Numerical Linear Algebra

Comparison of Several Numerical Methods used to solve Poisson's Equation

Christiana Mavroyiakoumou

University of Oxford

A case study report submitted for the degree of

M.Sc. in Mathematical Modelling and Scientific Computing

Hilary 2017

1 Introduction

In this report, we apply the finite difference scheme to the Poisson equation with homogeneous Dirichlet boundary conditions. This yields a system of linear equations with a large sparse system matrix that is a classical test problem for comparing direct and iterative linear solvers.

The solution of sparse linear systems by iterative methods has become one of the core applications and research areas of scientific computing. The size of systems that are solved routinely has increased tremendously over time. This is because the discretisation of partial differential equations can lead to systems that are arbitrarily large. We introduce the reader to the general theory of regular splitting methods and present some of the classical methods such as Jacobi, Gauss-Seidel, Relaxed Jacobi, SOR and SSOR.

Iterative methods yield the solution $\mathbf{U}$ of a linear system after an infinite number of steps. At each step, iterative methods require the computation of the residual of the system. In this report, we use iterative solution methods in order to solve

$$A\mathbf{U} = \mathbf{f}, \quad A \in \mathbb{R}^{n\times n},\ \mathbf{f} \in \mathbb{R}^n. \tag{1.1}$$

Linear systems can be solved by fixed point iteration. To do so, one must transform the system of linear equations into a fixed point form and this is in general achieved by splitting the matrix A into two parts, $A = M - N$. Assuming that M is an invertible matrix, this splitting induces an iterative method as follows: Given an initial approximation $\mathbf{U}^{(0)}$ we can compute $\mathbf{U}^{(k)}$, for $k \geqslant 1$, by solving the system of equations

$$M\mathbf{U}^{(k+1)} = N\mathbf{U}^{(k)} + \mathbf{f}, \quad k \geqslant 0, \tag{1.2}$$

and expect that $\mathbf{U}^{(k)}$ will converge to the desired solution. It is clear that the purpose of these iterative methods is to increase efficiency by choosing the matrix splitting in such a way that solving linear systems with the matrix M requires fewer operations than for the original system. In the extreme case that $M = A$ and thus $N = 0$, we converge in one iteration to the solution. So the choice of splitting is contingent upon M being a good approximation of A and also $Mx = y$ being easy to solve.

The iteration matrix of the method is $G = M^{-1}N$, and $\mathbf{b} = M^{-1}\mathbf{f}$. In this report, we compare the performance of the `backslash` command, five classical iterative methods, the Conjugate Gradient method and finally the two-grid algorithm.

2 Finite Difference Scheme

The objective of this report is to solve Poisson's equation on the unit square:

$$-\nabla^2 u = f(x,y) \quad \text{in} \quad \Omega = (0,1)\times(0,1) \subset \mathbb{R}^2, \tag{2.1}$$

$$u = 0 \quad \text{on} \quad \partial\Omega. \tag{2.2}$$

We solve this elliptic equation numerically using finite difference methods [LeV07]. In general, one has grid spacings $\Delta x = \frac{1}{N}$ and $\Delta y = \frac{1}{M}$ in the two directions and a mesh of the form

$$x_i = i\Delta x \ \text{ for } \ i = 0,\ldots,N \quad \text{and} \quad y_j = j\Delta y \ \text{ for } \ j = 0,\ldots,M. \tag{2.3}$$

Let $u_{i,j}$ represent an approximation to $u(x_i, y_j)$. The central difference approximations for the second derivatives lead to the difference formula

$$-\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} - \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} + \mathcal{O}[(\Delta x)^2, (\Delta y)^2] = f(x_i, y_j), \tag{2.4}$$

and $u_{i,0} = u_{i,M} = 0$, $1 \leqslant i \leqslant N-1$ and $u_{0,j} = u_{N,j} = 0$, $0 \leqslant j \leqslant M$ (so that we do not repeat the corners). The finite difference scheme is represented by the *5-point stencil* as shown in Figure 1.

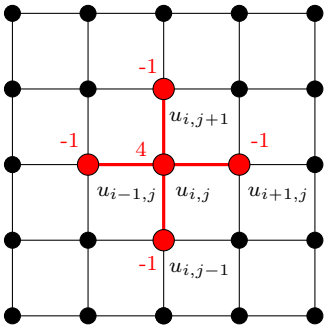


Figure 1: Portion of the computational grid for a two-dimensional elliptic equation. The 5-point Laplacian about the point (i,j) is indicated in red.

Note that instead of the classical row-wise ordering, one could use the red-black ordering which corresponds to an odd-even ordering. Since all 4 neighbours of a red point on the grid are black points, this means that the structure of the matrix equation would be different. For more details on this, see [LeV07, pp. 62].

This can be rewritten as a matrix problem of the form $A\mathbf{U} = \mathbf{f}$ and for this there are two approaches one could follow. One is to write $(N+1)(M+1)$ equations as above for the $(N+1)(M+1)$ unknowns. The other approach is to eliminate $u_{i,j}$

corresponding to boundary nodes so we solve for the $(N-1)(M-1)$ unknowns. For example, when $i = 1$ in finite difference equation we have

$$-\frac{u_{2,j} - 2u_{1,j} + \cancelto{0}{u_{0,j}}}{(\Delta x)^2} - \frac{u_{1,j+1} - 2u_{1,j} + u_{1,j-1}}{(\Delta y)^2} = f(x_1, y_j). \tag{2.5}$$

To simplify notation we assume that $\Delta x = \Delta y = h$ and so $M = N$. However, it is easy to handle the general case as well. We see that the vector of unknowns is partitioned as $\mathbf{U} = (u_{1,1}, u_{1,2}, \ldots, u_{1,N-1}, u_{2,1}, \ldots, u_{N-1,1}, \ldots, u_{N-1,N-1})^\top$. We order $\mathbf{f}$ in a similar way. Then we have $A\mathbf{U} = \mathbf{f}$ where $A \in \mathbb{R}^{(N-1)^2 \times (N-1)^2}$ is given by

$$A = \left.\begin{bmatrix} B & C & & & 0 \\ C & B & C & & \\ & \ddots & \ddots & \ddots & \\ & & C & B & C \\ 0 & & & C & B \end{bmatrix}\right\} (N-1) \text{ blocks} \tag{2.6}$$

with

$$B = \frac{1}{h^2}\begin{bmatrix} 4 & -1 & & & 0 \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ 0 & & & -1 & 4 \end{bmatrix} \quad \text{and} \quad C = -\frac{1}{h^2}\begin{bmatrix} 1 & & & & 0 \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ 0 & & & & 1 \end{bmatrix}, \tag{2.7}$$

where B, $C \in \mathbb{R}^{(N-1)\times(N-1)}$.

We input the matrix A into MATLAB and experiment with different ways for constructing it as efficiently as possible. We conclude that A can be represented in a succinct way using the sum of two Kronecker products. That is, $A = D_{xx} \oplus D_{yy} = D_{xx} \otimes I + I \otimes D_{yy}$, where D_{xx} and D_{yy} are one-dimensional discrete Laplacians in the x- and y- directions, respectively, and I are the identity matrices with the appropriate sizes. For more details see Appendix A.

2.1 Model Problems

The best way to test a numerical method for solving a partial differential equation is to use it on an equation with a known analytical solution. Using the same equation to test various numerical methods and comparing their performance, can help determine which method is the most efficient. In this report we look at two specific model problems and we make appropriate comparisons to draw meaningful conclusions. We assume that the domain Ω is $(0,1) \times (0,1) \subset \mathbb{R}^2$.

1. The first problem we consider is

$$-\nabla^2 u = 13\pi^2 \sin(2\pi x)\sin(3\pi y) \quad \text{in } \Omega, \tag{2.8}$$
$$u = 0 \quad \text{on } \partial\Omega.$$

   The exact solution of (2.8) is given by $u(x, y) = \sin(2\pi x)\sin(3\pi y)$.

2. The second problem we consider is

$$-\nabla^2 u = -(x-1)^3(42x^2 - 24x + 2)y(y-1) - 2x^2(x-1)^5 \quad \text{in } \Omega, \tag{2.9}$$
$$u = 0 \quad \text{on } \partial\Omega.$$

   The exact solution of (2.9) is given by $u(x, y) = (x-1)^5 x^2 y(y-1)$.

In order to check the convergence of the finite difference scheme we solve the matrix problem using MATLAB's `backslash` command.

2.2 Solution of Matrix Problems using `backslash`

In this section, we invoke MATLAB's built-in function `backslash` "\" to solve the matrix system we have constructed. This operator takes advantage of the sparsity of matrix A. We record the maximum error at a node of the mesh and see how it converges as we increase the number of mesh spacings in each coordinate direction. From Figure 2 it is clear that the method converges for both problems with the expected rate $\mathcal{O}(h^2)$.

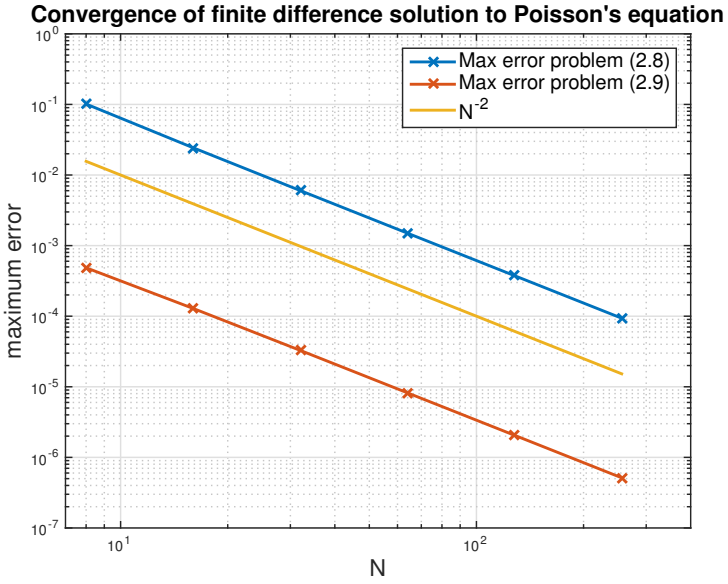


Figure 2: The convergence of the finite difference solutions to (2.8) and (2.9).

3 Classical Iterative Solution Methods

In this section we include an overview of several iterative methods that are used to solve large sparse linear systems as a result of discretising elliptic equations of the form (2.1). We introduce classical methods such as Jacobi, Relaxed Jacobi, Gauss-Seidel, SOR and SSOR. All these iterative methods are based on the idea that the matrix can be split into $A = M - N$. In order to be able to analyse the convergence properties of an iterative method we first need to give the following definition.

Definition 3.1 (Spectral Radius of Matrix A)**.** *The spectral radius of A is given by $\rho(A) = \max\{|\lambda| : \lambda \text{ is an eigenvalue of } A\}$.*

In general, given $A = M - N$ and (1.2) we can now state a general convergence result for iterative methods as it appears in [Saa03, pp. 104, § 4.2.1].

Theorem 3.2. *The iterative method $M\boldsymbol{U}^{(k+1)} = N\boldsymbol{U}^{(k)} + \boldsymbol{f}$ converges for any initial guess $\boldsymbol{U}^{(0)}$ to the solution $\boldsymbol{U}$ of the linear system $A\boldsymbol{U} = \boldsymbol{f}$ if and only if $\rho(M^{-1}N) < 1$.*

We will also write

$$\left[\ A\ \right] = \left[\ L\ \right] + \left[\ D\ \right] + \left[\ U\ \right]$$

where L is the lower triangular part, D is the diagonal and U is the upper triangular part of A. We now introduce the five iterative methods and their convergence theory.

3.1 Jacobi Method

Jacobi's method computes new values of $\mathbf{U}$ based on data from the previous iteration.

Algorithm 1 Jacobi Method

1: **Input:** matrix $A \in \mathbb{R}^{m\times m}$; and vector $\mathbf{f} \in \mathbb{R}^m$.
2: **for** iterates $k = 1, 2, \ldots$
3: **for** rows (equations) $i = 1, \ldots, n$
4: $U_i^{(k)} = \frac{1}{a_{ii}} \left(- \sum_{j=1, j\neq i}^{n} a_{ij} U_i^{(k-1)} + f_i \right)$
5: **end**
6: **end**

The Jacobi method corresponds to the splitting $A = M - N$, with $M = D$ and $N = -(L + U)$.

Therefore, we have the following

$$D\mathbf{U}^{(k)} = -(L+U)\mathbf{U}^{(k-1)} + \mathbf{f}, \tag{3.1}$$

which implies that the Jacobi iteration matrix is of the form $G_{\text{J}} = -D^{-1}(L+U)$.

Theorem 3.3 (Convergence of Jacobi)**.** *If the matrix $A \in \mathbb{R}^{n\times n}$ is strictly diagonally dominant, i.e. $|a_{ii}| > \sum_{j\neq i} |a_{ij}|$ for $i = 1, \ldots, n$, then the Jacobi iteration (3.1) converges.*

To simplify the notation, let us assume that we have the same discretisation in the x- and y-direction and so we denote $h = \frac{1}{N}$. The eigenvalues of the Jacobi iteration matrix G_{J} can be shown to be $\lambda_{\text{J}}^{r,s} = \frac{1}{2}\left[\cos(r\pi h) + \cos(s\pi h)\right]$ and so for the Jacobi method the spectral radius of the iteration matrix is given by

$$\rho(G_{\text{J}}) = \cos(\pi h) = 1 - \frac{\pi^2 h^2}{2} + \mathcal{O}(h^4). \tag{3.2}$$

The convergence, thus, gets worse as h gets smaller. Note that as $h \to 0$ ($N \to \infty$) then $\rho(G_{\text{J}}) \to 1$. Since the error is multiplied by the spectral radius at each step, the convergence slows down.

3.2 Relaxed Jacobi Method

A generalisation of the Jacobi method is the so-called *Relaxed Jacobi* method. A relaxation parameter, ω, is introduced and the method is as follows:

Algorithm 2 Relaxed Jacobi Method

1: **Input:** matrix $A \in \mathbb{R}^{m\times m}$; and vector $\mathbf{f} \in \mathbb{R}^m$.
2: **for** iterates $k = 1, 2, \ldots$
3: **for** rows $i = 1, \ldots, n$
4: $$U_i^{(k)} = \omega\left(f_i - \sum_{j=1, j\neq i}^{n} a_{ij}U_i^{(k-1)}\right)\frac{1}{a_{ii}} + (1-\omega)U_i^{(k)}$$
5: **end**
6: **end**

For $\omega Ax = \omega b$, $\omega \in \mathbb{R}^+$, $M = D$, $N = (1-\omega)D - \omega(L+U)$ our iteration matrix is

$$M^{-1}N = (1-\omega)I - \omega D^{-1}(L+U). \tag{3.3}$$

If Relaxed Jacobi's method is used, then the eigenvalues of the iteration matrix are given by

$$\lambda_{\text{RJ}}^{r,s} = (1-\omega) + \frac{\omega}{2}\left[\cos(r\pi h) + \cos(s\pi h)\right]. \tag{3.4}$$

In particular, if we choose $\omega = \frac{1}{2}$ we have $\lambda_{\mathrm{RJ}}^{r,s} = \frac{1}{2} + \frac{1}{4}[\cos(r\pi h) + \cos(s\pi h)]$, and so all the eigenvalues lie in the interval $(0, 1)$. The corresponding eigenvectors for $M^{-1}N$ have entries $v_{i,j}^{r,s} = \sin(ri\pi h)\sin(sj\pi h)$. Thus high frequency eigenvectors (r, s are large), correspond to small eigenvalues. The optimal choice though is $\omega = \frac{2}{3}$ (see [LeV07, pp. 106]) and we will use this value when we present some numerical results.

3.3 Gauss-Seidel Method

Even though the Jacobi method is very slow, it is not difficult to improve it. The *Gauss-Seidel* method differs to Jacobi's method in that at the k-th step the available values of $U_i^{(k)}$ are being used to update the solution. The algorithm becomes:

Algorithm 3 Gauss-Seidel Method

1: **Input:** matrix $A \in \mathbb{R}^{m\times m}$; and vector $\mathbf{f} \in \mathbb{R}^m$.
2: **for** iterates $k = 1, 2, \ldots$
3: **for** rows $i = 1, \ldots, n$
4: $U_i^{(k)} = \frac{1}{a_{ii}} \left(-\sum_{j=1}^{i-1} a_{ij} U_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} U_j^{(k-1)} + f_i \right)$
5: **end**
6: **end**

This corresponds to $A = M - N$, with $M = D + L$ and $N = -U$ to give

$$(D + L)\mathbf{U}^{(k)} = -U\mathbf{U}^{(k-1)} + \mathbf{f}. \tag{3.5}$$

The spectral radius of the iteration matrix for Gauss-Seidel is given by

$$\rho(G_{\mathrm{GS}}) = \cos^2(\pi h) = 1 - \pi^2 h^2 + \mathcal{O}(h^4). \tag{3.6}$$

This means that the Gauss-Seidel method generally converges about twice as fast as the Jacobi method. For more details, see [Dem97, Cor. 6.1].

3.4 Successive Over-Relaxation Method (SOR)

Like the Jacobi method, the Gauss-Seidel method also admits a modification based on adjustments to the correction vector. Here, we introduce the *Successive Over-Relaxation method (SOR)*, which is derived from Gauss-Seidel by introducing a parameter ω. It uses an acceleration procedure to increase the rate of convergence. The component $U_i^{(k)}$ is computed as for Gauss-Seidel but then averaged with its previous value.

Algorithm 4 Successive Over-Relaxation Method (SOR)

Input: matrix $A \in \mathbb{R}^{m\times m}$; and vector $\mathbf{f} \in \mathbb{R}^m$.
for iterates $k = 1, 2, \ldots$
    for rows $i = 1, \ldots, n$
        $$U_i^{(k)} = \omega \left(f_i - \sum_{j=1}^{i-1} a_{ij} U_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} U_j^{(k-1)} \right) \frac{1}{a_{ii}} + (1-\omega) U_i^{(k-1)}$$
    end
end

We use again the Gauss-Seidel iteration (3.5) but now we multiply it by ω and add on both sides of the resulting expression $(1-\omega)D\mathbf{U}$, to obtain the SOR iteration [GGK14, pp. 695]

$$(D + \omega L)\mathbf{U}^{(k)} = \omega \mathbf{f} + [(1-\omega)D - \omega U]\mathbf{U}^{(k-1)}. \tag{3.7}$$

Note that if we let $\omega = 1$ then we recover the Gauss-Seidel algorithm, $\omega < 1$ corresponds to under-relaxation and $\omega > 1$ to over-relaxation. The choice of the relaxation parameter ω is not arbitrary. The following theorem [GGK14, Th. 11.9] states the condition that ω must satisfy to ensure convergence of the SOR algorithm.

Theorem 3.4 (Kahan, [Kah58])**.** *Let $A \in \mathbb{R}^{n\times n}$ and $A = L + D + U$, where D is an invertible diagonal matrix. If the SOR iteration matrix takes the form $G_{SOR} = (D + \omega L)^{-1}[(1-\omega)D - \omega U]$, then the inequality*

$$\rho(G_{SOR}) \geqslant |\omega - 1| \tag{3.8}$$

holds for all $\omega \in \mathbb{R}$.

Using Theorem 3.2 we conclude that for convergence of SOR we need $\rho(G_{\text{SOR}}) < 1$. Together with Theorem 3.4, this implies that it is necessary to choose $\omega \in (0, 2)$.

According to [Dem97, Th. 6.7], the optimal relaxation parameter for SOR is given by $\omega_{\text{opt}} = \frac{2}{1+\sqrt{1-\rho(G_{\text{J}})^2}} = \frac{2}{1+\sin(\pi h)} \approx 2 - 2\pi h$, where $\rho(G_{\text{J}}) = \cos(\pi h)$ and the optimal spectral radius of SOR is $\rho(G_{\text{SOR}}) = \omega_{\text{opt}} - 1 = \frac{\rho(G_{\text{J}})^2}{[1+\sqrt{1-\rho(G_{\text{J}})^2}]^2}$, or equivalently

$$\rho(G_{\text{SOR}}) = \frac{\cos^2(\pi h)}{[1+\sin(\pi h)]^2} = 1 - 2\pi h + \mathcal{O}(h^2), \tag{3.9}$$

where we have used the series expansion of $\cos(\pi h)$ and $\sin(\pi h)$ for small h and also the binomial expansion. Even with this optimal ω we see that $\rho(G_{\text{SOR}}) \to 1$ only linearly in h as $h \to 0$, rather than quadratically as with the Jacobi method or with Gauss-Seidel. This makes a substantial difference in practice.

3.5 Symmetric Successive Over-Relaxation Method (SSOR)

It is sometimes useful to preserve symmetry. Therefore, if the matrix A is symmetric (which implies that $U = L^\top$) then we can use the *Symmetric Successive Over-Relaxation (SSOR)* method instead.

Algorithm 5 Symmetric Successive Over-Relaxation Method (SSOR)

Input: matrix $A \in \mathbb{R}^{m\times m}$; and vector $\mathbf{f} \in \mathbb{R}^m$.
for iterates $k = 1, 2, \ldots$
  for rows $i = 1, \ldots, n$
    $U_i^{\left(k-\frac{1}{2}\right)} = \omega \left(f_i - \sum_{j=1}^{i-1} a_{ij} U_j^{\left(k-\frac{1}{2}\right)} - \sum_{j=i+1}^{n} a_{ij} U_j^{(k-1)} \right) \frac{1}{a_{ii}} + (1-\omega) U_i^{(k-1)}$
    $U_i^{(k)} = \omega \left(f_i - \sum_{j=i+1}^{n} a_{ij} U_j^{(k)} - \sum_{j=1}^{i-1} a_{ij} U_j^{\left(k-\frac{1}{2}\right)} \right) \frac{1}{a_{ii}} + (1-\omega) U_i^{\left(k-\frac{1}{2}\right)}$
  end
end

This takes the form:

$$(D + \omega L)\mathbf{U}^{\left(k-\frac{1}{2}\right)} = \omega \mathbf{f} + \left[(1-\omega)D - \omega U\right]\mathbf{U}^{(k-1)} \tag{3.10}$$

$$(D + \omega U)\mathbf{U}^{(k)} = \omega \mathbf{f} + \left[(1-\omega)D - \omega L\right]\mathbf{U}^{\left(k-\frac{1}{2}\right)}. \tag{3.11}$$

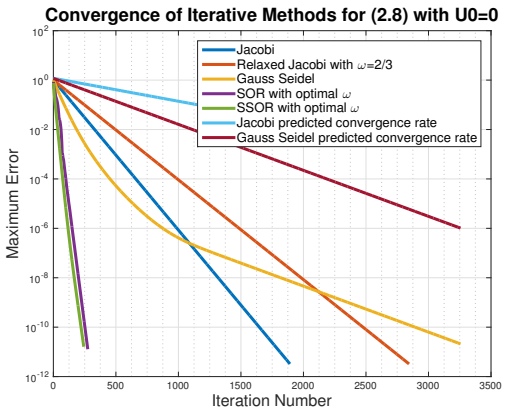
Thus, it can be shown that M from the matrix splitting is symmetric and is given by

$$M = \frac{1}{\omega(2-\omega)}(D + \omega L)D^{-1}(D + \omega U). \tag{3.12}$$

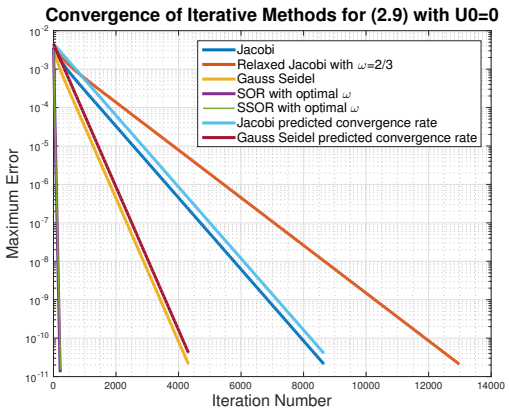
Provided that $\rho(LU) < \frac{1}{4}$, even with $\omega_{\text{opt}} = \frac{2}{1+\sqrt{2[1-\rho(G_{\text{J}})]}} \approx 2 - 2\pi h$, the SSOR method converges more slowly than SOR since $\rho(G_{\text{SSOR}}) \leqslant 1 - \pi h$ [Axe96, pp. 294].

3.6 Comparison of Convergence Rates of Iterative Methods

We use $N = 48$ and a stopping criterion in which the iterative method terminates when $||A\mathbf{U}^{(k)} - \mathbf{f}||_\infty < \texttt{TOL}$. In particular, we choose this tolerance to be $\texttt{TOL} = 10^{-8}$. In Figure 3, we plot the convergence of solutions computed using the iterative methods as a function of the iteration number. Here, we used $\mathbf{U}^{(0)} = \mathbf{0}$ for all methods, and the optimal relaxation parameters: $\omega = \frac{2}{3}$ for Relaxed Jacobi, and $\omega = 2 - 2\pi h$ for SOR and SSOR [Dem97, pp. 299]. Moreover, one observes that the relative rates of convergence of the iterative methods agree with the theory we have presented in the previous subsections.



(a) Model Problem 1: (2.8).



(b) Model Problem 2: (2.9).

Figure 3: The convergence of the iterative methods to the exact solution of the matrix problems (computed using `backslash` in MATLAB) using $\mathbf{U}^{(0)} = \mathbf{0}$.

	Number of iterations for convergence				
Model Problem	Jacobi	RJ[1] (ω_{opt})	Gauss-Seidel	SOR (ω_{opt})	SSOR(ω_{opt})
(2.8)	1892	2845	3256	275	243
(2.9)	8650	12980	4319	214	225

Table 1: Number of iterations required by all the iterative methods with `TOL` $= 10^{-8}$ and `maxIter` $= 20000$ for both model problems, using $\mathbf{U}^{(0)} = \mathbf{0}$.

4 The Conjugate Gradient Method (CG)

Now we present a more efficient method called the *Conjugate Gradient* method, used for solving linear systems of equations with a symmetric and positive-definite coefficient matrix. CG is the starting point of Krylov subspace methods and essentially consists of two parts: choosing a descent direction (the direction of the residual), and picking a local minimum for f along that direction.

Before introducing the CG algorithm, we give a useful bound for the error.

Theorem 4.1 (CONVERGENCE BOUND FOR THE CG METHOD)**.** *Let $\boldsymbol{U}^{(k)}$ be the approximation at the k-th step of the CG algorithm. Then the error $\boldsymbol{e}^{(k)} := \boldsymbol{U} - \boldsymbol{U}^{(k)}$ satisfies the estimate*

$$\frac{||e_k||_A}{||e_0||_A} = \frac{||u - u_k||_A}{||u - u_0||_A} \leqslant 2\left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^k, \tag{4.1}$$

where $\kappa = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}$ is the condition number of matrix A.

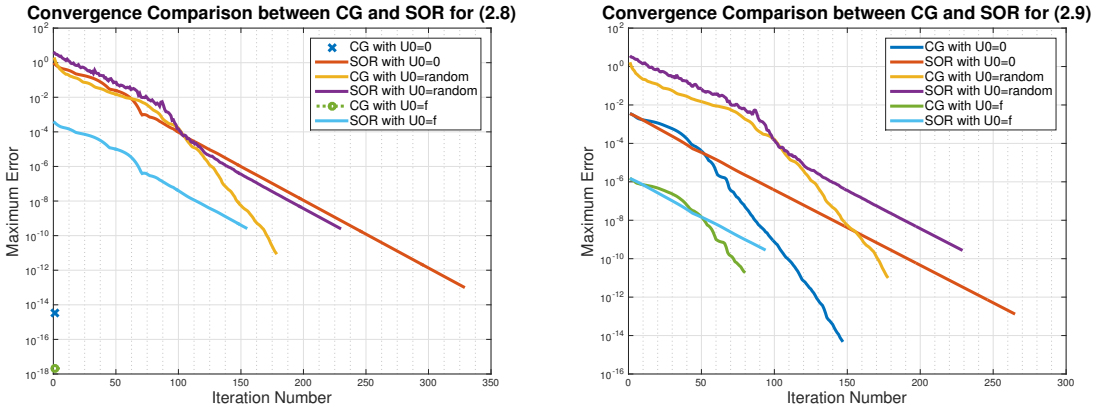
[1]Here we let RJ stand for Relaxed Jacobi.

Algorithm 6 Conjugate Gradient Method (CG)

1: **Initialise:** Choose $\mathbf{U}^{(0)}$, $\mathbf{p}_0 = \mathbf{r}_0 := \mathbf{f} - A\mathbf{U}^{(0)}$
2: **for** iterates $k = 0, 1, \ldots, n-1$
3: $\quad \alpha_k = \frac{r_k^\top p_k}{p_k^\top A p_k}$;
4: $\quad U_{k+1} = U_k + \alpha_k p_k$;
5: $\quad r_{k+1} = r_k - \alpha_k p_k$;
6: $\quad \beta_k = -\frac{p_k^\top A r_{k+1}}{p_k^\top A p_k}$;
7: $\quad p_{k+1} = r_{k+1} + \beta_k p_k$;
8: **end**

4.1 CG Method on Model Problems

In this subsection we present how the CG method behaves for the two model problems we are considering in this report. Figure 4 displays the ∞-norm of the error for the CG method as applied to the 5-point equations with $N = 48$, in the first 500 iterations. For comparison we also provide identical information for the SOR method. This is the outcome without preconditioning that we will see in § 6 and which accelerates convergence a great deal further. The numerical results agree with the theory since we expect the CG method to converge faster than any classical iterative method.



(a) Model Problem 1: (2.8). (b) Model Problem 2: (2.9).

Figure 4: The convergence of the CG method and SOR to the exact solution of the matrix problems (computed using `backslash` in MATLAB) with various $\mathbf{U}^{(0)}$.

In Table 2, we observe that the CG method converges in one step when the solution is an eigenvector of the coefficient matrix A. This happens in particular for (2.8) when we choose $\mathbf{U}^{(0)} = \mathbf{0}$ or $\mathbf{U}^{(0)} = \mathbf{f}$. This indicates that the Conjugate Gradients method is sensitive to initial guesses. See Appendix D for more details.

General Statement 4.2. If the solution $\mathbf{U}$ to $A\mathbf{U} = \mathbf{f}$ can be written as a linear combination of s eigenvectors of A, then if the initial guess $\mathbf{U}^{(0)}$ is a linear combination of the same s eigenvectors as the solution is, then the Conjugate Gradients method converges in at most s steps.

	Number of iterations for convergence			
Initial guess $\mathbf{U}^{(0)}$	CG (2.8)	SOR (ω_{opt}) (2.8)	CG (2.9)	SOR (ω_{opt}) (2.9)
Zero	1	328	147	265
RHS: $\mathbf{f}$	1	154	80	94
Random	178	229	178	229

Table 2: Number of iterations required by CG and SOR with $\texttt{TOL} = 10^{-10}$.

A similar statement holds for the Jacobi and Relaxed Jacobi methods. Note that if we use Relaxed Jacobi, then we have $G_{\text{RJ}} = (1-\omega)I + \omega G_{\text{J}}$, so the eigenvalues are given by (3.4). In particular, $\lambda_{\text{RJ}}^{r,s}\left(\frac{1}{2}\right) = \frac{1}{2} + \frac{1}{4}\left[\cos(r\pi h) + \cos(s\pi h)\right] > 0$, with $1 \leqslant r, s \leqslant N-1$. The smallest $\lambda_{\text{RJ}}^{r,s}\left(\frac{1}{2}\right)$ correspond to large r and s, which in turn correspond to high frequency eigenvectors. Thus, we get faster convergence of high frequency components of the error. This means that the error is smoothed and we can solve on a coarser grid. This motivates the next section.

5 The Multigrid Method

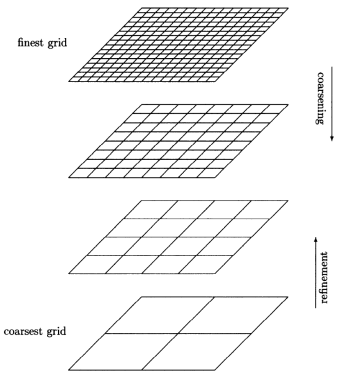


Figure 5: Nested grids, from the finest to the coarsest. Figure taken from [Ise09].

After discussing simple iterative methods and Krylov subspace methods such as the CG method, we are now ready to introduce multigrid methods. In this section, we will discuss the main ideas behind the multigrid algorithm, starting from the two-grid algorithm, and then outlining the multigrid one. A picture of the multigrid work-flow is given in Figure 5.

The multigrid method converges much faster than the classical iterative methods. The key idea is to switch to a coarser grid to estimate the error. The advantages are that iterating on a coarser grid takes less work, and the convergence rate for some error components is greatly improved by transferring the error to a coarser grid. We can rapidly damp all of the error modes using the smoother computational benefits associated with frequent visits to coarse grids.

5.1 Two-Grid Algorithm

We start by introducing the simplest version of multigrid algorithms, namely the two-grid algorithm. Note that it is important to choose an appropriate *smoother*, as well as *restriction* and *prolongation* operators. Before outlining the algorithm, let us first introduce some notation. We define two meshes: Ω^h (fine grid) and Ω^{2h} (coarse grid), the restriction operator, $R_h^{2h} : \Omega^h \to \Omega^{2h}$, and the prolongation operator, $P_{2h}^h : \Omega^{2h} \to \Omega^h$. In Figure 6, blue points correspond to coarse grid points and all points correspond to fine grid points.

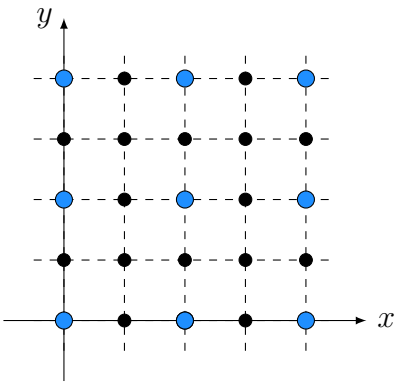


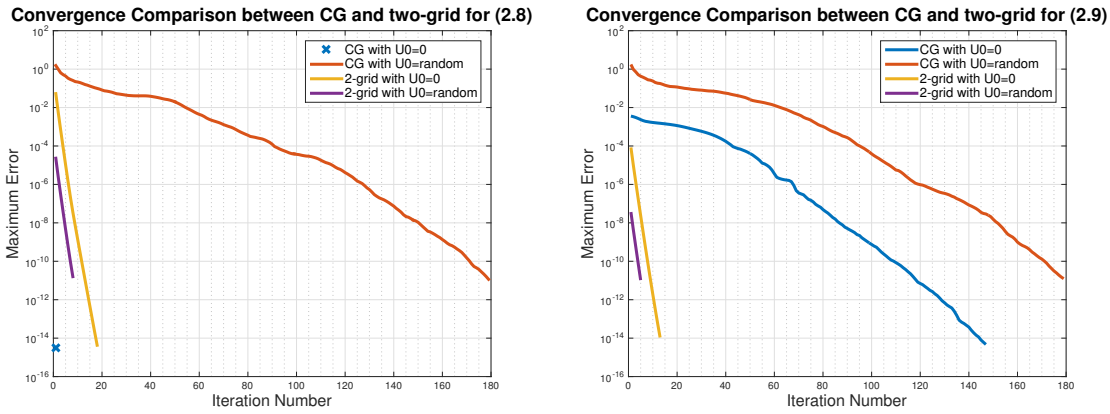
Figure 6: Schematic of the grid used for the two-grid algorithm. Also extends to multigrid.

The simplest way to define the prolongation operator is by linear interpolation; see [Saa03, Chap. 13]. Note that for the restriction operator we normally take $R_h^{2h} = \alpha(P_{2h}^h)^\top$ where $\alpha \in \mathbb{R}$ is such that $\alpha(P_{2h}^h)^\top(P_{2h}^h)\mathrm{e} = R_h^{2h}P_{2h}^h\mathrm{e} = \mathrm{e}$, with $\mathrm{e} = (1, 1, \ldots, 1)^\top$. Thus, we have $\mathbf{U}^h = P_{2h}^h\mathbf{U}^{2h}$ and $\mathbf{U}^{2h} = R_h^{2h}\mathbf{U}^h$. The loss of accuracy is small if $\mathbf{U}^h$ is a 'smooth vector', i.e. a vector of coefficients representing a non-oscillatory function. There are two possibilities for the coarse grid operator A^{2h}: the 5-point formula on the $2h$-mesh or the *Galerkin coarse grid operator* $A^{2h} = R_h^{2h}A^hP_{2h}^h = \alpha(P_{2h}^h)^\top A^hP_{2h}^h$. The latter is more commonly used in finite element methods.

Algorithm 7 Two-Grid Algorithm

1: **Initialise:** Choose $\mathbf{U}^{(0)}$
2: **for** two-grid iterations $i = 0$ until **convergence**
3: (Pre-smooth) ν_1-times on Ω^h
4: Calculate the residual $\mathbf{r}^h = \mathbf{f}^h - A^h\mathbf{U}^{(k)}$ and restrict it on Ω^{2h}: $\mathbf{r}^{2h} = R_h^{2h}\mathbf{r}^h$
5: Solve $A^{2h}\mathbf{e}^{2h} = \mathbf{r}^{2h}$ to get coarse grid correction $\mathbf{e}^{2h}$
6: Prolong the coarse error $\mathbf{e}^{2h}$: $\mathbf{U}^h = \mathbf{U}^h + P_{2h}^h\mathbf{e}^{2h}$
7: (Post-smooth) ν_2-times on Ω^h
8: Update the solution: $\mathbf{U}^{(k+1)} \leftarrow \mathbf{U}^{(k)}$
9: **end**

In Figure 7, we see that the two-grid method converges much faster than the CG method for any $\mathbf{U}^{(0)}$. Here we used 4 pre-smoothing and 4 post-smoothing Relaxed Jacobi $\left(\omega = \frac{2}{3}\right)$ iterations. See Table 6 in Appendix E for more details on the results.



(a) Model Problem 1: (2.8). (b) Model Problem 2: (2.9).

Figure 7: The convergence of the CG and two-grid method to the exact solution of the matrix problems (computed using `backslash` in MATLAB) with two $\mathbf{U}^{(0)}$'s.

5.2 Multigrid Algorithm — Motivation

The main components of the multigrid method are: a hierarchy of levels along with restriction and prolongation operators to move between grids, as well as a smoother.

Algorithm 8 Multigrid Algorithm: $\mathbf{U}_h^{(k)} = \mathrm{MG}^{\gamma}(A^h, \mathbf{U}^{(0)}, \mathbf{f}^h, \nu_1, \nu_2, \gamma)$

Initialise: Choose $\mathbf{U}^{(0)}$
(Pre-smooth) ν_1 times on Ω^h
if $(h == h_0)$
    Solve the problem
else
    Restrict residual on Ω^{2h}: $\mathbf{r}^{2h} = R_h^{2h}\mathbf{r}^h$, where $\mathbf{r}^h = \mathbf{f}^h - A^h\mathbf{U}^{(k)}$
    Set initial iterate on the next coarser grid: $\mathbf{e}^{2h} = 0$
    if $(2h == h_0)$
        Solve $A^{2h}\mathbf{e}^{2h} = \mathbf{r}^{2h}$ to get coarse grid correction $\mathbf{e}^{2h}$
    else
        Recursion: $\mathbf{e}^{2h} = \mathrm{MG}^{\gamma}(A^{2h}, 0, \mathbf{r}^{2h}, \nu_1, \nu_2, \gamma)$
    end
Prolong the coarse error $\mathbf{e}^{2h}$: $\mathbf{U}^h = \mathbf{U}^h + P_{2h}^h\mathbf{e}^{2h}$
(Post-smooth) ν_2 times on Ω^h
Update the solution: $\mathbf{U}^{(k+1)} \leftarrow \mathbf{U}^{(k)}$
end

A smoother is any scheme that has the smoothing property of damping quickly the high-frequency components of the error. We seek the solution to $A\mathbf{U} = \mathbf{f}$ defined on Ω^h. In the algorithm, we embed the two-level method into itself [Vol14]. We assume there exist $l+1$ grids, $l \geqslant 0$, where Ω^h is the finest grid and the grid spacing for each coarser grid doubles. Let $L = 2^l$ and h_0 stand for the coarsest mesh-size. Note that the implementation of the multigrid cycle is of recursive nature.

In Algorithm 8, there is a new parameter γ, which determines how many times MG is iterated. The case $\gamma = 1$ corresponds to the multigrid V-cycle and $\gamma = 2$ corresponds to the W-cycle. For an illustration of these, see the diagrams in Figure 14 and Figure 15 of Appendix E. Some tools of analysis for the multigrid would be the smoothing and approximation properties. The reader is referred to [ESW14, § 4.3.2] for more details.

6 Preconditioning Techniques

The convergence rate of iterative methods depends on the spectral properties of A. Thus, one may attempt to transform the linear system into one that has the same solution but better spectral properties. This captures the essence of preconditioning. Preconditioners replace the original problem $Au = f$ by a new system with a smaller condition number or a better clustered spectrum. Good preconditioners improve the convergence of the numerical method, sufficiently to overcome the extra cost of constructing and applying the preconditioner.

If P is a symmetric, positive-definite matrix (P is referred to as a *preconditioning matrix* or *left preconditioner*), then the preconditioned CG method (PCG) consists of applying the CG method to the preconditioned system

$$P^{-1}Au = P^{-1}f. \tag{6.1}$$

Similarly, if $AP^{-1}v = f$, where $v = Pu$, then P is called a *right preconditioner* and finally, if P is available in split form $P = P_L P_R$, such that $P_L^{-1}AP_R^{-1} = P_L^{-1}f$, where $v = P_R u$, then we have *centred preconditioners*. If we let $P = HH^\top$ then (6.1) can be written as

$$(H^{-1}AH^{-\top})(H^\top u) = H^{-1}f. \tag{6.2}$$

Note that even if the matrix product $P^{-1}A$ is non-symmetric it is still possible to use P as a preconditioner whilst preserving symmetry of the preconditioned system. Because of the similarity transformation

$$H^{-\top}(H^{-1}AH^{-\top})H^\top = P^{-1}A, \tag{6.3}$$

this CG method with preconditioner P is called the Preconditioned Conjugate Gradient method. This involves solving for $z^{(k)}$ given $r^{(k)}$ of $Pz^{(k)} = r^{(k)}$ at each iteration.

6.1 Choices for the Preconditioner P

In devising a preconditioner we have to find a matrix P that approximates A, and for which solving a system is easier than solving one with A, or we could even approximate the inverse of A. A complete and detailed discussion of preconditioners can be found in [Axe96], or in [VdV03, Chap. 13]. Requirements of the PCG method include:

1. $\kappa(P^{-1}A) \ll \kappa(A)$ for fast convergence,
2. the solution of $Pz^{(k)} = r^{(k)}$ should be cheap in terms of computational memory and operation counts.

One could take $P = I$ which corresponds to unpreconditioned CG and leads to simple ideas like $P = \text{diag}(A)$, $P = \text{triag}(A)$, or P consisting of part of A with bandwidth f. $P = I$ satisfies Requirement 2. Similarly, one could choose $P = A$, leading to consideration of incomplete triangular factorisations of A. Notice that this case satisfies Requirement 1. In this section, we present some standard choices for preconditioners largely following [Wat15].

Diagonal Preconditioners

One commonly used preconditioner which can be implemented easily is the *Jacobi preconditioner* in which P is $P = \text{diag}(A)$. This is in general effective when A is a symmetric, positive-definite matrix. If it is non-symmetric then a standard choice is $p_{ii} = \left(\sum_{j=1}^{n} a_{ij}^2 \right)^{1/2}$. More details can be found in [QSS10] and in [BBC+94, § 3]. Block diagonal preconditioners can be constructed in a similar way.

Polynomial Preconditioners

One could also attempt to estimate the spectrum of A, find a polynomial p such that $1 - zp(z)$ is small on the approximate spectrum and then define the preconditioner as $P^{-1} = p(A)$. This is known as *polynomial preconditioning*. The preconditioned system takes the form $p(A)Au = p(A)f$ and we expect the spectrum of $p(A)A$ to be more clustered near $z = 1$ than that of A. Suppose that A is of the form $A = I - B$, where $\rho(B) < 1$. Using the *Neumann series* we can write the inverse of A as $A^{-1} = \sum_{j=0}^{\infty} B^j$.

So an approximation may be derived by truncating this infinite series [BBC+94]. Another example is to express the residual polynomial $q(z) = 1 - zp(z)$ in terms of Chebyshev polynomials [Riv74].

Incomplete Cholesky Factorisation (IC)

Another approach is to apply a sparse Cholesky factorisation to A. If A is a symmetric, positive-definite matrix it is sometimes possible to carry out the incomplete factorisation $LL^\top = A + R$, where $L \in \mathbb{R}^{n\times n}$ has a given sparsity pattern. In this method we choose to discard small elements that fall outside this pattern. So, if R is a small remainder, then the preconditioner is $P = (LL^{-\top})^{-1}$ and its action on a vector is done by two sparse triangular solves. If we replace $Au = f$ by

$$L^{-1}AL^{-\top}(Lu) = L^{-1}f \tag{6.4}$$

and note that $L^{-1}AL^{-\top} = I - L^{-1}RL^{-\top}$, then since R is small, we can expect the eigenvalues of $L^{-1}AL^{-\top}$ to be clustered around 1.

We wish to confirm whether an Incomplete Cholesky factorisation preconditioner leads to faster convergence of the CG. If our domain is $\Omega = (0,1) \times (0,1)$ then $\kappa(A) = \mathcal{O}(h^{-2})$. See [ESW14, § 1.6] for a detailed explanation of this. Therefore, (4.1) implies that the number of CG iterations needed for convergence of the CG is $\mathcal{O}(h^{-1})$. If we choose the sparsity pattern of the factor L to be the same as the lower triangular part of A then $\kappa(P^{-1}A) = \mathcal{O}(h^{-2})$. One modification to IC is the modified Incomplete Cholesky (MIC) [BBC+94, § 3.4]: If $a_{ik}a_{kk}^{-1}a_{kj} \neq 0$, instead of simply discarding this quantity, subtract it from the diagonal element a_{ii}. This reduces to $\kappa(P^{-1}A) = \mathcal{O}(h^{-1})$ and bound (4.1) implies that the number of PCG iterations that are needed for convergence are $\mathcal{O}(h^{-1/2})$.

Sparse Approximate Inverse Preconditioners (SpAI)

In SpAI preconditioners we are constructing a sparse matrix M to approximate A^{-1}, see [GS95, CDG92]. Specified sparsity patterns on the approximate inverse are imposed. More specifically, let $\mathcal{N} := \{1, \ldots, N\}$ and $\mathcal{S}$ be a given set of index pairs (i, j), with $i, j \in \mathcal{N}$, then $\mathcal{G}_\mathcal{S}$ is the space of all $N \times N$ matrices with entries in positions indexed by $\mathcal{S}$. Similarly, if we define $\mathcal{S}_j := \{i : (i, j) \in \mathcal{S}\}$ then $\mathcal{G}_\mathcal{S}^j$ is the space of all N-vectors with entries in positions indexed by $\mathcal{S}_j$. In particular, for a non-singular matrix A the approximate inverse is defined as $P = M^{-1}$ that solves

$$M = \arg\min_{M\in\mathcal{G}_\mathcal{S}} ||AM - I||_F^2. \tag{6.5}$$

For an explanation on why we choose to minimise the Frobenius norm, see [HG94]. We find the non-zero entries of M by solving the unconstrained least-squares problem

$$\min_{M \in \mathcal{G}_{\mathcal{S}}} ||AM - I||_F^2 = \min_{m_j \in \mathcal{G}_{\mathcal{S}}^j} \sum_{j=1}^{N} ||Am_j - e_j||_2^2 = \sum_{j=1}^{N} \min_{m_j \in \mathcal{G}_{\mathcal{S}}^j} ||Am_j - e_j||_2^2, \tag{6.6}$$

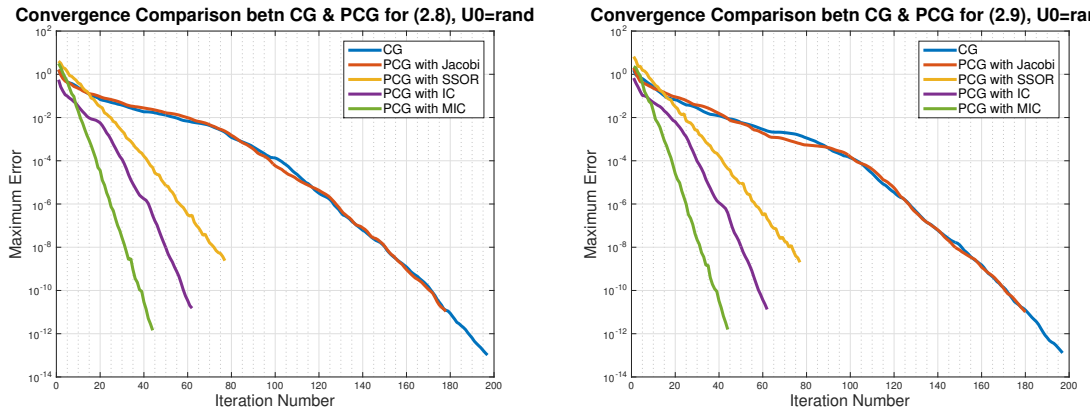
where m_j represents the j-th column of M, e_j is the j-th column of the identity matrix I, $||\cdot||_F$ is the Frobenius norm and $||\cdot||_2$ is the 2-norm. Notice that this choice leads to parallelism since the columns m_j of M can be computed independently. Indeed, the solution of (6.6) separates into N independent least-squares problems:

$$\min_{m_j \in \mathcal{G}_{\mathcal{S}}^j} ||Am_j - e_j||_2^2 \quad \text{with } j = 1, \ldots, N. \tag{6.7}$$

The main advantage is that the least-squares problems are cheap to solve given that $\mathcal{S}$ is sparse and the residuals can be improved by enlarging the space $\mathcal{S}$.

6.2 The Preconditioned Conjugate Gradient Method (PCG)

We compare the preconditioned CG method with the regular CG method. Figure 8 shows how various preconditioners improve the convergence rate of the CG method.



(a) Model Problem 1: (2.8). (b) Model Problem 2: (2.9).

Figure 8: The convergence of unpreconditioned CG and preconditioned CG using Jacobi, SSOR, IC and the modified IC, with $\mathbf{U}^{(0)} = \texttt{randn(N-1)}$.

	Number of iterations for convergence				
Model Problem	CG	PCG (Jacobi)	PCG (SSOR)	PCG (IC)	PCG (MIC)
(2.8) & (2.9)	197	178 & 180	77	62	44

Table 3: Number of iterations required by CG and the four PCG methods with $\texttt{TOL} = 10^{-12}$ and $\texttt{maxIter} = 5000$ for both model problems, using $\mathbf{U}^{(0)} = \texttt{randn(N-1)}$.

We observe that the Jacobi preconditioner $P = \text{diag}(A)$ does not improve CG since the diagonal elements of A are all identical. It is essentially the identity matrix multiplied by a constant which corresponds to the regular CG method. Thus, the Jacobi preconditioner would only work if the diagonal elements were not all the same. Moreover, the CG method with an SSOR preconditioner performs sufficiently better than unpreconditioned CG. However, it is clear that Incomplete Cholesky preconditioners are still more effective, with the modified version being the most effective out of the preconditioners tested in this report. This result again agrees with the theory.

7 Conclusion

The approach in this report was to focus on a number of numerical methods and present some of their most important properties. We considered two model problems to validate the implementations we carried out and give a sense of how the various algorithms perform for each of these model problems.

We investigated the classical iterative methods for solving general, large and sparse linear systems, together with their convergence theory. We concluded that SOR is the most efficient out of the five classical iterative methods tested; however, it is still slower than the CG or two-grid method. Moreover, we concluded that the CG method with IC preconditioning was more efficient than the regular CG method. Various other preconditioners were suggested to improve the convergence of the algorithm.

Finally we noticed that there are no major differences between problems (2.8) and (2.9) when applying the numerical methods. An exception is the CG method, when $\mathbf{U}^{(0)}$ is an eigenvector of A, since the solution is also an eigenvector of A for (2.8).

References

[Axe96] O. Axelsson. *Iterative solution methods*. Cambridge university press, 1996.

[BBC+94] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the solution of linear systems: building blocks for iterative methods*. SIAM, 1994.

[CDG92] J. D. F. Cosgrove, J. C. Diaz, and A. Griewank. Approximate inverse preconditionings for sparse linear systems. *International journal of computer mathematics*, 44(1-4):91–110, 1992.

[Dem97] J. W. Demmel. *Applied numerical linear algebra*. SIAM, 1997.

[ESW14] H. C. Elman, D. J. Silvester, and A. J. Wathen. *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics*. Oxford University Press (UK), 2014.

[GGK14] W. Gander, M. J. Gander, and F. Kwok. *Scientific computing-An introduction using Maple and MATLAB*, volume 11. Springer Science & Business, 2014.

[GS95] N. I. M. Gould and J. A. Scott. *On approximate-inverse preconditioners*. Central Laboratory for the Research Councils, Library and Information Services, Rutherford Appleton Laboratory, 1995.

[HG94] T. Huckle and M. Grote. A new approach to parallel preconditioning with sparse approximate inverses. *Manuscript SCCM*, pages 94–03, 1994.

[Ise09] A. Iserles. *A first course in the numerical analysis of differential equations*. Number 44. Cambridge university press, 2009.

[Kah58] W. M. Kahan. *Gauss-Seidel methods of solving large systems of linear equations*. PhD thesis, Thesis–University of Toronto, 1958.

[LeV07] R. J. LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007.

[QSS10] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical mathematics*, volume 37. Springer Science & Business Media, 2010.

[Riv74] T. J. Rivlin. Chebyshev polynomials. *John Wiley & Sons, Inc., copublished in the United States with John Wiley*, 1974.

[Saa03] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[VdV03] H. A. Van der Vorst. *Iterative Krylov methods for large linear systems*, volume 13. Cambridge University Press, 2003.

[Vol14] J. Volker. Lecture notes in multigrid methods. https://www.wias-berlin.de/people/john/LEHRE/MULTIGRID/multigrid.pdf, Winter Semester 2013/14. As seen on April 30, 2017.

[Wat15] A. J. Wathen. Preconditioning. *Acta Numerica*, 24:329–376, 2015.

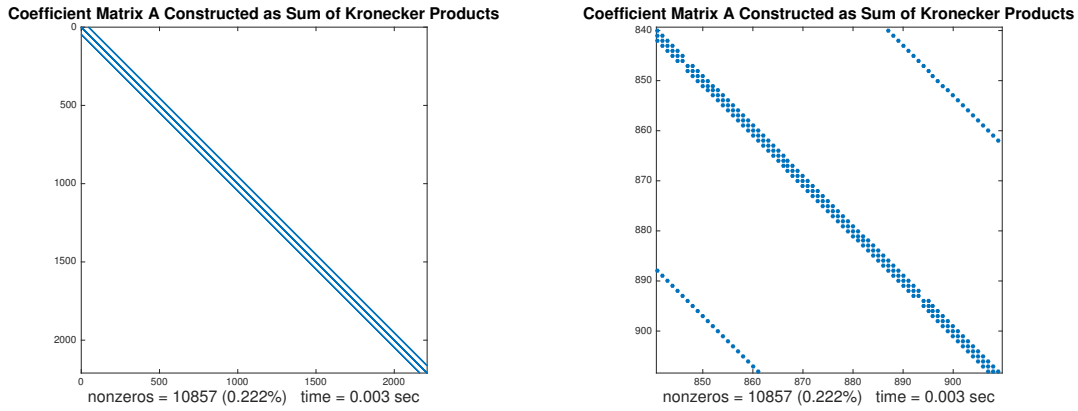
Appendix A Sparse Storage in MATLAB

In this report, we were working in MATLAB with a sparse matrix arising from the finite difference method. Note that the sparse matrix commands store only the nonzero elements and so if a matrix contains many zeros, converting it to sparse storage saves memory. In particular, we construct the matrix A in the following way:

Listing 1: Constructing the coefficient matrix A using sparse storage in MATLAB.

```
function [A] = Laplacian(N,M)
n=N-1;
m=M-1;
Ix=speye(n,n);
Iy=speye(m,m);
Ex=sparse(2:n,1:n-1,1,n,n);
Ey=sparse(2:m,1:m-1,1,m,m);
Dxx=Ex+Ex'-2*Ix;
Dyy=Ey+Ey'-2*Iy;
A=kron(Dxx,Iy)+kron(Ix,Dyy);
end
```

Graphic visualisation of the structure of the sparse matrix A is sometimes a useful tool and so we investigate the sparsity pattern of A using the `spy(A)` command. In Figure 9, we see the structure of A, the number of nonzero elements of the matrix and the time it takes for it to be constructed for the case $N=M=48$ (as used in the implementations of our numerical methods).



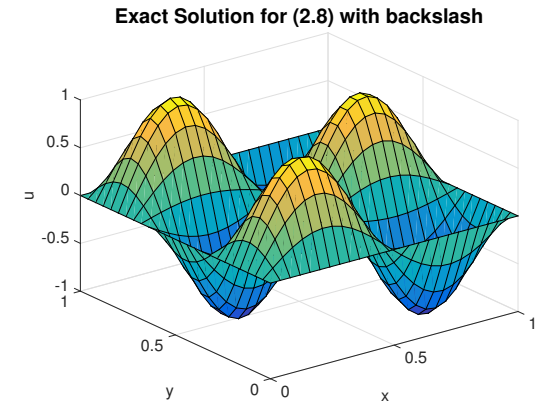
(a) Sparsity pattern of matrix A. (b) Zoomed-in part of sparsity pattern of$A$.

Figure 9: Plots of the sparsity pattern of the coefficient matrix A with $N=M=48$ using MATLAB's `spy` command.

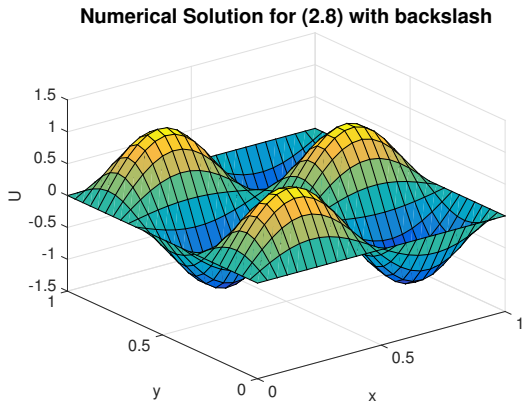
The `backslash` command in MATLAB can be used to solve systems using sparse storage by implementing some very efficient direct methods which depend on the structure of matrix A, i.e. whether it is tridiagonal, banded, or symmetric and positive-definite. In Appendix B we present some results obtained using `backslash`.

Appendix B More Graphs using `backslash`

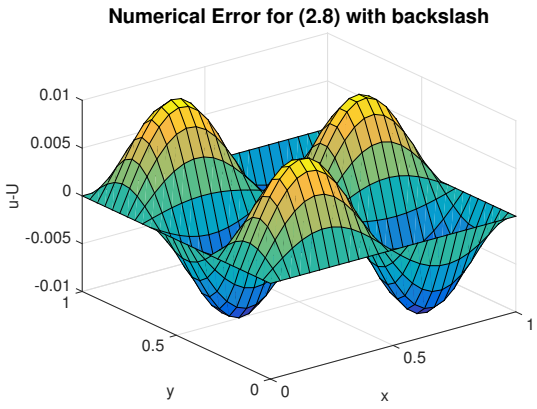
Figures 10a, 10b show the mesh plot of the analytical solution and the numerical solution vs. (x, y) for $N{=}M{=}25$, respectively and Figures 10c, 10d show the error at each mesh point computed by subtracting the exact solution from the numerical solution obtained through `backslash` for $N{=}M{=}25$ and $N{=}M{=}50$, respectively. Experimenting with the number of mesh points we noticed that the greater the number of mesh points, the smaller the error. Figure 10b shows that the numerical solution is smooth and zero at the boundaries as expected.



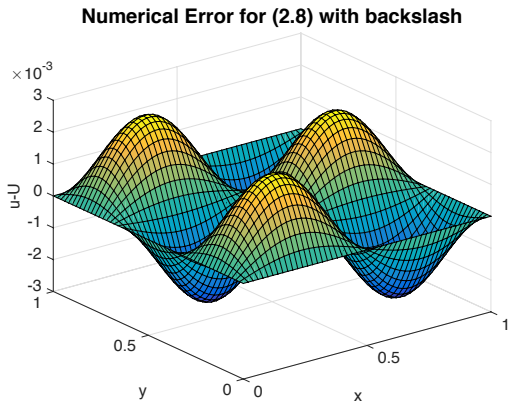
(a) Exact solution with $N{=}M{=}25$.



(b) Numerical solution with $N{=}M{=}25$.



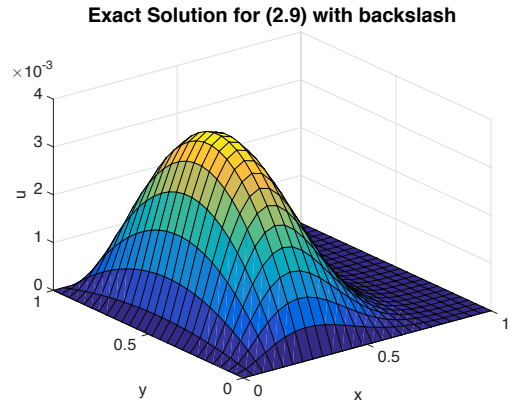
(c) Numerical error with $N{=}M{=}25$.



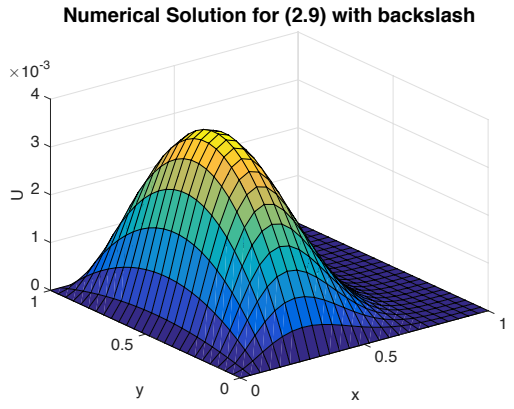
(d) Numerical error with $N{=}M{=}50$.

Figure 10: Mesh plots produced by MATLAB using the `backslash` command for (2.8).

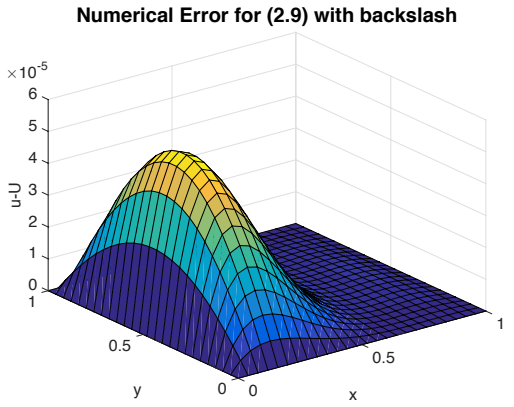
We proceed in a similar way for (2.9) in Figure 11. We can see that the numerical solution is smooth and that as we increase the number of mesh points the numerical error, which is the difference between the numerical solution and the analytical solution of the Poisson problem (2.9), decreases. Notice in particular how the scales in the error axis compare in Figure 11c and Figure 11d.



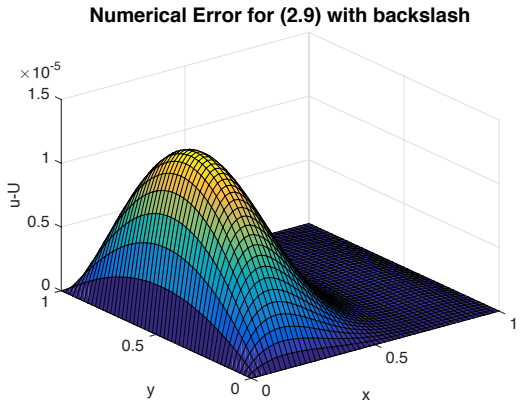
(a) Exact solution with $N{=}M{=}25$.



(b) Numerical solution with $N{=}M{=}25$.



(c) Numerical error with $N{=}M{=}25$.



(d) Numerical error with $N{=}M{=}50$.

Figure 11: Mesh plots produced by MATLAB using the `backslash` command for (2.9).

Appendix C More on the Comparison of Classical Iterative Methods

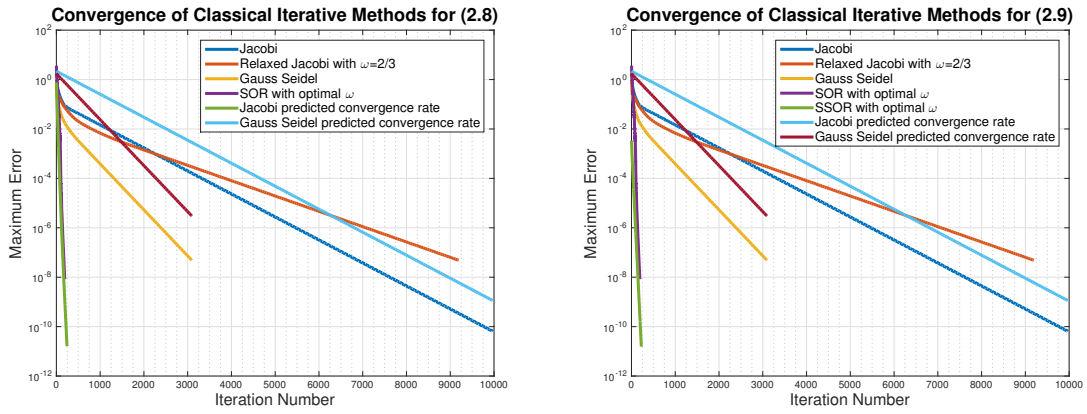
First we give a summary of the M and N splittings for the five classical iterative methods in Table 4. In the table, ω corresponds to a relaxation parameter, L is the lower triangular part of A, D is the diagonal part and U is the upper triangular part.

	Matrix Splittings of $A = M - N$	
Method	M	N
Jacobi	D	$-(L+U)$
Relaxed Jacobi	D	$(1-\omega)I - \omega D^{-1}(L+U)$
Gauss-Seidel	$D+L$	$-U$
SOR	$D+\omega L$	$(1-\omega)D - \omega U$
SSOR	$\frac{1}{\omega(2-\omega)}(D+\omega L)D^{-1}(D+\omega U)$	$\left[I - \frac{1}{2-\omega}(D+\omega L)D^{-1}\right]\left[U + \frac{(1-\omega)}{\omega}D\right]$

Table 4: Summary of M and N splittings for A for the five classical iterative methods.

In this appendix, we compare again the rate of convergence of the five classical iterative methods as introduced in § 3 but this time using a different initial guess to the one chosen in §§ 3.6.

Comparing Figure 3 with Figure 12, we notice that the iterative methods converge much faster when we take as an initial guess $\mathbf{U}^{(0)} = \mathbf{0}$ rather than $\mathbf{U}^{(0)} =$ `randn(N-1)`. This is because the zero vector is closer to the true solution, and so it takes less iterations until it converges. Iterative methods are also sensitive to initial guesses.



(a) Model Problem 1: (2.8). (b) Model Problem 2: (2.9).

Figure 12: The convergence of the iterative methods to the exact solution of the matrix problems (computed using `backslash` in MATLAB), using $\mathbf{U}^{(0)} =$ `randn(N-1)`.

In Table 5 we notice that there is no big difference in the number of iterations required for the five iterative methods to converge between the two model problems, or no difference at all.

	Number of iterations for convergence				
Model Problem	Jacobi	RJ (ω_{opt})	Gauss-Seidel	SOR (ω_{opt})	SSOR(ω_{opt})
(2.8)	9970	9191	3096	191	243
(2.9)	9970	9191	3096	191	225

Table 5: Number of iterations required by all iterative methods with `TOL` $= 10^{-8}$ and `maxIter` $= 20000$ for both model problems, using $\mathbf{U}^{(0)} =$ `randn(N-1)`.

Appendix D Convergence of CG in One Step

In this appendix, we include more details on why the CG method converges in just one iteration when $\mathbf{U}^{(0)} = \mathbf{0}$ or when $\mathbf{U}^{(0)} = \mathbf{f}$.

First note that the solution for (2.8) is $\mathbf{U} = \mathbf{V}^{(2,3)}$, where this is the eigenvector of A with index $(2, 3)$. This means that the solution is an eigenvector of A. Since $\mathbf{U} = \mathbf{V}^{(2,3)}$ then we have that $A\mathbf{U} = A\mathbf{V}^{(2,3)} = \lambda^{(2,3)}\mathbf{V}^{(2,3)} = \mathbf{f}$, with $\mathbf{f}$ also an eigenvector of A.

Let us now choose as an initial guess the vector $\mathbf{U}^{(0)} = c\mathbf{V}^{(2,3)}$. Note that here the scalar c can either be $c = 0$ which implies that $\mathbf{U}^{(0)} = 0$, or $c = \lambda^{(2,3)}$ which implies that $\mathbf{U}^{(0)} = \mathbf{f}$. Then we write

$$\mathbf{p}_0 = \mathbf{r}_0 = \mathbf{f} - A\mathbf{U}^{(0)} = \lambda^{(2,3)}(1 - c)\mathbf{V}^{(2,3)}, \tag{D.1}$$

where $||\mathbf{p}_0|| >$ `TOL` presumably. Calculating the rest of the components of the CG method we obtain

$$\alpha_0 = \frac{||r_0||_2^2}{p_0^\top A p_0} = \frac{1}{\lambda}, \tag{D.2}$$

$$\mathbf{U}^{(1)} = \mathbf{U}^{(0)} + \alpha_0\mathbf{p}_0 = c\mathbf{V} + \frac{1}{\lambda}\lambda(1 - c)\mathbf{V} \tag{D.3}$$

and finally

$$\mathbf{r}_1 = \mathbf{r}_0 - \alpha_0 A\mathbf{p}_0 = \lambda(1 - c)\mathbf{V} - \frac{1}{\lambda}A\lambda(1 - c)\mathbf{V} = \mathbf{0}. \tag{D.4}$$

This shows that the CG method for (2.8) converges in one step if $\mathbf{U}^{(0)} = \mathbf{0}$ or $\mathbf{U}^{(0)} = \mathbf{f}$.

Appendix E More on the Multigrid Method

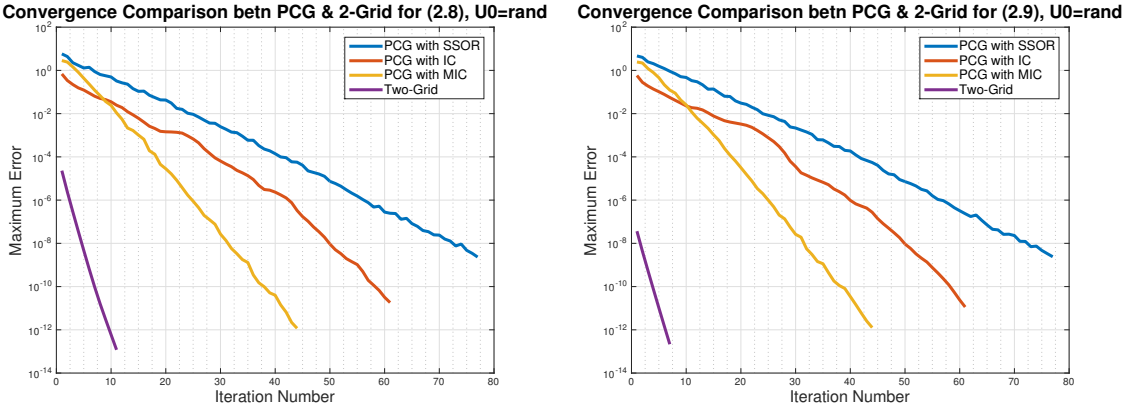
In Table 6 we see that the two-grid method converges in just a few iterations for both (2.8) and (2.9). It is in general much faster than the unpreconditioned CG method.

The only occasion where this does not hold is when the choice of $\mathbf{U}^{(0)}$ is such that it favors the CG method, i.e. if we choose $\mathbf{U}^{(0)}$ to be an eigenvector of A just like the solution of the problem.

	Number of iterations for convergence			
Initial guess $\mathbf{U}^{(0)}$	CG (2.8)	Two-Grid (2.8)	CG (2.9)	Two-Grid (2.9)
Zero	1	18	147	13
Random	179	8	179	5

Table 6: Iteration number required by CG and Two-Grid method with $\texttt{TOL} = 10^{-10}$.

However, we can also compare the two-grid method against the preconditioned CG method to test which of these converges faster. The results for $N = 48$ with $\texttt{TOL} = 10^{-12}$ and $\texttt{maxItN} = 5000$ are presented in Figure 13.



(a) Model Problem 1: (2.8). (b) Model Problem 2: (2.9).

Figure 13: Comparison of convergence of the various PCG methods and the Two-Grid method to the exact solution of the matrix problems (computed using `backslash` in MATLAB), using $\mathbf{U}^{(0)} = \texttt{randn(N-1)}$.

	Number of iterations for convergence			
Model Problem	Two-Grid	PCG (SSOR)	PCG (IC)	PCG (MIC)
(2.8) & (2.9)	11 & 7	77	61	44

Table 7: Number of iterations required by Two-Grid and the four PCG methods with $\texttt{TOL} = 10^{-12}$ and $\texttt{maxIter} = 5000$ for both model problems, using $\mathbf{U}^{(0)} = \texttt{randn(N-1)}$.

We mentioned in §§ 5.2 that if we iterate the MG once ($\gamma = 1$) then this corresponds to a V-cycle multigrid. The name becomes apparent when one looks at

Figure 14 and the way we move through the different levels of grids. The V-cycle consists of going down through the grids from fine to coarse, performing a relaxation sweep on each grid, then coming back up from coarse to fine, and again performing a relaxation sweep.

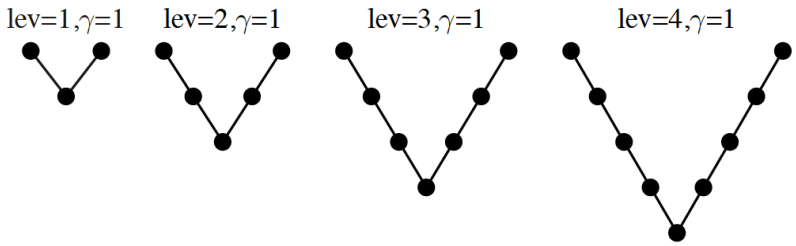


Figure 14: V-cycles in multigrid methods. Figure taken from [Saa03].

But other patterns of visiting grids are also possible. For instance, if one iterates the MG twice ($\gamma = 2$) then this corresponds to a multigrid W-cycle. Here, one uses two V-cycles at each of the coarser levels, resulting in a pattern like the one shown in Figure 15 for four grid levels.

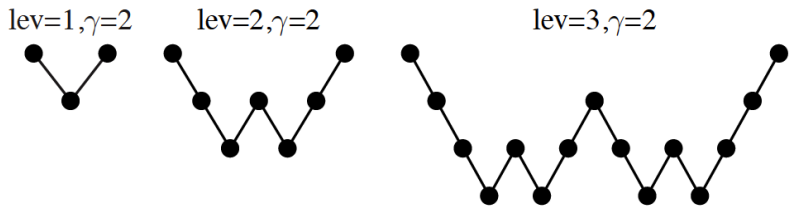


Figure 15: W-cycles in multigrid methods. Figure taken from [Saa03].