domain is divided into p overlapping subdomains, $\Omega_h = \Omega_h^1 \cup \ldots \cup \Omega_h^p$, with an additional coarse grid Ω_H. The restriction and interpolation operators, R_i and R_i^T, can be derived from the geometrical relation between the subdomain Ω_h^i and Ω_h, with no reference to the matrix to be solved. Therefore, these operators can be used also in this application. This also applies to the restriction and interpolation operators for the coarse grid. Therefore, the subdomain matrix is constructed as,

$$(I - \Delta t A)_i = R_i (I - \Delta t A) R_i^T .$$

With these subdomain matrices, the Schwarz algorithms in Section 5 can be used directly.

In the previous section, where we considered the multigrid methods, we noticed that the smoothers improved as $\Delta t \to 0$. Also, for domain decomposition methods, the error appears to be become more local as Δt decreases. In fact, if $\Delta t \leq C H^2$ then the coarse grid solver is not necessary, see [7] and [8].

8 The Bidomain Model

In the previous sections we have seen that it is possible to make optimal preconditioners for both of the matrices A and $I - \Delta t A$, where A was similar to a discrete Laplacian. However, the Bidomain model contains a system of PDEs, where each component is either A or $I - \Delta t A$. It is not easy to extend the theoretical framework developed for A and $I - \Delta t A$ to this model problem directly, but this is a possibility. This is not what we will do here. Instead, we will show that these components can be reused in the powerful concept of (block) preconditioning, to give an order optimal preconditioner.

To clarify the description we restate the equations that was derived in Chapter **??**,

$$\begin{aligned} v_t &= \nabla \cdot (M_i \nabla v) + \nabla \cdot (M_i \nabla u), \\ 0 &= \nabla \cdot (M_i \nabla v) + \nabla \cdot ((M_i + M_e) \nabla u). \end{aligned}$$

In Chapter **??**, these equations were discretized by the finite element method in space and a Crank-Nicholson scheme in time, such that we arrive at the following system of algebraic equations,

$$\begin{aligned} I v^n + \frac{\Delta t}{2} A_i v^n + \frac{\Delta t}{2} A_i u^n &= I v^{n-1} - \frac{\Delta t}{2} A_i v^{n-1} - \frac{\Delta t}{2} A_i u^{n-1}, \\ \frac{\Delta t}{2} A_i v^n + \frac{\Delta t}{2} A_{i+e} u^n &= -\frac{\Delta t}{2} A_i v^{n-1} - \frac{\Delta t}{2} A_{i+e} u^{n-1}. \end{aligned}$$

Recall that the matrices A_i and A_{i+m} are discretizations of the above differential operators, i.e.,

$$A_i = (\nabla \cdot M_i \nabla)_h,$$
$$A_{i+e} = (\nabla \cdot M_{i+e} \nabla)_h,$$

and I is the mass matrix in the finite element method. Later we will also need the discrete Laplacian,

$$A_0 = \Delta_h.$$

The right-hand sides are not of particular importance when considering preconditioners. In fact, we construct the preconditioners such that they handle any right-hand side. That is, letting b^{n-1} and c^{n-1} to be the respective right-hand sides,

$$\begin{pmatrix} b^{n-1} \\ c^{n-1} \end{pmatrix} = \begin{pmatrix} Iv^{n-1} - \frac{\Delta t}{2} A_i v^{n-1} - \frac{\Delta t}{2} A_i u^{n-1} \\ -\frac{\Delta t}{2} A_i v^{n-1} - \frac{\Delta t}{2} A_{i+e} u^{n-1} \end{pmatrix},$$

then we can write (133) as

$$\begin{pmatrix} I + \frac{\Delta t}{2} A_i & \frac{\Delta t}{2} A_i \\ \frac{\Delta t}{2} A_i & \frac{\Delta t}{2} A_{i+e} \end{pmatrix} \begin{pmatrix} v^n \\ u^n \end{pmatrix} = \begin{pmatrix} b^{n-1} \\ c^{n-1} \end{pmatrix}. \tag{133}$$

The matrix in (133) is symmetric and positive definite. This makes the Conjugate Gradient method an appropriate iterative solver, when combined with a suitable preconditioner. In the following we will describe a block preconditioner, where the blocks are constructed by preconditioners for A_i and $I + \frac{\Delta t}{2} A_i$. These have been described earlier in this chapter.

Block preconditioners have been considered before, in particular for the Navier-Stokes equations in e.g., [10], [11], [18], and [25]. More general block preconditioners can be found in [2] and [15]. Applications to the Bidomain model are described in [17] and [22].

We will now describe an optimal preconditioner for the Bidomain model. The matrix in (133) is

$$T = \begin{pmatrix} I + \frac{\Delta t}{2} A_i & \frac{\Delta t}{2} A_i \\ \frac{\Delta t}{2} A_i & \frac{\Delta t}{2} A_{i+e} \end{pmatrix}. \tag{134}$$

In the following we will show that T is spectrally equivalent to the inverse of the block Jacobi preconditioner. Let

$$S = \begin{pmatrix} I + \frac{\Delta t}{2} A_0 & 0 \\ 0 & \frac{\Delta t}{2} A_0 \end{pmatrix}. \tag{135}$$

The exact Jacobi preconditioner will then be S^{-1}, but of course inverting S is too costly. However, in Section 4, Section 5 and Section 7 optimal preconditioners for A_0 and $I + \frac{\Delta t}{2} A_0$ were described. Therefore, we know that an order optimal preconditioner for S can be made. This is indeed the main motivation behind constructing the preconditioner this form. Let this preconditioner be R, defined as

$$R = \begin{pmatrix} (\widehat{I + \frac{\Delta t}{2} A_0})^{-1} & 0 \\ 0 & (\widehat{\frac{\Delta t}{2} A_0})^{-1} \end{pmatrix}. \tag{136}$$

Spectral equivalence is assosiative, in the sense that if A, B, and C are three matrices and $A \sim B$ and $B \sim C$, then $A \sim C$. We have that $R^{-1} \sim S$. Therefore, it remains to show that $S \sim T$, and this would imply that $R^{-1} \sim T$. Then, we can deduce that the condition number of the preconditioned system is bounded[7],

$$K(RT) \leq c. \tag{137}$$

where c is independent of h and Δt.

Hence, the concept of block preconditioning allow us to build an optimal preconditioner by reusing standard algorithms based on multigrid and domain decomposition for linear scalar PDEs that are either elliptic or parabolic. These methods have been studied extensively in the literature.

The last piece of the puzzle is to prove that S and T are spectrally equivalent, $S \sim T$, c.f. Section 6.2. This is done in the rest of this section.

Assumptions
We assume that there exists constants c_0 and c_1, independent of h and Δt, such that

$$c_0(A_0 v, v) \leq (A_\alpha v, v) \leq c_1(A_0 v, v), \tag{138}$$

for $\alpha = i, e$.

The assumption simply state that the matrix generated by the Laplace operator is spectrally equivalent to the matrices generated by the variable coefficient problems.

Notice that, since

$$(A_{i+e} u, u) = (A_i u, u) + (A_e u, u),$$

we have from (138) that

$$2c_0(A_0 v, v) \leq (A_{i+e} u, u) \leq 2c_1(A_0 v, v). \tag{139}$$

The proof is split in two parts, the upper and the lower bound. We start with the upper bound.

[7] Notice that we assume that S, T and R^{-1} are spectrally equivalent independent of h and Δt, c.f. Section 6.2.

Upper bound:
Let

$$W = \begin{pmatrix} v \\ u \end{pmatrix}.$$

We start by proving that

$$(TW, W) \leq c(SW, W), \quad \forall W,$$

for a suitable choice of c (which is independent of h and Δt). First we compute a more direct expression for (TW, W),

$$\begin{aligned}
(TW, W) &= \left(\begin{pmatrix} I + \frac{\Delta t}{2} A_i & \frac{\Delta t}{2} A_i \\ \frac{\Delta t}{2} A_i & \frac{\Delta t}{2} A_{i+e} \end{pmatrix} \begin{pmatrix} v \\ u \end{pmatrix}, \begin{pmatrix} v \\ u \end{pmatrix} \right) \\
&= \left(\left((I + \tfrac{\Delta t}{2} A_i) v + \tfrac{\Delta t}{2} A_i u, \tfrac{\Delta t}{2} A_i v + \tfrac{\Delta t}{2} A_{i+e} u \right), \begin{pmatrix} v \\ u \end{pmatrix} \right) \\
&= ((I + \frac{\Delta t}{2} A_i) v, v) + \Delta t (A_i v, u) + \frac{\Delta t}{2} (A_{i+e} u, u).
\end{aligned}$$

The corresponding expression for the preconditioner is

$$\begin{aligned}
(SW, W) &= \left(\begin{pmatrix} I + \frac{\Delta t}{2} A_0 & 0 \\ 0 & \frac{\Delta t}{2} A_0 \end{pmatrix} \begin{pmatrix} v \\ u \end{pmatrix}, \begin{pmatrix} v \\ u \end{pmatrix} \right) \\
&= \left(\left((I + \tfrac{\Delta t}{2} A_0) v, \tfrac{\Delta t}{2} A_0 u \right) \begin{pmatrix} v \\ u \end{pmatrix} \right) \\
&= ((I + \frac{\Delta t}{2} A_0) v, v) + \frac{\Delta t}{2} (A_0 u, u).
\end{aligned}$$

By using (138) and (139), we get

$$\begin{aligned}
(TW, W) &\leq ((I + \frac{\Delta t}{2} A_i) v, v) + \frac{\Delta t}{2} (A_{i+e} u, u) + \Delta t (A_i v, u) \\
&\quad + \frac{\Delta t}{2} (A_i (v - u), (v - u)) \\
&= ((I + \frac{\Delta t}{2} A_i) v, v) + \frac{\Delta t}{2} (A_{i+e} u, u) + \Delta t (A_i v, u) \\
&\quad + \frac{\Delta t}{2} (A_i v, v) + \frac{\Delta t}{2} (A_i u, u) - \Delta t (A_i v, u) \\
&= ((I + \Delta t A_i) v, v) + \Delta t (A_i u, u) + \frac{\Delta t}{2} (A_e u, u) \\
&\leq c_1 ((I + \Delta t A_0) v, v) + c_1 \Delta t (A_0 u, u) + c_1 \frac{\Delta t}{2} (A_0 u, u) \\
&\leq 3 c_1 \left\{ ((I + \frac{\Delta t}{2} A_0) v, v) + \frac{\Delta t}{2} (A_0 u, u) \right\}.
\end{aligned}$$

Hence

$$(TW, W) \leq 3 c_1 (SW, W). \tag{140}$$

Lower bound

$$\begin{aligned}(TW,W) &= ((I+\frac{\Delta t}{2}A_i)v,v)+\Delta t(A_iv,u)+\frac{\Delta t}{2}(A_{i+e}u,u)\\ &= (Iv,v)+\frac{\Delta t}{2}(A_i(v+u),v+u)+\frac{\Delta t}{2}(A_eu,u)\\ &\geq c_0(v,v)+\frac{\Delta tc_0}{2}(A_0(v+u),v+u)+\frac{\Delta tc_0}{2}(A_0u,u)\\ &\geq c_0(v,v)+\frac{\Delta tc_0}{2}(A_0(v+u),v+u)+\frac{\Delta tc_0}{2}(A_0u,u)\\ &\quad -\frac{\Delta tc_0}{2}(A_0(\varepsilon u+\frac{1}{\varepsilon}v),\varepsilon u+\frac{1}{\varepsilon}v).\end{aligned}$$

where $\varepsilon>0$ is to be determined. Here

$$(A_0(\varepsilon u+\frac{1}{\varepsilon}v),\varepsilon u+\frac{1}{\varepsilon}v)=\varepsilon^2(A_0u,u)+\frac{1}{\varepsilon^2}(A_0v,v)+2(A_0v,u),$$

so

$$\begin{aligned}(TW,W) &\geq c_0\Big\{(v,v)+\frac{\Delta t}{2}(A_0v,v)+\frac{\Delta t}{2}(A_0u,u)\\ &\quad +\Delta t(A_0v,u)+\frac{\Delta t}{2}(A_0u,u)-\frac{\Delta t}{2}\varepsilon^2(A_0u,u)\\ &\quad -\frac{\Delta t}{2\varepsilon^2}(A_0v,v)-\Delta t(A_0v,u)\Big\}\\ &= c_0\left\{(v,v)+\frac{\Delta t}{2}(1-\frac{1}{\varepsilon^2})(A_0v,v)+\frac{\Delta t}{2}(2-\varepsilon^2)(A_0u,u)\right\}.\end{aligned}$$

By picking $\varepsilon^2=3/2$, we have $1-\frac{1}{\varepsilon^2}=1/3$ and $2-\varepsilon^2=1/2$, and then

$$\begin{aligned}(TW,W) &\geq c_0\left\{(v,v)+\frac{\Delta t}{6}(A_0v,v)+\frac{\Delta t}{4}(A_0u,u)\right\}\\ &\geq \frac{c_0}{3}\left\{((I+\frac{\Delta t}{2}A_0)v,v)+\frac{\Delta t}{2}(A_0u,u)\right\}\\ &= \frac{c_0}{3}(SW,W). \qquad (141)\end{aligned}$$

Summarizing the results from (140) and (141), we have

$$\frac{c_0}{3}(SW,W)\leq(TW,W)\leq 3c_1(SW,W).$$

Hence, T and S are spectrally equivalent. This imply that the condition number of $S^{-1}T$ is bounded. In fact,

$$\kappa(S^{-1}T)=9\frac{c_1}{c_0}.$$

References

1. D. N. Arnold, R. S. Falk, and R. Winther. Preconditioning in H(div) and applications. *Math. Comp. 66*, 1997.
2. D.N. Arnold, R.S. Falk, and R. Winther. Preconditioning discrete approximations of the Reissner–Mindlin plate model. M^2AN, 31:517–557, 1997.
3. R. E. Bank and T. Dupont. An optimal order process for solving finite element equations. *Math. Comp.*, 36:35–51, 1981.
4. J. H. Bramble. *Multigrid Methods*, volume 294 of *Pitman Research Notes in Mathematical Sciences*. Longman Scientific & Technical, Essex, England, 1993.
5. A. Brandt. Multiscale scientific computation review 2001. In T. J. Barth, T. F. Chan, and R. Haimes, editors, *Multiscale and Multiresolution Methods: Theory and Applications*. Springer, 2001.
6. W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. SIAM Books, 2nd edition, 1996.
7. X.-C. Cai. Additive schwarz algorithms for parabolic convection-diffusion equations. *Numer. Math.*, 60:41–62, 1991.
8. X.-C. Cai. Multiplicative schwarz methods for parabolic problems. *SIAM J. Sci. Comput.*, 15(3):587–603, 1994.
9. T. F. Chan and T. P. Mathew. Domain decomposition algorithms. *Acta Numerica*, pages 61–143, 1994.
10. H. C. Elman. Preconditioners for saddle point problems arising in computational fluid dynamics. *Appl. Numer. Math.*, 43:75–89, 2002.
11. H. C. Elman, D. J. Silvester, and A. J. Wathen. Block preconditioners for the discrete incompressible Navier–Stokes equations. *Int. J. Numer. Meth. Fluids*, 40:333–344, 2002.
12. G.H. Golub and C.F. van Loan. *Matrix Computations*. Johns Hopkins University Press, 1989.
13. W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*. Springer-Verlag, 1994.
14. M.R. Hestenes and E. Stiefel. Methods of conjungate gradients for solving linear systems. *J. Res. Nat. Bur. Standards*, 49:409–436, 1952.
15. A. Klawonn. An optimal preconditioner for a class of saddle point problems with a penalty term. *SIAM J. Sci. Comput.*, 19:540–552, 1998.
16. M. A. Olshanskii and A. Reusken. On the convergence of a multigrid method for linear reaction-diffusion problems. *Computing*, 65(3):193–202, 2000.
17. M. Pennacchio and V. Simoncini. Efficient algebraic solution of reaction-diffusion systems for the cardiac excitation process. *Journal of Computational and Applied Mathematics*, 145:49–70, 2000.
18. T. Rusten and R. Winther. A preconditioned iterative method for saddle point problems. *SIAM J. Matrix Anal.*, 13:887–904, 1992.
19. H. A. Schwarz. Gesammelte Mathematische Abhandlungen. *Springer, Berlin*, 2:133–143, 1890. First published in *Vierteljahrsschrift Naturforsch, Ges. Zürich*, 15:272-286, 1870.
20. B. F. Smith, P. E. Bjørstad, and W. D. Gropp. *Domain decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.
21. J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer-Verlag, 1993.

22. J. Sundnes, G. T. Lines, K. A. Mardal, and A. Tveito. Multigrid block preconditioning for the coupled bidomain and forward problem. *Computer Methods in Biomechanics and Biomedical Engineering*, 5 (6):397–409.
23. V. Thomée. *Galerkin Finite Element Methods for Parabolic Problems.* Springer-Verlag, 2nd edition, 1997.
24. U. Trottenberg, C. Oosterlee, and A. Schuller. *Multigrid.* Academic Press, 2001.
25. S. Turek. *Efficient Solvers for Incompressible Flow Problem.* Springer Verlag, 1999.
26. A. Tveito and R. Winther. *Introduction to Partial Differential Equations – A Computational Approach.* Springer-Verlag, 1998.
27. R. Winther. Iterative methods for partial differential equations, part ii. Lecture notes in IMPD.