A Comparison of Solving the Poisson Equation Using Several Numerical Methods in Matlab and Octave on the Cluster maya

Sarah Swatski, Samuel Khuvis, and Matthias K. Gobbert (gobbert@umbc.edu)

Department of Mathematics and Statistics, University of Maryland, Baltimore County

Technical Report HPCF–2014–10, www.umbc.edu/hpcf > Publications

Abstract

Systems of linear equations resulting from partial differential equations arise frequently in many phenomena such as heat, sound, and fluid flow. We apply the finite difference method to the Poisson equation with homogeneous Dirichlet boundary conditions. This yields in a system of linear equations with a large sparse system matrix that is a classical test problem for comparing direct and iterative linear solvers. We compare the performance of Gaussian elimination, three classical iterative methods, and the conjugate gradient method in both Matlab and Octave. Although Gaussian elimination is fastest and can solve large problems, it eventually runs out of memory. If very large problems need to be solved, the conjugate gradient method is available, but preconditioning is vital to keep run times reasonable. Both Matlab and Octave perform well with intermediate mesh resolutions; however, Matlab is eventually able to solve larger problems than Octave and runs moderately faster.

Key words. Finite Difference Method, Iterative Methods, Matlab, Octave, Poisson Equation.

AMS subject classifications (2010). 65F05, 65F10, 65M06, 65Y04, 35J05.

1 Introduction

Partial differential equations (PDEs) are used in numerous disciplines to model phenomena such as heat, sound, and fluid flow. While many PDEs can be solved analytically, there are many that cannot be solved analytically or for which an analytic solution is far too costly or time-consuming. However, in many applications, a numerical approximation to the solution is sufficient. Therefore, various numerical methods exist to approximate the solutions to PDEs.

An ideal way to test a numerical method for PDEs is to use it on a PDE with a known analytical solution. That way, the true solution can be used to compute the error of the numerical method. By testing multiple numerical methods on the same PDE and comparing their performance, we can determine which numerical methods are most efficient.

The finite difference method uses finite differences to approximate the derivatives of a given function. After it is applied, we will have a system of linear equations for the unknowns. We will apply this method to the Poisson equation, which is an elliptic PDE that is linear and has constant coefficients. It can be solved analytically using techniques such as separation of variables and Fourier expansions. We will use the system of linear equations resulting from the finite difference method applied to the Poisson equation to compare the linear solvers Gaussian elimination, classical iterative methods, and the conjugate gradient method. This problem is a popular test problem and studied in [2, 3, 4, 7].

Matlab is the most commonly used commercial package for numerical computation in mathematics and related fields. However, Octave is a free software package that uses many of the same features and commands as Matlab. Mathematics students typically take at least one course that utilizes a numerical computation software, and Matlab is essentially the software of choice by professors and textbook authors alike. Since Octave uses many of the same commands as Matlab and is free to the public, utilizing this software could save students and universities a significant amount of money paid in licenses to Matlab.

2 The Poisson Equation

We will test the methods on the Poisson equation with homogeneous Dirichlet boundary conditions given by

$$-\triangle u = f \quad \text{in } \Omega, \tag{2.1}$$
$$u = 0 \quad \text{on } \partial\Omega, \tag{2.2}$$

with right-hand side function

$$f(x,y) = -2\pi^2 \cos(2\pi x)\sin^2(\pi y) - 2\pi \sin^2(\pi x)\cos(2\pi y),$$

on the two-dimensional unit square domain $\Omega = (0,1) \times (0,1) \subset \mathbb{R}^2$. In (2.1), the Laplace operator is defined by

$$\triangle u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

and $\partial\Omega$ in (2.2) represents the boundary of the domain Ω. This problem has a closed-form true solution of

$$u(x,y) = \sin^2(\pi x)\, \sin^2(\pi y). \tag{2.3}$$

This problem is discussed in Section 3 of [1] and Section 8.1 of [7].

3 The Finite Difference Method

We first define a grid of mesh points with uniform mesh spacing $h = \frac{1}{N+1}$. Thus, our grid can be represented by $\Omega_h = \{(x_i, y_i) = (ih, jh), i, j = 0, \ldots, N+1\}$. We apply the second-order finite difference approximation (given as (8.1.3) on page 546 of [7]) to the x-derivative at all interior points of Ω_h. This results in an equation that approximates the x-derivative of u at a given (x_i, y_j) through evaluating u at $(x_{i-1}, y_j), (x_i, y_j)$, and (x_{i+1}, y_j). A similar equation results by applying the second-order finite difference approximation to the y-derivative at all interior points of Ω_h. These equations are explicitly stated as (3) and (4) in Section 3 of [1]. Substituting these results in (2.1) produces a system of equations for the approximation of the unknowns $u_{i,j} \approx u(x_i, y_j)$. Since the boundary conditions give $u_{0,j} = u_{i,0} = u_{N+1,j} = u_{i,N+1} = 0$, we see that

$$-u_{i-1,j} - u_{i,j-1} + 4u_{i,j} - u_{i+1,j} - u_{i,j+1} = h^2 f_{i,j} \quad i, j = 1, \ldots, N, \tag{3.1}$$

where $f_{i,j} = f(x_i, y_j)$ for short.

When $N = 3$, we obtain the nine equations

$$\begin{aligned}
4u_{1,1} - u_{2,1} - u_{1,2} &= h^2 f_{1,1}, && (3.2)\\
-u_{1,1} + 4u_{2,1} - u_{3,1} - u_{2,2} &= h^2 f_{2,1}, && (3.3)\\
- u_{2,1} + 4u_{3,1} - u_{3,2} &= h^2 f_{3,1}, && (3.4)\\
-u_{1,1} + 4u_{1,2} - u_{2,2} - u_{1,3} &= h^2 f_{1,2}, && (3.5)\\
- u_{2,1} - u_{1,2} + 4u_{2,2} - u_{3,2} - u_{2,3} &= h^2 f_{2,2}, && (3.6)\\
- u_{3,1} - u_{2,2} + 4u_{3,2} - u_{3,3} &= h^2 f_{3,2}, && (3.7)\\
- u_{1,2} + 4u_{1,3} - u_{2,3} &= h^2 f_{1,3}, && (3.8)\\
- u_{2,2} - u_{1,3} + 4u_{2,3} - u_{3,3} &= h^2 f_{2,3}, && (3.9)\\
- u_{3,2} - u_{2,3} + 4u_{3,3} &= h^2 f_{3,3}. && (3.10)
\end{aligned}$$

When $N = 3$, we see that (3.2) through (3.10) are linear in $u_{i,j}$. Extending this for any N, we see that the N^2 equations produced from (3.1) will also be linear in $u_{i,j}$. Therefore this problem can be organized into $\mathbf{A}\mathbf{u} = \mathbf{b}$, with dimension N^2 where $\mathbf{A} \in \mathbb{R}^{N^2 \times N^2}$ and $\mathbf{u}, \mathbf{b} \in \mathbb{R}^{N^2}$. Since the boundary values are provided, there will be exactly N^2 unknowns. For this system, we see that

$$\mathbf{A} = \begin{bmatrix} S & -I & & & \\ -I & S & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & S & -I \\ & & & -I & S \end{bmatrix} \in \mathbb{R}^{N^2 \times N^2}, \quad \text{where} \quad \mathbf{S} = \begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{bmatrix} \in \mathbb{R}^{N \times N},$$

$\mathbf{I}$ is the $N \times N$ identity matrix, and $\mathbf{b}_m = h^2 f_{i,j}$ where $m = i + (j-1)N$. To produce $\mathbf{A}$ in Matlab and Octave, we note representation of $\mathbf{A}$ as the sum of two Kronecker products. That is, $\mathbf{A}{=}\mathbf{I}\otimes\mathbf{T}+\mathbf{T}\otimes\mathbf{I}$ where $\mathbf{I}$ is the $N \times N$ identity matrix and $\mathbf{T} \in \mathbb{R}^{N\times N}$ is of the same form as $\mathbf{S}$ with 2's on the diagonal instead of 4's.

Clearly, $\mathbf{A}$ is symmetric since $\mathbf{A} = \mathbf{A}^T$. $\mathbf{A}$ is also positive definite because all eigenvalues are positive. This is a result of the Gershgorin disk theorem stated on pages 482-483 of [7]. The 1st and Nth Gershgorin disks, corresponding to the 1st and Nth rows of $\mathbf{A}$, are centered at $a_{11} = a_{NN} = 4$ on the complex plane and have radius $r = 2$. The kth Gershgorin disk, corresponding to the kth row of $\mathbf{A}$ where $2 \le k \le N-1$, is centered at $a_{kk} = 4$ on the complex plane with radius $r \le 4$. Since $\mathbf{A}$ is symmetric, all eigenvalues are real, therefore we know that the eigenvalues, λ, satisfy $0 \le \lambda \le 8$. However, $\mathbf{A}$ is non-singular, therefore we know $\lambda \ne 0$ [7]. Therefore, all eigenvalues are positive, thus the matrix $\mathbf{A}$ is positive definite.

In order to discuss the convergence of the finite difference method, we must obtain the finite difference error. This error is given by the difference between the true solution $u(x, y)$ defined in (2.3) and the numerical solution u_h defined on the mesh points where $u_h(x_i, y_j) = u_{i,j}$. We will use the $L^\infty(\Omega)$ norm defined by $||u - u_h||_{L^\infty(\Omega)} = \sup_{(x,y)\in\Omega} |u(x, y) - u_h(x, y)|$ discussed in [1] to compute this error. The finite difference theory predicts that the quantity $||u - u_h||_{L^\infty(\Omega)} \le Ch^2$ as $h \to 0$, where C is a constant independent of h [1]. As a result, for sufficiently small values of h, we suspect that the ratio of errors between consecutive mesh resolutions is

$$\text{Ratio} = \frac{||u - u_{2h}||_{L^\infty(\Omega)}}{||u - u_h||_{L^\infty(\Omega)}} \approx \frac{C(2h)^2}{Ch^2} = 4.$$

Thus, the finite difference method is second-order convergent if the ratio tends to 4 as N increases [1]. We will print this ratio in our tables so that we can discuss the convergence of the method.

4 The Computing Environment

The UMBC High Performance Computing Facility (HPCF) is the community-based, interdisciplinary core facility for scientific computing and research on parallel algorithms at UMBC. The current machine in HPCF is the 240-node distributed-memory cluster maya. The newest part of the cluster are the 72 nodes with two eight-core 2.6 GHz Intel E5-2650v2 Ivy Bridge CPUs and 64 GB memory that include 19 hybrid nodes with two state-of-the-art NVIDIA K20 GPUs (graphics processing units) designed for scientific computing and 19 hybrid nodes with two cutting-edge 60-core Intel Phi 5110P accelerators. All nodes are connected via InfiniBand to a central storage of more than 750 TB. The calculations in this report were performed on one node with two eight-core 2.6 GHz Intel E5-2650v2 Ivy Bridge CPUs and 64 GB memory using Matlab R2014a (8.3.0.532) and GNU Octave version 3.8.1.

4.1 Matlab

"Matlab is a high-level language and interactive environment for numerical computation, visualization, and programming," as stated on its webpage `www.mathworks.com`. Matlab is available for purchase through its webpage as a student version or for institutional licensing at considerable cost. The software was created by Cleve Moler, a numerical analyst at the University of New Mexico's Computer Science Department. The initial goal was to be able to make LINPACK and EISPACK more easily accessible by students, but it grew when its potential was recognized and Mathworks was established in 1983 [1]. The key features include its high-level language, interactive environment, mathematical functions, and built-in graphics.

4.2 Octave

"GNU Octave is a high-level interpreted language, primarily intended for numerical computations," as stated on its webpage `www.octave.org`. It can be downloaded for free from `http://sourceforge.net/projects/octave`. The software was developed by John W. Eaton, for use with an undergraduate textbook on chemical reaction design. The capabilities of Octave grew when redevelopment was necessary, and its development became a full-time project in 1992. The main features of Octave are similar to Matlab, as it has an interactive command line interface and graphics capabilities.

5 Matlab Results

We will now solve the linear system produced from the Poisson Equation in Matlab R2014a using the various numerical methods. For Gaussian elimination with dense storage, we will use the code available with [1]. Since the code is written in sparse storage, for our calculations in dense storage we will simply insert `A=full(A)` after line 12 in the code. Sparse storage is here shorthand for using sparse storage mode of an array, in which only non-zero elements are stored. By contrast, in dense storage mode, all elements of the array are stored, whatever their value.

For the classical iterative methods, we will write a function `classiter` to solve the system using the Jacobi method, the Gauss-Seidel method, and the successive overrelaxation (SOR(ω)) method. Each of the methods can be written in the form $\mathbf{M}\mathbf{x}^{(k+1)} = \mathbf{N}\mathbf{x}^{(k)} + \mathbf{b}$ where the splitting matrix $\mathbf{M}$ depends on the method and $\mathbf{N} = \mathbf{M} - \mathbf{A}$, and $\mathbf{x}^{(k)}$ is the k^{th} iterate [7]. As a result, we can set up the splitting matrix $\mathbf{M}$ depending on the method and use the same code to compute the iterations. The function `classiter` takes the following input: the system matrix $\mathbf{A}$, right-hand side vector $\mathbf{b}$, tolerance `tol`, maximum number of iterations `maxit`, a parameter `imeth` telling the function which method to use, the SOR relaxation parameter ω, and an initial guess $\mathbf{x}^{(0)}$. The splitting matrix $\mathbf{M}$ is then defined by the following: for the Jacobi method, $\mathbf{M}$ contains the diagonal elements of the system matrix $\mathbf{A}$, `M=spdiags(diag(A),[0],N,N)`; for the Gauss-Seidel method, $\mathbf{M}$ consists of the lower triangular and diagonal entries of $\mathbf{A}$, `M=tril(A)`; and for the SOR(ω) method, $\mathbf{M}$ is defined to be a matrix consisting of the inverse of ω multiplied by the diagonal elements of $\mathbf{A}$ plus the lower triangular elements of $\mathbf{A}$, `M=(1/omega)*spdiags(diag(A),[0],N,N)+tril(A,-1)` [7]. The body of the function computes the iterates using the efficient update formula `x=x+M\r`, where $\mathbf{r}$ is the residual value $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$. The function returns the final iterate $\mathbf{x}^{(k)}$, a flag stating whether the method successfully converged to the desired tolerance before reaching the maximum number of iterations, the final value of the relative residual $||\mathbf{r}^{(k)}||_2/||\mathbf{b}^{(k)}||_2$, the number of iterations taken, and a vector of the residuals `resvec` where `resvec(k)`$= ||\mathbf{r}^{(k)}||_2$. To run this function, we use the code for the conjugate gradient method provided with [1], replacing the call to `pcg` with a call to our function `classiter`. Therefore, line 16 of the code becomes `[u,flag,relres,iter,resvec] = classiter(A,b,tol,maxit,imeth,omega,x0)`.

For the final iterative method, the conjugate gradient method, we will use the code available with [1]. For the conjugate gradient method with SSOR(ω) preconditioning, we must send `pcg` the SSOR splitting matrix $\mathbf{M}$. However, for an efficient implementation, we will send `pcg` the lower triangular factor $\mathbf{M}_1$ and upper triangular factor $\mathbf{M}_2$ such that $\mathbf{M} = \mathbf{M}_1\mathbf{M}_2$ with

$$\mathbf{M}_1 = \sqrt{\frac{\omega}{2-\omega}}\left(\frac{1}{\omega}\mathbf{D} - \mathbf{E}\right)\mathbf{D}^{-1/2}, \quad \mathbf{M}_2 = \sqrt{\frac{\omega}{2-\omega}}\mathbf{D}^{-1/2}\left(\frac{1}{\omega}\mathbf{D} - \mathbf{F}\right),$$

where $\mathbf{D}$ is a diagonal matrix, $\mathbf{E}$ is a strictly lower triangular matrix, and $\mathbf{F}$ is a strictly upper triangular matrix such that $\mathbf{A} = \mathbf{D} - \mathbf{E} - \mathbf{F}$ [7]. We note that $\mathbf{M}_2 = \mathbf{M}_1^T$, and use this for more efficient code. Therefore, we obtain the code for the preconditioned conjugate gradient method in the following way. First, we compute the value of ω. Then, we compute the matrices $\mathbf{D}$, $\mathbf{D}^{-1/2}$ and $\mathbf{E}$. We do this by writing `d = diag(A)`, `D=spdiags(d,0,N^2,N^2)`, `v = 1./sqrt(d)`, `Ds = spdiags(v,0,N^2,N^2)`, and `E=-tril(A,-1)`. Then we compute $\mathbf{M}_1$ and $\mathbf{M}_2$ by `M1 = sqrt(omega/(2-omega))*((1/omega)*D-E)*Ds` and `M2=M1'`. Finally we call `pcg` with the code `[u,flag,relres,iter,resvec] = pcg(A,b,tol,maxit,M1,M2,x0)`.

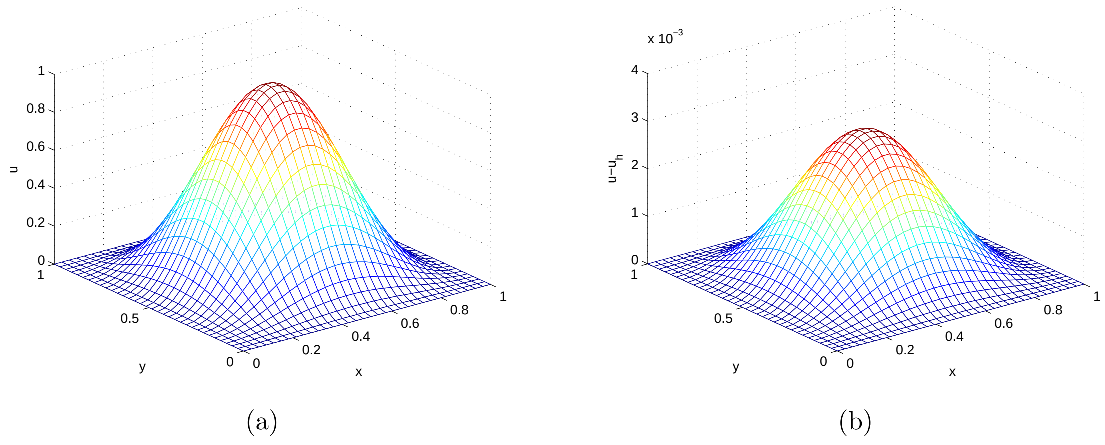


Figure 5.1: Mesh plots for $N = 32$ produced by Matlab using Gaussian elimination, where (a) shows the numerical solution and (b) shows the numerical error.

Figure 5.1 (a) shows the mesh plot of the numerical solution vs. (x, y) for $N = 32$ and Figure 5.1 (b) shows the error at each mesh point computed by subtracting the numerical solution from the analytical solution. Both were produced by solving the system using Gaussian elimination with sparse storage. We will see in the following subsections that since all of the methods converge for $N = 32$, the plots do not depend on the method used to compute the numerical solution. Figure 5.1 (a) shows that the numerical solution is smooth. We also note that the numerical solution is zero at the boundaries as expected. It is clear that the maximum error seen in Figure 5.1 (b) occurs at the center of the domain.

All of the results in this section will be given in tables where for each value of the mesh resolution N, the table lists the number of degrees of freedom, "DOF"$= N^2$, the norm of the finite difference error $\|u - u_h\|_{L^\infty(\Omega)}$, the ratio of consecutive errors, and the observed wall clock time in HH:MM:SS. The tables for the iterative methods will include an additional column representing the number of iterations taken by the method `#iter`. For all of the iterative methods, we will use the zero vector as an initial guess and a tolerance of 10^{-6} on the relative residual of the iterates.

5.1 Gaussian Elimination with Dense Storage

We will begin by solving the linear system using Gaussian elimination with dense storage. The system is solved using the backslash operator in Matlab. Table 5.1 (a) shows the results of implementing this problem using Gaussian elimination with dense storage with mesh resolutions $N = 2^\nu$ for $\nu = 2, \ldots, 14$.

As seen in Table 5.1 (a), as N increases, the norms of the finite difference error go to zero. Also, the ratios of consecutive errors tend to 4 as expected from the discussion at the end of Section 3. This confirms that the finite difference method for this problem is second-order convergent, as discussed in [1]. Table 5.1 (a) also shows that the time taken for $N \leq 256$ is less than 5 minutes. However, the table also shows that when $N = 512$, the computer runs out of memory to store the dense system matrix $\mathbf{A}$. Therefore, we are unable to solve the Poisson problem using finite differences for N larger than 256 using Gaussian elimination with dense storage.

5.2 Gaussian Elimination with Sparse Storage

We will now solve the linear system using Gaussian elimination with sparse storage. Again, the system is solved using the backslash operator. Table 5.1 (b) shows the results of Gaussian elimination with sparse storage applied to the problem with mesh resolutions $N = 2^\nu$ for $\nu = 2, \ldots, 14$.

(a) Gaussian Elimination using Dense Storage in Matlab

N	DOF	$\|u - u_h\|$	Ratio	Time
4	16	1.1673e-1	N/A	<00:00:01
8	64	3.9152e-2	2.9813	<00:00:01
16	256	1.1267e-2	3.4748	<00:00:01
32	1024	3.0128e-3	3.7399	<00:00:01
64	4096	7.7812e-4	3.8719	<00:00:01
128	16384	1.9766e-4	3.9366	00:00:09
256	65536	4.9807e-5	3.9685	00:04:41
512	Out of Memory			
1024	Out of Memory			
2048	Out of Memory			
4096	Out of Memory			
8192	Out of Memory			
16384	Out of Memory			

(b) Gaussian Elimination using Sparse Storage in Matlab

N	DOF	$\|u - u_h\|$	Ratio	Time
4	16	1.1673e-1	N/A	<00:00:01
8	64	3.9152e-2	2.9813	<00:00:01
16	256	1.1267e-2	3.4748	<00:00:01
32	1024	3.0128e-3	3.7399	<00:00:01
64	4096	7.7812e-4	3.8719	<00:00:01
128	16384	1.9766e-4	3.9366	<00:00:01
256	65536	4.9807e-5	3.9685	<00:00:01
512	262144	1.2501e-5	3.9843	<00:00:01
1024	1048576	3.1313e-6	3.9922	00:00:03
2048	4194304	7.8362e-7	3.9960	00:00:15
4096	16777216	1.9607e-7	3.9965	00:01:05
8192	67108864	4.9325e-8	3.9751	00:04:42
16384	Out of Memory			

Table 5.1: Convergence results for the test problem in Matlab using Gaussian elimination with (a) dense storage and (b) sparse storage. For $N = 8{,}192$ using GE with sparse storage, the job was run on the user node since it requires 74.6 GB of memory. Running this job on the compute node results in a runtime of 6 minutes and 55 seconds.

As seen in Table 5.1 (b), as N increases, the norms of the finite difference error go to zero and the ratios of consecutive errors tend to 4 implying convergence. Table 5.1 (b) also shows that the time taken for $N \leq 8{,}192$ is less than 5 minutes. For $N = 8{,}192$, 74.6 GB of memory are required. A compute node only has 64 GB of physical memory. If a job exceeds the 64 GB of physical memory then the the node will use the swap space located on the hard drive. This results in a slower runtime of 6 minutes and 55 seconds. By running on a user node with 128 GB of physical memory we can avoid the use of swap space to obtain a runtime of 4 minutes and 42 seconds. However, when $N = 16{,}384$, even the user node with 128 GB runs out of memory in the computation of the solution. Therefore, we are unable to solve the Poisson problem using finite differences for N larger than 8,192 using Gaussian elimination with sparse storage.

From the results in Tables 5.1 (a) and (b), we see that Gaussian elimination with sparse storage allows for the linear system arising from the Poisson equation to be solved using larger mesh resolutions, N, without running out of memory. For those mesh resolutions for which both methods were able to solve the problem, $N \leq 256$, we see that Gaussian elimination with sparse storage is faster than Gaussian elimination with dense storage. Therefore, we see the advantages of storing sparse matrices, namely matrix $\mathbf{A}$ from this problem, in sparse storage. Using sparse storage reduces the amount of memory required to store the matrix and operations performed using matrices in sparse storage require less memory and are often faster. Therefore, we will utilize sparse storage for the remainder of the methods tested.

5.3 The Jacobi Method

We will begin by solving the linear system using the first of the three classical iterative methods tested, the Jacobi method. Table 5.2 (a) shows the results of implementing this problem using the Jacobi method with mesh resolutions $N = 2^\nu$, for $\nu = 2, \ldots, 14$.

As seen in Table 5.2 (a), as N increases, the norms of the finite difference error appear to be going to zero and the ratios of consecutive errors appear to tend to 4 implying convergence second order convergence. Table 5.2 (a) also shows that the time taken for $N \leq 512$ is less than 1 hour 18 minutes. However, when $N = 1{,}024$, the Jacobi method begins to take an excessive amount of time to solve the problem. Therefore, we are unable to solve the Poisson problem using finite differences for N larger than 512 using the Jacobi method.

5.4 The Gauss-Seidel Method

We will now solve the linear system using the second of the three classical iterative methods tested, the Gauss-Seidel method. Table 5.2 (b) shows the results of implementing this problem using the Gauss-Seidel method with mesh resolutions $N = 2^\nu$, for $\nu = 2, \ldots, 14$.

As seen in Table 5.2 (b), as N increases, the norms of the finite difference error appear to be going to zero and the ratios of consecutive errors appear to tend to 4 implying second order convergence. Table 5.2 (b) also shows that the time taken for $N \leq 1{,}024$ is less than 12 hours. However, when $N = 2{,}048$, the Gauss-Seidel method begins to take an excessive amount of time to solve the problem. Therefore, we are unable to solve the Poisson problem using finite differences for N larger than 2,048 using the Gauss-Seidel method.

5.5 The Successive Overrelaxation Method

We will now solve the linear system using the third of the three classical iterative methods tested, the successive overrelaxation method (SOR(ω)). The optimal ω, $\omega_{opt} = 2/(1 + \sin(\pi h))$, defined in [7] and [8] was used. Table 5.2 (c) shows the results of implementing this problem using the SOR(ω_{opt}) method with mesh resolutions $N = 2^\nu$, for $\nu = 2, \ldots, 14$.

As seen in Table 5.2 (c), as N increases, ω_{opt} tends to 2. We also note that the norms of the finite difference error appear to be going to zero while the ratios of consecutive errors appear to tend to 4 implying second order convergence. For the two finest mesh resolutions, the reduction in error appears slightly more erratic, which points to the tolerance of the iterative linear solver not being tight enough to ensure a sufficiently accurate solution of the linear system. Table 5.2 (c) also shows that the time taken for $N \leq 4{,}096$ is less than 2 hours. However, when $N = 8{,}192$, the method begins to take an excessive amount of time to solve the problem. Therefore, we are unable to solve the Poisson problem using finite differences for N larger than 4,096 using the SOR(ω_{opt}) method.

From the results in Tables 5.2 (a), (b) and (c), we see that the SOR(ω_{opt}) method allows for the system to be solved with larger mesh resolutions N than both the Jacobi and Gauss-Seidel methods. This is a result of the fact that the number of iterations required to solve the linear system for a given mesh resolution N decreases when the method is changed from Jacobi to Gauss-Seidel to SOR(ω_{opt}). Therefore, the SOR(ω_{opt}) method is the best method among the three classical iterative methods tested. However, we see that none of the classical iterative methods has any advantage over Gaussian elimination, which as able to solve a larger problem faster, provided sparse storage is used.

5.6 The Conjugate Gradient Method

We will now solve the linear system using the conjugate gradient method. Table 5.3 (a) shows the results of the conjugate gradient method applied to the problem with mesh resolutions $N = 2^\nu$ for $\nu = 2, \ldots, 14$.

(a) Jacobi Method in Matlab

N	DOF	$\|u - u_h\|$	Ratio	`#iter`	Time
4	16	1.1673e-1	N/A	64	00:00:02
8	64	3.9151e-2	2.9814	214	<00:00:01
16	256	1.1266e-2	3.4751	769	<00:00:01
32	1024	3.0114e-3	3.7411	2902	<00:00:01
64	4096	7.7673e-4	3.8771	11258	00:00:01
128	16384	1.9626e-4	3.9577	44331	00:00:17
256	65536	4.8397e-5	4.0551	175921	00:03:57
512	262144	1.1089e-5	4.3646	700881	01:06:10
1024		Excessive Time required			
2048		Excessive Time required			
4096		Excessive Time required			
8192		Excessive Time required			
16384		Excessive Time required			

(b) Gauss-Seidel Method in Matlab

N	DOF	$\|u - u_h\|$	Ratio	`#iter`	Time
4	16	1.1673e-1	N/A	33	<00:00:01
8	64	3.9151e-2	2.9814	108	<00:00:01
16	256	1.1266e-2	3.4751	386	<00:00:01
32	1024	3.0114e-3	3.7411	1452	<00:00:01
64	4096	7.7673e-4	3.8771	5630	<00:00:01
128	16384	1.9626e-4	3.9577	22166	00:00:11
256	65536	4.8398e-5	4.0551	87962	00:02:44
512	262144	1.1089e-5	4.3645	350442	00:46:42
1024	1048576	1.7182e-6	6.4538	1398959	11:23:42
2048		Excessive Time required			
4096		Excessive Time required			
8192		Excessive Time required			
16384		Excessive Time required			

(c) Successive Overrelaxation Method (SOR(ω_{opt})) in Matlab

N	DOF	ω_{opt}	$\|u - u_h\|$	Ratio	`#iter`	Time
4	16	1.2596	1.1673e-1	N/A	14	<00:00:01
8	64	1.4903	3.9152e-2	2.9813	25	<00:00:01
16	256	1.6895	1.1267e-2	3.4749	47	<00:00:01
32	1024	1.8264	3.0125e-3	3.7401	92	<00:00:01
64	4096	1.9078	7.7785e-4	3.8729	181	<00:00:01
128	16384	1.9525	1.9737e-4	3.9410	359	<00:00:01
256	65536	1.9758	4.9522e-5	3.9856	716	00:00:01
512	262144	1.9878	1.2214e-5	4.0544	1429	00:00:12
1024	1048576	1.9939	2.8630e-6	4.2662	2861	00:01:33
2048	4194304	1.9969	5.5702e-7	5.1399	5740	00:13:22
4096	16777216	1.9985	4.6471e-7	1.1986	11559	01:48:06
8192			Excessive Time required			
16384			Excessive Time required			

Table 5.2: Convergence results for the test problem in Matlab using (a) the Jacobi method, (b) the Gauss-Seidel method, and (c) the SOR(ω_{opt}) method. Excessive time corresponds to more than 12 hours wall clock time.

(a) The Conjugate Gradient Method in Matlab					
N	DOF	$\|u-u_h\|$	Ratio	`#iter`	Time
4	16	1.1673e-1	N/A	3	00:00:01
8	64	3.9152e-2	2.9813	10	<00:00:01
16	256	1.1267e-2	3.4748	24	<00:00:01
32	1024	3.0128e-3	3.7399	48	<00:00:01
64	4096	7.7811e-4	3.8719	96	<00:00:01
128	16384	1.9765e-4	3.9368	192	<00:00:01
256	65536	4.9797e-5	3.9690	387	<00:00:01
512	262144	1.2494e-5	3.9856	783	00:00:06
1024	1048576	3.1266e-6	3.9961	1581	00:00:46
2048	4194304	7.8019e-7	4.0075	3192	00:06:44
4096	16777216	1.9366e-7	4.0287	6452	00:53:30
8192	67108864	4.7375e-8	4.0878	13033	07:12:30
16384	Excessive Time required				

(b) The Conjugate Gradient Method with SSOR(ω_{opt}) Preconditioning in Matlab						
N	DOF	ω_{opt}	$\|u-u_h\|$	Ratio	`#iter`	Time
4	16	1.2596	1.1673e-1	N/A	7	<00:00:01
8	64	1.4903	3.9153e-2	2.9813	9	<00:00:01
16	256	1.6895	1.1267e-2	3.4748	14	<00:00:01
32	1024	1.8264	3.0128e-3	3.7399	19	<00:00:01
64	4096	1.9078	7.7812e-4	3.8719	28	<00:00:01
128	16384	1.9525	1.9766e-4	3.9366	40	<00:00:01
256	65536	1.9758	4.9811e-5	3.9683	57	<00:00:01
512	262144	1.9878	1.2502e-5	3.9842	83	00:00:02
1024	1048576	1.9939	3.1321e-6	3.9917	121	00:00:10
2048	4194304	1.9969	7.8394e-7	3.9953	176	00:00:55
4096	16777216	1.9985	1.9620e-7	3.9957	256	00:05:21
8192	67108864	1.9992	4.9109e-8	3.9952	375	00:30:51
16384	268435456	1.9996	1.2301e-8	3.9923	548	03:07:29

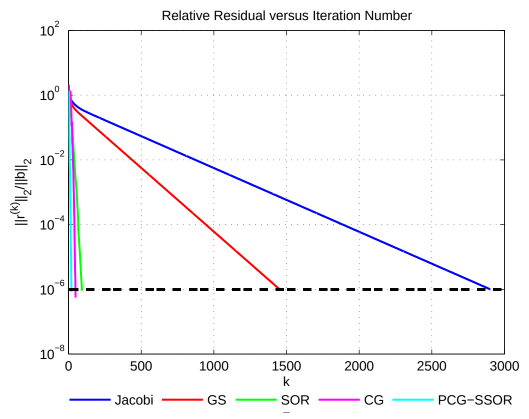
Table 5.3: Convergence results for the test problem in Matlab using (a) the conjugate gradient method and (b) the conjugate gradient method with SSOR(ω_{opt}) preconditioning. For $N = 16{,}384$ using PCG, the job was run on the user node since it requires 87.5 GB. It is also possible to run this on the compute node, however the runtime is over 24 hours. Excessive time requirement corresponds to more than 12 hours wall clock time.

As seen in Table 5.3 (a), as N increases, the norms of the finite difference error go to zero and the ratios of consecutive errors tend to 4 implying second order convergence. Table 5.3 (a) also shows that the time taken for $N \leq 8{,}192$ is less than 7 hours and 13 minutes. However, when $N = 16{,}384$, the computer begins to require excessive time to solve the problem. Therefore, we are unable to solve the Poisson problem using finite differences for N larger than 8,192 using the conjugate gradient method.

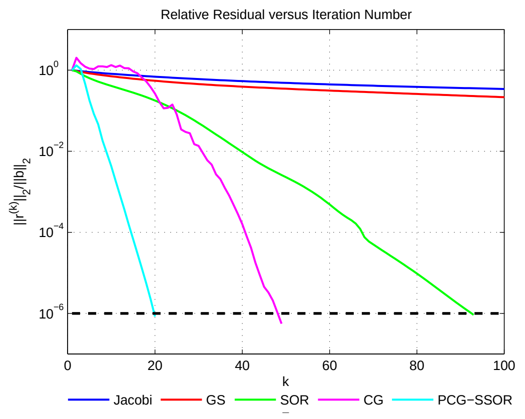
5.7 The Conjugate Gradient Method with Preconditioning

We will now solve the linear system using the conjugate gradient method with symmetric successive overrelaxation (SSOR(ω)) preconditioning. We will use the optimal ω, as discussed in Section 5.5. Table 5.3 (b) shows the results of the conjugate gradient method with SSOR(ω_{opt}) preconditioning applied to the problem with mesh resolutions $N = 2^\nu$ for $\nu = 2, \ldots, 14$.

As seen in Table 5.3 (b), as N increases, the norms of the finite difference error go to zero and the ratios of consecutive errors tend to 4 implying convergence. Table 5.3 (b) also shows that the time taken



(a)



(b)

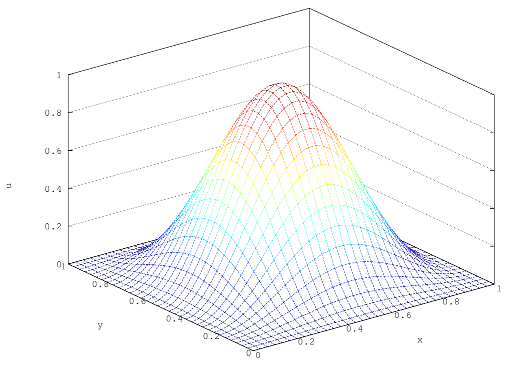
Figure 5.2: The relative residual versus iteration number k for $N = 32$ for all iterative methods where "GS" represents Gauss-Seidel, "SOR" represents successive overrelaxation, "CG" represents conjugate gradient, and "PCG-SSOR" represents conjugate gradient with symmetric successive overrelaxation preconditioning. Figure (a) shows the graph to the maximum number of iterations required for all methods and (b) shows the plot to $k = 100$. The dashed black line represents the desired tolerance, 10^{-6}. These figures were produced in Matlab.

for $N \leq 16{,}384$ is less than 3 hours and 8 minutes. For $N = 16{,}384$ using PCG, the job required 87.5 GB of memory to run. This was completed in over 24 hours by using the swap space on a compute node. By using a user node with 128 GB of memory and thus not requiring the use of swap space, the runtime is significantly reduced. However, when $N = 32{,}768$, the computer runs out of memory to store the system matrix $\mathbf{A}$. Therefore, we are unable to solve the Poisson problem using finite differences for N larger than 16,384 using the conjugate gradient method with SSOR(ω_{opt}) preconditioning.

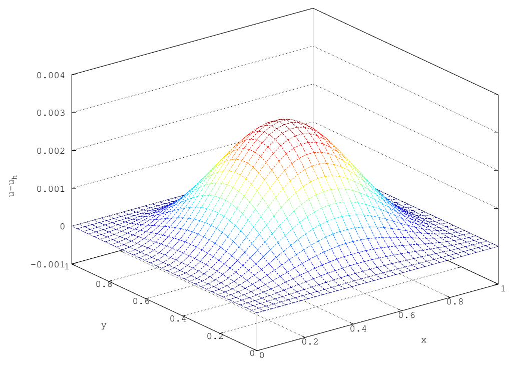
From the results in Tables 5.3 (a) and (b), we see that the conjugate gradient method with SSOR(ω_{opt}) preconditioning converges with an order of magnitude reduction in iterations as well as run time. This demonstrates that the added cost of preconditioning in each iteration is worth the cost. The results for conjugate gradients both without and with preconditioning show that that their decreased memory requirements allow to solve larger problems than Gaussian elimination. But only the preconditioned conjugate gradient method is efficient enough to solve the larger problems in reasonable amount of time.

5.8 A Comparison of the Iterative Methods

To compare all iterative methods graphically, we will plot the relative residuals $||r^{(k)}||_2/||b||_2$ versus the iteration number k for $N = 32$ in a semi-log plot shown in Figure 5.2 (a) and (b). This plot is produced using the `semilogy` command in Matlab. It is clear from Figure 5.2 (a) that the Jacobi and Gauss-Seidel methods take far more iterations to converge than the remaining three iterative methods. Figure 5.2 (b) zooms in on the plot so that the behavior between SOR(ω_{opt}), the conjugate gradient method, and the conjugate gradient method with SSOR(ω_{opt}) preconditioning can be compared. It is clear from Figure 5.2 (b) that the conjugate gradient method with SSOR(ω_{opt}) preconditioning is superior to the conjugate gradient method without preconditioning, which in turn is superior to SOR(ω_{opt}). Figure 5.2 (b) also shows that the conjugate gradient methods experience a slight increase in the relative residual within the first 10 iterations. However, once these iterations are complete, the superiority of the methods shine through in the quick convergence rate.



(a)



(b)

Figure 6.1: Mesh plots for $N = 32$ produced by Octave using Gaussian elimination, where (a) shows the numerical solution and (b) shows the numerical error.

6 Octave Results

We will now solve the linear system produced from the Poisson Equation in GNU Octave version 3.8.1 using the various numerical methods. Since our Matlab code is compatible with Octave, we will use the same code which we described in Section 5. For all of the iterative methods, we will use the zero vector as an initial guess and a tolerance of 10^{-6} on the relative residual of the iterates.

Figure 6.1 (a) shows the mesh plot of the numerical solution vs. (x, y) for $N = 32$ and Figure 6.1 (b) shows the error at each mesh point computed by subtracting the numerical solution from the analytical solution. Both were produced by solving the system using Gaussian elimination with sparse storage. We see that Figure 6.1 is very similar to Figure 5.1 and both confirm that the numerical solution is smooth and the maximum error occurs at the center of the domain.

All of the results in this Section will be given in tables where for each value of the mesh resolution N, the table lists the number of degrees of freedom "DOF"$= N^2$, the norm of the finite difference error $\|u - u_h\|_{L^\infty(\Omega)}$, the ratio of consecutive errors, and the observed wall clock time in HH:MM:SS. The tables for the iterative methods will include an additional column representing the number of iterations taken by the method `#iter`.

6.1 Gaussian Elimination with Dense Storage

We will begin by solving the linear system using Gaussian elimination with dense storage. The system is solved using the backslash operator in Octave. Table 6.1 (a) shows the results of implementing this problem using Gaussian elimination with dense storage with mesh resolutions $N = 2^\nu$, for $\nu = 2, \ldots, 14$.

As seen in Table 6.1 (a), as N increases, the norms of the finite difference error go to zero and the ratios of consecutive errors tend to 4 as expected. Table 6.1 (a) also shows that the time taken for $N \leq 128$ is less than 4 minutes, as fractions of a minute were rounded up. Comparing this to Table 5.1 (a), we see that for $N = 128$, Matlab is able to solve the problem faster. Octave runs out of memory when $N = 256$, however Matlab is able to solve the problem for this larger mesh resolution. Therefore, when solving problems with Gaussian elimination with dense storage, Matlab is superior to Octave.

6.2 Gaussian Elimination with Sparse Storage

We will now solve the linear system using Gaussian elimination with sparse storage. Again, the system is solved using the backslash operator. Table 6.1 (b) shows the results of Gaussian elimination with sparse

(a) Gaussian Elimination using Dense Storage in Octave

N	DOF	$\|u - u_h\|$	Ratio	Time
4	16	1.1673e-1	N/A	<00:00:01
8	64	3.9152e-2	2.9813	<00:00:01
16	256	1.1267e-2	3.4748	<00:00:01
32	1024	3.0128e-3	3.7399	<00:00:01
64	4096	7.7812e-4	3.8719	00:00:05
128	16384	1.9766e-4	3.9366	00:04:00
256		out of memory		
512		out of memory		
1024		out of memory		
2048		out of memory		
4096		out of memory		
8192		out of memory		
16384		out of memory		

(b) Gaussian Elimination using Sparse Storage in Octave

N	DOF	$\|u - u_h\|$	Ratio	Time
4	16	1.1673e-1	N/A	<00:00:01
8	64	3.9152e-2	2.9813	<00:00:01
16	256	1.1267e-2	3.4748	<00:00:01
32	1024	3.0128e-3	3.7399	<00:00:01
64	4096	7.7812e-4	3.8719	<00:00:01
128	16384	1.9766e-4	3.9366	<00:00:01
256	65536	4.9807e-5	3.9685	<00:00:01
512	262144	1.2501e-5	3.9843	<00:00:01
1024	1048576	3.1313e-6	3.9922	00:00:06
2048	4194304	7.8362e-7	3.9960	00:00:37
4096	16777216	1.9607e-7	3.9966	00:04:21
8192		out of memory		
16384		out of memory		

Table 6.1: Convergence results for the test problem in Octave using Gaussian elimination with (a) dense storage and (b) sparse storage.

storage applied to the problem with mesh resolutions $N = 2^\nu$ for $\nu = 2, \ldots, 14$.

As seen in Table 6.1 (b), as N increases, the norms of the finite difference error go to zero and the ratios of consecutive errors tend to 4 as expected. Table 6.1 (b) also shows that the time taken for $N \leq 4{,}096$ is less than 5 minutes. Comparing this to Table 5.1 (b), we see that for $N \leq 512$, Octave was able to solve the problem in almost the same amount of time as Matlab. However, for $1{,}024 \leq N \leq 4{,}096$, Octave requires almost twice the amount of time required by Matlab. Additionally, Octave runs out of memory when $N = 8{,}192$, but Matlab is able to solve the problem for this larger mesh resolution. Therefore, when solving problems with Gaussian elimination with sparse storage with large mesh resolutions, Matlab should be used.

From the results in Tables 6.1 (a) and (b), we see that Gaussian elimination with sparse storage allows for the linear system arising from the Poisson equation to be solved using larger mesh resolutions, N, without running out of memory and is faster. Therefore, we will utilize sparse storage for the remainder of the methods tested.

6.3 The Jacobi Method

We will begin by solving the linear system using the first of the three classical iterative methods tested, the Jacobi method. Table 6.2 (a) shows the results of implementing this problem using the Jacobi method with mesh resolutions $N = 2^\nu$, for $\nu = 2, \ldots, 14$.

As seen in Table 6.2 (a), as N increases, the norms of the finite difference error appear to be going to zero and the ratios of consecutive errors appear to tend to 4 implying second order convergence. Table 6.2 (a) also shows that the time taken for $N \leq 512$ is less than 1 hour 42 minutes. Comparing this to Table 5.2 (a), we see that for $N \leq 32$, Octave was able to solve the problem in almost the same amount of time as Matlab. However, when $N \geq 32$, Octave takes almost two times the amount of time taken by Matlab to solve the problem. Therefore, when solving problems with the Jacobi method using a large mesh resolution, Matlab is superior to Octave. Also, as with Matlab, the table shows that when $N = 1{,}024$, the Jacobi method begins to take an excessive amount of time to solve the problem.

6.4 The Gauss-Seidel Method

We will now solve the linear system using the second of the three classical iterative methods tested, the Gauss-Seidel method. Table 6.2 (b) shows the results of implementing this problem using the Gauss-Seidel method with mesh resolutions $N = 2^\nu$, for $\nu = 2, \ldots, 14$.

As seen in Table 6.2 (b), as N increases, the norms of the finite difference error appear to be going to zero and the ratios of consecutive errors appear to tend to 4 implying second order convergence. Table 6.2 (b) also shows that the time taken for $N \leq 512$ is less than 1 hour and 5 minutes. Comparing this to Table 5.2 (b), we see that for $N \leq 128$, Octave was able to solve the problem in almost the same amount of time as Matlab. However, when $N \geq 256$, Octave takes about one and a half times the amount of time taken by Matlab to solve the problem. Therefore, when solving problems with the Gauss-Seidel method using a large mesh resolution, Matlab is superior to Octave. Also, the table shows that when $N = 1{,}024$, the Gauss-Seidel method takes more time than we consider reasonable, while Matlab is able to return a result in a reasonable amount of time.

6.5 The Successive Overrelaxation Method

We will now solve the linear system using the third of the three classical iterative methods tested, the successive overrelaxation method (SOR(ω)). The optimal ω, as discussed in Section 5.5$\omega_{opt} = 2/(1+\sin(\pi h))$ will be used. Table 6.2 (c) shows the results of implementing this problem using the SOR(ω_{opt}) method with mesh resolutions $N = 2^\nu$, for $\nu = 2, \ldots, 14$.

As seen in Table 6.2 (c), as N increases, ω_{opt} tends to 2. We also note that the norms of the finite difference error appear to be going to zero while the ratios of consecutive errors appear to tend to 4, implying second order convergence. For the two finest mesh resolutions, the reduction in error appears slightly more erratic, which points to the tolerance of the iterative linear solver not being tight enough to ensure a sufficiently accurate solution of the linear system. Table 6.2 (c) also shows that the time taken for $N \leq 4{,}096$ is less than 2 hours 41 minutes. Comparing this to Table 5.2 (c), we see that for $N \leq 512$, Octave was able to solve the problem in almost the same amount of time as Matlab. However, when $N \geq 1{,}024$, Octave takes between almost one and a half times the amount of time taken by Matlab to solve the problem. Also, similar to the Matlab results, the table shows that when $N = 8{,}192$, the SOR method begins to take an excessive amount of time to solve the problem.

From the results in Tables 6.2 (a), (b) and (c), we see that the SOR(ω_{opt}) method allows for the system to be solved for larger mesh resolutions N than the Gauss-Seidel and Jacobi methods. Therefore, the SOR(ω_{opt}) method is the best method among the three classical iterative methods tested.

6.6 The Conjugate Gradient Method

We will now solve the linear system using the conjugate gradient method. Table 6.3 (a) shows the results of the conjugate gradient method applied to the problem with mesh resolutions $N = 2^\nu$ for $\nu = 2, \ldots, 14$.

(a) Jacobi Method in Octave					
N	DOF	$\|u-u_h\|$	Ratio	`#iter`	Time
4	16	1.1673e-01	N/A	64	<00:00:01
8	64	3.9151e-02	2.9814	214	<00:00:01
16	256	1.1266e-02	3.4751	769	<00:00:01
32	1024	3.0114e-03	3.7411	2902	<00:00:01
64	4096	7.7673e-04	3.8771	11258	00:00:02
128	16384	1.9626e-04	3.9577	44331	00:00:23
256	65536	4.8397e-05	4.0551	175921	00:05:47
512	262144	1.1089e-05	4.3646	700881	01:41:00
1024		Excessive Time required			
2048		Excessive Time required			
4096		Excessive Time required			
8192		Excessive Time required			
16384		Excessive Time required			

(b) Gauss-Seidel Method in Octave					
N	DOF	$\|u-u_h\|$	Ratio	`#iter`	Time
4	16	1.1673e-01	N/A	33	<00:00:01
8	64	3.9151e-02	2.9814	108	<00:00:01
16	256	1.1266e-02	3.4751	386	<00:00:01
32	1024	3.0114e-03	3.7411	1452	<00:00:01
64	4096	7.7673e-04	3.8771	5630	<00:00:01
128	16384	1.9626e-04	3.9577	22166	00:00:13
256	65536	4.8398e-05	4.0551	87962	00:03:25
512	262144	1.1089e-05	4.3645	350442	01:04:33
1024		Excessive Time required			
2048		Excessive Time required			
4096		Excessive Time required			
8192		Excessive Time required			
16384		Excessive Time required			

(c) Successive Overrelaxation Method (SOR(ω_{opt})) in Octave						
N	DOF	ω_{opt}	$\|u-u_h\|$	Ratio	`#iter`	Time
4	16	1.2596	1.1673e-01	N/A	14	<00:00:01
8	64	1.4903	3.9152e-02	2.9813	25	<00:00:01
16	256	1.6895	1.1267e-02	3.4749	47	<00:00:01
32	1024	1.8264	3.0125e-03	3.7401	92	<00:00:01
64	4096	1.9078	7.7785e-04	3.8729	181	<00:00:01
128	16384	1.9525	1.9737e-04	3.9410	359	<00:00:01
256	65536	1.9758	4.9522e-05	3.9856	716	00:00:02
512	262144	1.9878	1.2214e-05	4.0544	1429	00:00:16
1024	1048576	1.9939	2.8630e-06	4.2662	2861	00:02:13
2048	4194304	1.9969	5.5702e-07	5.1399	5740	00:18:37
4096	16777216	1.9985	4.6471e-07	1.1986	11559	02:40:15
8192			Excessive Time required			
16384			Excessive Time required			

Table 6.2: Convergence results for the test problem in Octave using (a) the Jacobi method, (b) the Gauss-Seidel method, and (c) the SOR(ω_{opt}) method. Excessive time requirement corresponds to more than 12 hours wall clock time.

(a) The Conjugate Gradient Method in Octave

N	DOF	$\|u - u_h\|$	Ratio	`#iter`	Time
4	16	1.1673e-01	N/A	3	<00:00:01
8	64	3.9152e-02	2.9813	10	<00:00:01
16	256	1.1267e-02	3.4748	24	<00:00:01
32	1024	3.0128e-03	3.7399	48	<00:00:01
64	4096	7.7811e-04	3.8719	96	<00:00:01
128	16384	1.9765e-04	3.9368	192	<00:00:01
256	65536	4.9797e-05	3.9690	387	<00:00:01
512	262144	1.2494e-05	3.9856	783	00:00:07
1024	1048576	3.1266e-06	3.9961	1581	00:01:06
2048	4194304	7.8019e-07	4.0075	3192	00:10:08
4096	16777216	1.9354e-07	4.0311	6452	01:21:49
8192	67108864	4.6775e-08	4.1377	13033	11:06:22
16384	Excessive Time required				

(b) The Conjugate Gradient Method with SSOR(ω_{opt}) Preconditioning in Octave

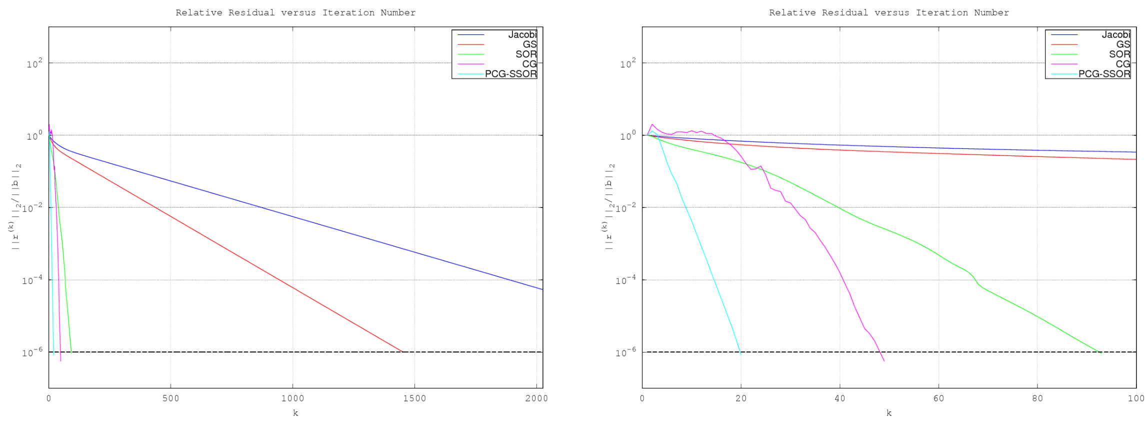
N	DOF	ω_{opt}	$\|u - u_h\|$	Ratio	`#iter`	Time
4	16	1.2596	1.1673e-01	N/A	7	<00:00:01
8	64	1.4903	3.9153e-02	2.9813	9	<00:00:01
16	256	1.6895	1.1267e-02	3.4748	14	<00:00:01
32	1024	1.8264	3.0128e-03	3.7399	19	<00:00:01
64	4096	1.9078	7.7812e-04	3.8719	28	<00:00:01
128	16384	1.9525	1.9766e-04	3.9366	40	<00:00:01
256	65536	1.9758	4.9811e-05	3.9683	57	<00:00:01
512	262144	1.9878	1.2502e-05	3.9842	83	00:00:02
1024	1048576	1.9939	3.1321e-06	3.9917	121	00:00:09
2048	4194304	1.9969	7.8394e-07	3.9953	176	00:00:57
4096	16777216	1.9985	1.9618e-07	3.9961	257	00:05:41
8192	67108864	1.9992	4.9102e-08	3.9953	376	00:32:07
16384	Excessive Time required					

Table 6.3: Convergence results for the test problem in Octave using (a) the conjugate gradient method and (b) the conjugate gradient method with SSOR(ω_{opt}) preconditioning. Excessive time requirement corresponds to more than 12 hours wall clock time.

As seen in Table 6.3 (a), as N increases, the norms of the finite difference error go to zero and the ratios of consecutive errors tend to 4 as expected, implying second order convergence. Table 6.3 (a) also shows that the time taken for $N \leq 8{,}192$ is less than 11 hours and 7 minutes. Comparing this to Table 5.3 (a), we see that for small N values, Octave requires about the same amount of time as Matlab and for larger N values Octave takes between almost one and a half times the amount of time taken by Matlab to solve the problem. Therefore, when solving problems with the conjugate gradient method using a large mesh resolution, Matlab is superior to Octave, although both perform well. Also, similar to the Matlab results, the table shows that when $N = 16{,}384$, the conjugate gradient begins to take an excessive amount of time to solve the problem.

6.7 The Conjugate Gradient Method with Preconditioning

We will now solve the linear system using the conjugate gradient method with symmetric successive overrelaxation (SSOR(ω)) preconditioning. We will use the optimal ω, as discussed in Section 5.5. Table 6.3 (b) shows the results of the conjugate gradient method with SSOR(ω_{opt}) preconditioning applied to the problem with mesh resolutions $N = 2^\nu$ for $\nu = 2, \ldots, 14$.



(a) (b)

Figure 6.2: The relative residual versus iteration number k for $N = 32$ for all iterative methods where "GS" represents Gauss-Seidel, "SOR" represents successive overrelaxation, "CG" represents conjugate gradient, and "PCG-SSOR" represents conjugate gradient with symmetric successive overrelaxation preconditioning. Figure (a) shows the graph to the maximum number of iterations required for all methods and (b) shows the plot to $k = 100$. The black line represents the desired tolerance, 10^{-6}. These figures were produced in Octave.

As seen in Table 6.3 (b), as N increases, the norms of the finite difference error go to zero and the ratios of consecutive errors tend to 4 as expected. Table 6.3 (b) also shows that the time taken for $N \leq 8{,}192$ is less than 33 minutes. Comparing this to Table 5.3 (b), we see that for all N values available in Table 6.3 (b), Octave requires about the same time to solve the problem as Matlab. However, Octave requires excessive time to solve the problem when $N = 16{,}384$ regardless of whether it was run on a compute node with 64 GB of memory or an user node with 128 GB of memory, while Matlab could solve this mesh resolution in a little over 3 hours on the user node. Therefore, when solving problems with the conjugate gradient method with symmetric successive overrelaxation (SSOR(ω)) preconditioning for a large mesh resolution, Matlab is superior to Octave.

From the results in Tables 6.3 (a) and (b), we see that the conjugate gradient method with SSOR(ω_{opt}) preconditioning converges with an order of magnitude fewer iterations and the time to reach convergence decreases by a factor of ten. Therefore, we see the advantages of using a carefully selected preconditioning matrix.

6.8 A Comparison of the Iterative Methods

To compare all iterative methods graphically, we will plot the relative residuals $||r^{(k)}||_2/||b||_2$ versus the iteration number k for $N = 32$ in a semi-log plot shown in Figure 6.2 (a) and (b). This plot is produced using the `semilogy` command in Octave. Figures 6.2 (a) and (b) are clearly similar to Figures 5.2 (a) and (b) which were produced by Matlab. It is clear from Figure 6.2 (a) that the Jacobi and Gauss-Seidel methods take far more iterations to converge than the remaining three iterative methods. It is clear from Figure 6.2 (b) that the conjugate gradient method with SSOR(ω_{opt}) preconditioning is superior to the conjugate gradient method without preconditioning, which in turn is superior to SOR(ω_{opt}).

7 Comparisons and Conclusions

We will now compare the methods discussed in Sections 5 and 6. We noted that Gaussian elimination was more efficient with sparse storage than dense storage. We saw that SOR(ω_{opt}) was the most efficient of the three classical iterative methods tested; however, it is slower than both Gaussian elimination and conjugate gradients without being able to solve larger problems than these. We concluded that the conjugate gradient method with SSOR(ω_{opt}) preconditioning was more efficient than the conjugate gradient method without preconditioning. Therefore, we will compare now only Gaussian elimination with sparse storage and the conjugate gradient method with SSOR(ω_{opt}) preconditioning.

Gaussian elimination with sparse storage in Matlab was able to solve the system for mesh resolutions up to $N = 8{,}192$ the fastest. Therefore, it is a good method to use for systems with sparse coefficient matrices and relatively small mesh sizes. However, for $N = 16{,}384$, this method runs out of memory. Gaussian elimination with sparse storage in Octave was only able to solve the problem for mesh resolutions up to $N = 4{,}096$, and ran out of memory when $N = 8{,}192$. For those mesh resolutions in which both Matlab and Octave were able to solve the problem, Octave was significantly slower with the highest N value. Therefore when solving a system using Gaussian elimination with sparse storage, Matlab should be used when a large value of N is needed.

The conjugate gradient method with SSOR(ω_{opt}) preconditioning in Matlab is the best method for solving a linear system for mesh resolutions larger than $N = 8{,}192$. This method did take longer than Gaussian elimination with sparse storage in Matlab for mesh resolutions $N \leq 8{,}192$, however it was quicker than the SOR(ω_{opt}) method. It was able to solve the system with a mesh resolution of $N = 16{,}384$ in Matlab in just over 3 hours. The conjugate gradient method with SSOR(ω_{opt}) preconditioning in Octave was only able to solve the problem for mesh resolutions up to $N = 8{,}192$. For those mesh resolutions in which Matlab and Octave were both able to solve the problem, Matlab and Octave required almost the same amount of time. Therefore, if one needs to solve a problem with a very large mesh resolution, the conjugate gradient method with SSOR(ω_{opt}) preconditioning in Matlab should be used. Otherwise, if one needs to solve a problem using the conjugate gradient method with SSOR(ω_{opt}) preconditioning for a smaller mesh size, both Matlab and Octave are good software choices.

Overall, it is important to consider the size of the mesh resolution used to determine which method and software to use to solve the system. If the mesh resolution is very large, in this case $N = 16{,}384$, the only method that will solve the problem is the conjugate gradient method with SSOR(ω_{opt}) preconditioning in Matlab. If $N \leq 8{,}192$, both Matlab and Octave can be used. If $N = 8{,}192$, Gaussian elimination with sparse storage in Matlab will solve the problem the fastest. However, if Octave is used, the conjugate gradient method with SSOR(ω_{opt}) preconditioning will solve the problem fastest without running out of memory. For small N values, both Matlab and Octave can be used, and Gaussian elimination with sparse storage should be used since it can solve the problem fastest.

We conclude that for problems of the textbook type typically assigned in a course, Octave would be a sufficient software package. However, once one branches into more in-depth mathematics, Matlab should be used to provide faster results for problems.

We return now to the previous report on this test problem by comparing the timings on maya in this report with the timings on tara reported in [1]. Precisely, for the results on the conjugate gradient method, we compare Table 5.3 (a) here with Table 3.1 (b) in [1] for Matlab and Table 6.3 (a) here with Table 3.2 (b) in [1] for Octave. The timings for the conjugate gradient method on maya are twice as fast as those were on tara in each case. For the results on Gaussian elimination with sparse storage, we compare Table 5.1 (b) here with Table 3.1 (a) in [1] for Matlab and Table 6.1 (b) here with Table 3.2 (a) in [1] for Octave. Gaussian elimination in Octave solved the same size of problem on maya as on tara before running out of memory; the speed improved by a factor of about three in all cases. Gaussian elimination in Matlab was able to solve the problem on a finer mesh on maya than on tara; the speed improved modestly, starting from already excellent timings on tara in [1]. The speedup observed here for Matlab and Octave from tara to maya is consistent with the speedup for serial jobs in the C programming language for this test problem reported in [5].

Acknowledgments

The hardware used in the computational studies is part of the UMBC High Performance Computing Facility (HPCF). The facility is supported by the U.S. National Science Foundation through the MRI program (grant nos. CNS–0821258 and CNS–1228778) and the SCREMS program (grant no. DMS–0821311), with additional substantial support from the University of Maryland, Baltimore County (UMBC). See `www.umbc.edu/hpcf` for more information on HPCF and the projects using its resources. This project began as the class project of the first author for Math 630 Numerical Linear Algebra during the Spring 2014 semester [6]. The second author acknowledges financial support as HPCF RA.

References

[1] Ecaterina Coman, Matthew W. Brewster, Sai K. Popuri, Andrew M. Raim, and Matthias K. Gobbert. A comparative evaluation of Matlab, Octave, FreeMat, Scilab, R, and IDL on tara. Technical Report HPCF–2012–15, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2012.

[2] James W. Demmel. *Applied Numerical Linear Algebra.* SIAM, 1997.

[3] Anne Greenbaum. *Iterative Methods for Solving Linear Systems*, vol. 17 of *Frontiers in Applied Mathematics.* SIAM, 1997.

[4] Arieh Iserles. *A First Course in the Numerical Analysis of Differential Equations.* Cambridge Texts in Applied Mathematics. Cambridge University Press, second edition, 2009.

[5] Samuel Khuvis and Matthias K. Gobbert. Parallel performance studies for an elliptic test problem on the cluster maya. Technical Report HPCF–2014–6, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2014.

[6] Sarah Swatski. Solving the Poisson equation using several numerical methods, 2014. Department of Mathematics and Statistics, University of Maryland, Baltimore County.

[7] David S. Watkins. *Fundamentals of Matrix Computations.* Wiley, third edition, 2010.

[8] Shiming Yang and Matthias K. Gobbert. The optimal relaxation parameter for the SOR method applied to the Poisson equation in any space dimensions. *Appl. Math. Lett.*, vol. 22, pp. 325–331, 2009.