

ASSESSING RANDOM ITERATION

A Project

The Department of Mathematics
The University of Southern Mississippi

In Supplement
to the Requirements for the Degree
Bachelor of Science
In the Department of Mathematics

by
Rochelle Jenkins
February, 2002

Approved by:

Joseph Kolibal
Department of Mathematics

Wallace Pye, Chair
Department of Mathematics

Assessing Random Iteration: A Study Examining Parallel Iterative Solvers

1 Introduction

The development of iterative methods is driven by the desire to improve, or accelerate the rate of convergence of these methods. Clearly, in attempting to construct a sequence of iterations, $\mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \dots, \mathbf{u}^{(n)}$ which converge to \mathbf{u} , it is important to consider methods which yield the smallest error, $\mathbf{e}^{(k)} = \|\mathbf{u} - \mathbf{u}^{(n)}\|$ in the least number of iterations, n . Historically, this process started with the realization that the Jacobi iterative scheme could be improved by reordering the iteration so as to use the latest value of the iterate, resulting in the Gauss-Seidel method. This simple change greatly improves the rate of convergence. Subsequently, it was realized that both of these approach represent simplifications of a more sophisticated iterative algorithm, known as SOR in which an optimal choice of a relaxation parameter ω significantly accelerates the convergence over Gauss-Seidel [10]. In the last 50 years of numerical development, several new approaches have been introduced, included the conjugate gradient method which can ideally obtain the solution of a linear system in about \sqrt{n} iterations, where n is the order of the matrix.

This study is about examining the effects of iteration order on relaxation methods, and is aimed at assessing the impact of parallelizing elementary methods such as Jacobi and Gauss-Seidel. The purpose is to assess the effects of randomly selecting the iteration order (as would be the case in a massively parallel, asynchronous computation) on the convergence rate of the algorithm. Understanding the effects of iteration order on iterative matrix solvers was the first step taken years ago toward understanding how to improve the convergence rate of the Jacobi algorithms and is the motivation for using Gauss-Seidel iteration over Jacobi iteration.

With the advent of parallel computers the need to consider possible re-orderings arise naturally in the context of parallelism in which the updates at each node are done, ideally independently, so as to allow each processor to proceed independently of the others. This is the case, for example, where a single processor is responsible for updating the values

at a particular node based on its nearest neighbors, but where the update is independent of the need to synchronize the update with any processors which do not share a common computational stencil. This approach is in contrast with the requirement imposed by many sequential algorithms in which there is an ordering of the spatial updates as the iteration proceeds from one iteration level to the next. Clearly, changing the order will not improve the convergence rate and it may make it worse when the algorithm is applied on a sequential processor, however on a massively parallel architecture, if n processors can be assigned to n nodes easily the speedup due to parallelism, may offset the losses due to the asynchrony of the algorithm. This issue will be examined for two simple, yet foundational methods, Jacobi, and Gauss-Seidel, and the rates of convergence of the asynchronous algorithm will be compared with those for the standard, sequential method for solving some elementary matrix inversion problems which arise in the numerical solution of partial differential equations. The interest is in examining the issues which arise from these re-orderings, and to more fully understand the implications for parallel computation.

A simplified problem involving the solution of Poisson's equation is used to study the convergence properties of Jacobi and Gauss-Seidel iteration. The problem solved is

$$-\Delta u(x) = f(x) \text{ in } \Omega, \quad \text{and } u(x) = g(x) \text{ on } \partial\Omega \quad (1)$$

where Ω is the domain, which in this case is the interval $[a, b]$ (which was taken to be $[0, 1]$ for simplicity), and g is a known function on the boundary of the domain. The discretization of the model problem uses finite difference methods in which the second derivative is approximated using a standard central difference approximation. This gives rise to a tridiagonal matrix A which can be conveniently solved using iterative matrix solvers such as Jacobi and Gauss-Seidel iteration. For simplicity in these studies, the source term g was taken typically to be a constant, i.e., $g(x) = 1$, or linear in x , i.e., $g(x) = x$, and homogeneous Dirichlet boundary conditions were applied to the problem, i.e., $u(0) = 0$, and $u(1) = 0$. While these constitute extremely simple models, they do allow for an investigation of the properties of Jacobi and Gauss-Seidel iteration, as the iteration order is changed.

Consequently the finite difference approximation used to solve (1) consists of the following matrix equation

$$-\frac{(u_{j+1} - 2u_j + u_{j-1}))}{h^2} = f_j; \quad (2)$$

This discretization is explicitly given by

$$\begin{pmatrix} 2 & -1 & 0 & & & \dots & 0 \\ -1 & 2 & -1 & & & & \vdots \\ 0 & -1 & 2 & -1 & & & \\ & & & & -1 & 2 & -1 & 0 \\ \vdots & & & & & -1 & 2 & -1 \\ 0 & \dots & & & 0 & -1 & 2 \end{pmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} h^2 f_1 + u_a \\ h^2 f_2 \\ \vdots \\ h^2 f_{n-2} \\ h^2 f_{n-1} + u_b \end{bmatrix}. \quad (3)$$

Note that nodes u_0 and u_n are not part of the system of equations since their values are set by the boundary conditions, and are moved to the right side of the matrix equation. These are denoted by u_a and u_b to shown that they are set by the boundary conditions..

2 Iterative methods for solving equations

The history of solving problems which require the solution of linear systems of equations is quite long and well developed. Many practical problems which require the solution of systems involving large sparse matrices arise as a consequence of modeling the numerical solution of partial differential equations in which the discretized problem requires that a matrix equation be solved. These equations can be written concisely for many problems in the form

$$A\mathbf{u} = \mathbf{f}, \quad (4)$$

where A is the matrix of coefficients, \mathbf{u} is the unknown solution vector for which the solution is sought, and f is some known vector, typically denoted the source or forcing term.

Solving the linear system in (4) using direct methods i.e., by algebraically manipulating the coefficients associated with the dependent variables in the system of equations so as to obtain the vector \mathbf{u} is effective, but numerically costly except when the matrix is not too large. The direct solution process using Gaussian elimination involves on the order of n^3 operations, and even using techniques such as LU decomposition, only reduces the order of the operations by a factor of three [3].

For many problems involving the solution of partial differential equations in which the solution is discretized using finite difference or finite element methods, the matrix A is quite large, and sparse, i.e., where most of the entries in the matrix are zero. The solution of large linear systems, especially large sparse systems, can be solved quite efficiently using iterative methods. Direct methods have the advantage that the matrix can be solved

in a finite number of steps, while iterative methods compute a sequence of approximations $\mathbf{u}^{(0)}, \mathbf{u}^{(1)}, \dots, \mathbf{u}^{(n)}$ which converge to the vector \mathbf{u} in (4). These indirect methods allow the system of equations to be solved the matrix equation is solved without ever having to explicitly invert the matrix in (4), i.e., there is no need to formally construct A^{-1} so as to provide a solution of the matrix equation. While it would seem that direct methods would always be used, in fact iterative methods tend to be preferred computationally. Iterative methods often have many advantages over direct methods in terms of speed and the demand on computer memory, and iterative methods for solving large sparse systems been developed extensively over the last several years.

Solving an equation iteratively can be demonstrated by finding the roots of the real valued function $f : \mathbb{R} \rightarrow \mathbb{R}$. Suppose that to find the roots of f , i.e., solving $f(x) = 0$, it is possible to construct a function $g : \mathbb{R} \rightarrow \mathbb{R}$, such that $g(x) = x$ in which the roots of f correspond the same value as the fixed point of g . For example, if $f(x) = x^2 - x - 1$, it is possible to construct g such that the roots of f , i.e., when $x^2 - x - 1 = 0$ correspond to

$$x = x^2 - 1 = g(x) \tag{5}$$

$$x = \sqrt{x^2 + 1} = g(x) \tag{6}$$

$$x = (x + 1)/x = g(x) \tag{7}$$

in which the function $g(x)$ is defined by expressing f so as to place x on the left of the equation, and $g(x)$ on the right. Then in each of these cases $x = g(x)$ and the value at which this occurs is the same as the value at which $g(x) = x$, i.e., at which g has a fixed point.

To approximate this fixed point, x , choose an initial approximation x_0 , and generate a sequence $\{x_n\}_{n=0}^{\infty}$ by letting $x_n = g(x_{n+1})$ for each $n \geq 1$. If the mapping g is surjective and differentiable on its domain, and if $|g'(x)| < 1$, then it can be shown that the sequence converges to a unique fixed point x , i.e.,

$$g(x) = g(\lim_{n \rightarrow \infty} x_n) = \lim_{n \rightarrow \infty} g(x_n) = \lim_{n \rightarrow \infty} x_{n+1} = x.$$

This technique for finding the roots of an equation is called fixed-point iteration.

The same idea may also be applied to functions $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$, specifically those which involve solving linear systems as in (4). In attempting to solve this linear system iteratively, it is no longer practical to attempt so construct a fixed point iteration for each scalar equation, instead it is more productive (and more direct) to construct the fixed point iteration using the matrix equation itself.

Begin by replacing the original matrix A with a simpler, invertible matrix M . The idea is to choose M so that it is related to A , and yet it is easier to invert than A . By subtracting M from A and moving it to the other side, it follows that

$$M\mathbf{u} = (M - A)\mathbf{u} + \mathbf{f}. \quad (8)$$

To solve this new matrix, make an initial guess \mathbf{u}_0 and then at each step the current value \mathbf{u}_k leads to a next approximation \mathbf{u}_{k+1} . This corresponds to the explicit recursion given by

$$M\mathbf{u}_{k+1} = (M - A)\mathbf{u}_k + \mathbf{f}. \quad (9)$$

Since \mathbf{u}_k is known on the right side of (9), and since M was chosen to be easily invertible, the system can be solved readily for \mathbf{u}_{k+1} .

2.1 Jacobi iteration

A simple, effective, if not necessarily efficient choice for M is the diagonal part of A . If M is invertible, then all of the a_{ii} are non-zero, and this yields the Jacobi Iterative method. One iteration of this method corresponds to solving for every variable once, and so the Jacobi iteration involved in going from the k -th step to the $(k + 1)$ -st step can be written out as,

$$D\mathbf{u}^{(k+1)} = (D - A)\mathbf{u}^{(k)} + \mathbf{f}, \quad (10)$$

Interpreting the matrix equation as a system of equations, yields an explicit formula for the Jacobi iteration procedure:

$$\begin{aligned} a_{11}u_1^{(k+1)} &= (-a_{12}u_2 - a_{13}u_3 - \dots - a_{1n}u_n)^{(k)} + f_1 \\ a_{22}u_2^{(k+1)} &= (-a_{21}u_1 - a_{23}u_3 - \dots - a_{2n}u_n)^{(k)} + f_2 \\ &\vdots \\ a_{jj}u_j^{(k+1)} &= (-a_{j1}u_1 - a_{j2}u_2 - \dots - a_{j,j-1}u_{j-1} - a_{j,j+1}u_{j+1} - \dots - a_{jn}u_n)^{(k)} + f_j \\ &\vdots \\ a_{nn}u_n^{(k+1)} &= (-a_{n1}u_1 - a_{n2}u_2 - \dots - a_{n,n-1}u_{n-1})^{(k)} + f_n. \end{aligned}$$

and from this it is obvious that two storage arrays are required to compute the solution, and that the update can be done in any order when moving through the index of scalar equations. Thus, this algorithm, if not necessarily optimal, is easily parallelizable in that the update of the $k + 1$ -st iterate is obtained in any order from the value of the corresponding k -th iterate.

When A is sparse, the terms on the right are mostly zeros, and the computation of the update $\mathbf{u}^{(k+1)}$ requires only a few combinations of the a_{ij} and u_j . For example, for the solution of Poisson's equation in one dimension, the update of each nodal value u_j would involve only its two nearest neighbors u_{j-1} and u_{j+1} .

2.2 Gauss-Seidel iteration

The disadvantage with the Jacobi method is that it is using the old values of \mathbf{u}^k to obtain the new solution vector $\mathbf{u}^{(k+1)}$, and thus the computation of the $u_j^{(k+1)}$ iterate does not make use of the newest information about the value of $u_{j-1}^{(k+1)}$. A better technique known as the Gauss-Seidel iterative method allows half as much of the storage, unlike Jacobi which requires all the components of $\mathbf{u}^{(k)}$ be kept until the calculation of $\mathbf{u}^{(k+1)}$ is complete, and as will be seen, converges faster.

The Gauss-Seidel method is like the Jacobi method, except it uses updated values as soon as they are available. This method starts using each component of the new $\mathbf{u}^{(k+1)}$ as soon as it is computed, and the $u_j^{(k+1)}$ overwrites the $u_j^{(k)}$ component, and the old vector $\mathbf{u}^{(k)}$ is destroyed as fast as $\mathbf{u}^{(k+1)}$ is created. The Gauss-Seidel iteration can be written as,

$$\begin{aligned}
 a_{11}u_1^{k+1} &= (-a_{12}u_2 - a_{13}u_3 - \dots - a_{1n}u_n)^k + f_1 \\
 a_{22}u_2^{k+1} &= (-a_{21}u_1)^{k+1} + (-a_{23}u_3 - \dots - a_{2n}u_n)^k + f_2 \\
 &\vdots \\
 a_{jj}u_j^{k+1} &= (-a_{j1}u_1 - a_{j2}u_2 - \dots - a_{j,j-1}u_{j-1})^{k+1} + (-a_{j,j+1}u_{j+1} - \dots - a_{j,n}u_n)^k + f_j \\
 &\vdots \\
 a_{nn}u_n^{k+1} &= (-a_{n1}u_1 - a_{n2}u_2 - \dots - a_{n,n-1}u_{n-1})^{k+1} + f_n.
 \end{aligned}$$

Written as a matrix equation, the Gauss-Seidel iteration can be concisely expressed as

$$(D - L)\mathbf{u}^{(k+1)} = U\mathbf{u}^{(k)} + \mathbf{f}. \quad (11)$$

where $M = A - D - L - U$, and D , L , and U are the diagonal, lower triangular, and upper triangular matrices associated with A . In contrast to Jacobi iteration, there is only one storage array which is kept when iterating numerically, and the order of the iteration is significant. Thus the solution classically proceeds by iterating the components from $j = 1, 2, \dots, n$, in increasing order. When solving boundary value problems, such as Poisson's equation, it is also advantageous to alternate the sequencing from of the components from $j = 0, 2, \dots, n$ on one sweep, followed by iterating the components from $j = n, n -$

$1, \dots, 0$ on the next sweep. This allows both boundary conditions to contribute equally to the development of the solution in the interior.

As the convergence rate of the Jacobi method was improved by going to Gauss-Seidel, the Gauss-Seidel method can be improved by a technique known as successive over-relaxation, or the SOR method. The SOR method can be derived from the Gauss-Seidel method by introducing a parameter, ω . The choice of ω depends on the individual problem, but it is greater than 0, and never exceeds 2 (otherwise the technique will not converge). Written in matrix notation, SOR consists of

$$(D - \omega L)\mathbf{u}^{(k+1)} = [(1 - \omega)D + \omega U]\mathbf{u}^{(k)} + \omega \mathbf{f}, \quad (12)$$

and can be seen to be a combination of the Jacobi and Gauss-Seidel methods, with the Jacobi iteration weighted by $1 - \omega$, and the Gauss-Seidel method weighted by ω . For an optimal choice of ω it is well known that SOR yields convergence results that are substantially improved over Gauss-Seidel, and although SOR does not involve changing the order the iterations, from a historical perspective it brings another idea into play which will be important in this analysis, namely its development lead to an understanding of why SOR improves the convergence of Gauss-Seidel: The rate of convergence of all of these methods is determined by the eigenvalues of the iteration matrix M , and understanding how reordering, or randomizing the iteration order affects the eigenvalues.

3 Assessing the convergence of iterative methods

The convergence of iterative methods must depend on M and A . Notice this by subtracting (9) from (8) an error equation, $\mathbf{e}^{(k)} = \mathbf{u} - \mathbf{u}^{(k)}$ arises. The error equation then becomes

$$M\mathbf{e}^{(k+1)} = (M - A)\mathbf{e}^{(k)}. \quad (13)$$

This is what is known as the one-step difference equation, from which the vector \mathbf{f} has disappeared. By multiplying the matrix equation by M^{-1} , the equations then becomes

$$\mathbf{e}^{(k+1)} = M^{-1}(M - A)\mathbf{e}^{(k)} = B\mathbf{e}^{(k)}. \quad (14)$$

Notice that in (14) at every step of the iteration the error vector, $\mathbf{e}^{(k)}$ is premultiplied by the matrix $B = I - M^{-1}A$. After k iterations there have been k multiplications by B , and the current error is related to the original error by $\mathbf{e}^{(k)} = B^k \mathbf{e}^{(0)}$. The powers of B^k approach zero if and only if every eigenvalue of B satisfies $|\lambda_i| < 1$, and rate of convergence is determined by the largest of the $|\lambda_i|$, which known as the spectral radius of B .

It is also important to recognize that the error $B^k \mathbf{e}^{(0)}$ splits into n different terms, each one proceeding independently according to its eigenvalue:

$$B^k \mathbf{e}^{(0)} = c_1 \lambda_1^k \mathbf{x}_1 + \cdots + c_n \lambda_n^k \mathbf{x}_n \quad (15)$$

In (15) the vectors $\mathbf{x}_1, \dots, \mathbf{x}_n$ are the eigenvectors and the assumption is that B contains a complete set¹. Every multiplication by B , at every iteration step, will multiply each eigenvector x_i by one more factor λ_i . If (15) is correct at the starting point $k = 0$, then (15) is correct. The constants c_i come from the initial error by $\mathbf{e}^{(0)} = c_1 \mathbf{x}_1 + \cdots + c_n \mathbf{x}_n$. Equation (15) shows that the error $B^k \mathbf{e}^{(0)}$ goes to zero when every eigenvalue satisfies $|\lambda_i| > 1$. Notice that the largest eigenvalue, say $|\lambda_i|$, is dominant in determining the behavior of the sum in (15), so that it governs the rate at which the errors approach zero.

4 Computational Results

A computer program implementing the Jacobi and Gauss-Seidel algorithms (random and non-random) was constructed using C++ in order to examine numerical and computational issues associated with random iteration. Referring to the Jacobi and Gauss-Seidel iterations discussed earlier in this section, the inclusion of randomness into the iteration process is implemented on the j th iterate using:

$$a_{jj} u_j^{(k+1)} = (-a_{j1} u_1 - a_{j2} u_2 - \cdots - a_{j,j-1} u_{j-1} - a_{j,j+1} u_{j+1} - \cdots - a_{jn} u_n)^{(k)} + f_j \quad (16)$$

Rather than update j sequentially, j is indexed based on the integers in the interval $[0, n]$. The nodes are then randomly selected, (except for nodes $j = 0$ and $j = n$ which are held fixed because of the boundary conditions). The same approach is also applied to the Gauss-Seidel iterative method in order to achieve a random ordering of the iterates, i.e.,

$$a_{jj} u_j^{(k+1)} = (-a_{j1} u_1 - a_{j2} u_2 - \cdots - a_{j,j-1} u_{j-1})^{(k+1)} + (-a_{j,j+1} u_{j+1} - \cdots - a_{jn} u_n)^{(k)} + f_j \quad (17)$$

where the values of j are chosen at random. Note that randomly selecting the indices simulates the effect of allowing the nodes to be updated in an unsynchronized manner in a parallel computing environment.

The program itself was tested to determine whether the numerical solutions were consistent with those expected analytically. Since both problems examined, $-u_{xx} = 1$,

¹In the event that it does not, the analysis can be carried out using Jordan form, and the generalized eigenvectors, that is, it becomes more complicated to write everything down, but the analytical consequences remain the same.

and $-u_{xx} = x$ have analytical solutions, it is possible to validate the accuracy of iterative solvers directly. The error in the solution at each iteration k can be defined as the difference between the analytical solution, and the numerical solution, i.e., $e_j = |U^{(k)}(x_j) - u(x_j)|$ where $U^{(k)}(x_j) = U_j^{(k)}$ is the error at each node. An approximation to the average error across all of the interior grid points is then given by

$$e = \frac{1}{n} \sum_{j=1}^n |U^{(k)}(x_j) - u(x_j)|. \quad (18)$$

Several studies were done to assess the performance of the random algorithms. In all of these studies it was observed that the rate of convergence of the random iteration algorithm was lower than for the non-random algorithm. This is to be expected since for the random case, some nodes will not be updated sufficiently frequently, while others may be updated too often. This means that information from the boundaries cannot move across the domain as rapidly as if the iteration was being done using a non-random model. In the context of the eigenvalues of the iteration matrix M , whether the convergence examined Jacobi, or Gauss-Seidel iteration, the effect of applying the updates randomly is to decrease the effective eigenvalue of the system (in effect, updating the same node does not change the solution on iteration k , and hence it is reasonable to expect that this means that the eigenvalues of the iteration matrix are made larger).

In the numerical studies that were done, there are some noteworthy observations which should be made before going into a more detailed analysis of the convergence of the algorithms: Clearly, while the convergence rate was decreased for the random iterations, in all cases the problems converged, Examining the rates of convergence for the problem solved on a 1000 nodes, shown in Figs. 1–4, the scatter diagram showing the error plotted against iteration shows a relatively narrow banding of the errors around the mean. This is encouraging since it means that the random iteration process, though slower does not lead to non-convergent problems, at least not in these elementary cases which were examined.

The failure of the randomly selected update mechanism to converge was a possibility which was considered in the development of these algorithms, and in part the convergence of either random iterations schemes is somewhat a consequence of the rather simple problem being solved, nevertheless even for problems which for which convergence is more difficult to achieve, the random iteration approach can be prevented by from introducing an additional level of instability if all of the independent processors are tied together, or linked, i.e., after a certain number of random iterations, the process is followed by a non-random iteration so as to synchronize all of the nodes.

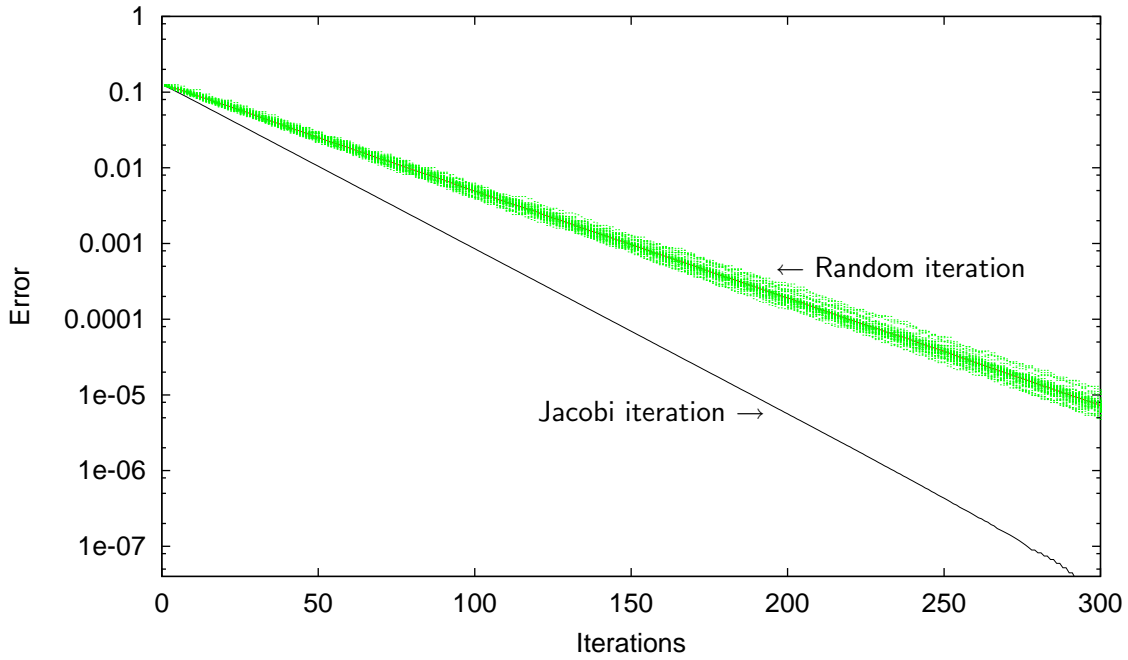


Figure 1: Comparison of the convergence of Jacobi iteration solving for $-u_{xx} = 1$ on a grid of 1000 points. The graph show the error as a function of the iteration number. The graphs show three lines. The bottom one is for standard Jacobi, and the top solid line shows the average error taken over 50 runs using random Jacobi iteration. The points around this line show the individual errors for each of the 50 runs.

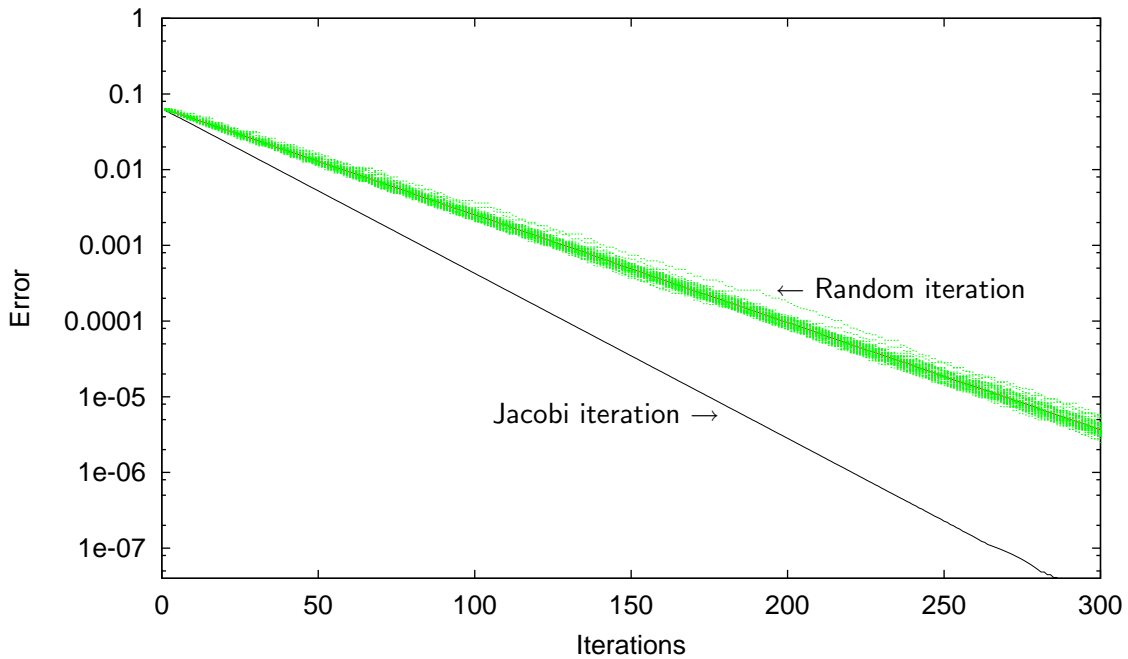


Figure 2: Comparison of the convergence of Jacobi iteration solving for $-u_{xx} = x$ on a grid of 1000 points. The graph show the error as a function of the iteration number. The graphs show three lines. The bottom one is for standard Jacobi, and the top solid line shows the average error taken over 50 runs using random Jacobi iteration. The points around this line show the individual errors for each of the 50 runs.

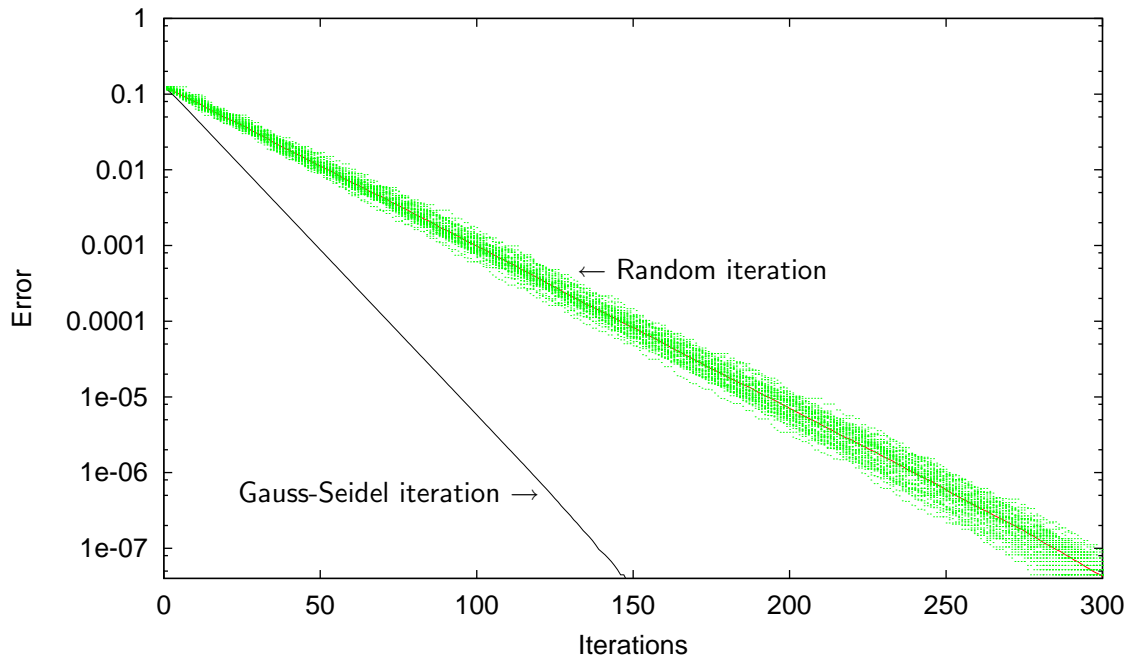


Figure 3: Comparison of the convergence of Gauss-Seidel iteration solving for $-u_{xx} = 1$ on a grid of 1000 points. The graph shows the error as a function of the iteration number. The graph shows three lines. The bottom one is for standard Gauss-Seidel and the top solid line shows the average error taken over 50 runs using random Jacobi iteration. The points around this line show the individual errors for each of the 50 runs.

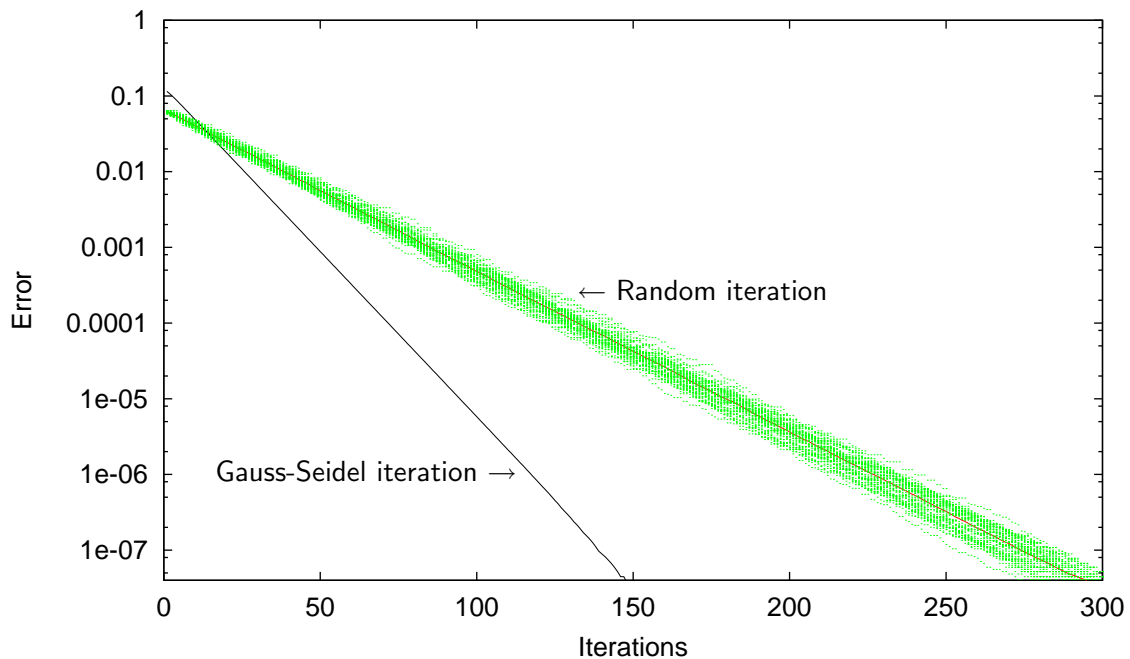


Figure 4: Comparison of the convergence of Gauss-Seidel iteration solving for $-u_{xx} = x$ on a grid of 1000 points. The graph shows the error as a function of the iteration number. The graph shows three lines. The bottom one is for standard Gauss-Seidel and the top solid line shows the average error taken over 50 runs using random Jacobi iteration. The points around this line show the individual errors for each of the 50 runs.

The computer models substantiate that it seems reasonable to construct convergent parallel algorithms using the standard iterative matrix solvers (such as Jacobi, Gauss-Seidel, and for that matter the substantially more efficient SOR method). To understand the reason for the observed convergence rates for the random iteration, a more thorough analysis of the eigenvalues of the iteration matrix, or more specifically the eigenvalues of the error iteration matrix B given in (14) must be evaluated when the updates are done randomly.

5 Analysis of the random iteration convergence rates

There are two parts to computing the convergence rate of the random iteration procedures outlined at the beginning of Sec. 4: Computing the eigenvalues of B , and computing the effective eigenvalue of B when a random iteration is used. The effective eigenvalue refers to the modification of the eigenvalues of the system which results as a consequence of nodes being updated which have already been updated, or failing to be updated as the iteration proceeds.

To simplify this computation, the results are examined for three small cases: a 2×2 , 3×3 , and 4×4 iteration matrix B corresponding to problems on 4, 5, and 6 grid points (since the boundary nodes do not enter in the matrix). For the Jacobi 2×2 matrix case, the matrix A is written as follows:

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}. \quad (19)$$

The matrix B in (14) is next constructed from A and M . For the Jacobi iteration, the matrix M is just the diagonal portion of A and is written as follows:

$$M = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \quad (20)$$

The inverse matrix of M is this trivially found to be

$$M^{-1} = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix}, \quad (21)$$

and, consequently the error iteration matrix is given by

$$B = \begin{pmatrix} 0 & 1/2 \\ 1/2 & 0 \end{pmatrix} \quad (22)$$

It is clear that the eigenvalues of the matrix B for the 2×2 Jacobi matrix B are $\lambda_1 = 1/2$ and $\lambda_2 = 1/2$.

The other two cases are developed in the same fashion and the results are

3×3 Case

$$A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}, \quad M = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}, \quad \text{and } B = \begin{pmatrix} 0 & 1/2 & 0 \\ 1/2 & 0 & 1/2 \\ 0 & 1/2 & 0 \end{pmatrix}, \quad (23)$$

with eigenvalues for the Jacobi 3×3 error iteration matrix B given by $\lambda_1 = 0$, $\lambda_2 = \sqrt{2}/2$, and $\lambda_3 = -\sqrt{2}/2$.

4×4 Case

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}, \quad M = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}, \quad \text{and } B = \begin{pmatrix} 0 & 1/2 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1/2 & 0 \end{pmatrix}, \quad (24)$$

with the eigenvalues for the Jacobi 4×4 error iteration matrix given by $\lambda_1 = -1/4 + \sqrt{5}/4$, $\lambda_2 = -1/4 - \sqrt{5}/4$, $\lambda_3 = 1/4 + \sqrt{5}/4$, and $\lambda_4 = 1/4 - \sqrt{5}/4$.

For the Gauss-Seidel 2×2 , 3×3 , and 4×4 matrix cases, the matrix A is written the same as in the Jacobi cases. Each case for the other matrices, M and B , are as follows:

2×2 Case

$$M = \begin{pmatrix} 2 & 0 \\ -1 & 2 \end{pmatrix}, \quad \text{and } B = \begin{pmatrix} 0 & 1/2 \\ 0 & 1/4 \end{pmatrix}, \quad (25)$$

with the eigenvalues of B for the 2×2 Gauss-Seidel case given by are $\lambda_1 = 0$ and $\lambda_2 = 1/4$.

3×3 Case

$$M = \begin{pmatrix} 2 & 0 & 0 \\ -1 & 2 & 0 \\ 0 & -1 & 2 \end{pmatrix}, \quad \text{and } B = \begin{pmatrix} 0 & 1/2 & 0 \\ 0 & 1/4 & 1/2 \\ 0 & 1/8 & 1/4 \end{pmatrix}, \quad (26)$$

with the eigenvalues for the 3×3 Gauss-Seidel matrix B given by $\lambda_1 = 0$, $\lambda_2 = 0$, and $\lambda_3 = 1/2$.

4×4 Case

$$M = \begin{pmatrix} 2 & 0 & 0 & 0 \\ -1 & 2 & 0 & 0 \\ 0 & -1 & 2 & 0 \\ 0 & 0 & -1 & 2 \end{pmatrix} \quad \text{and } B = \begin{pmatrix} 0 & 1/2 & 0 & 0 \\ 0 & 1/4 & 1/2 & 0 \\ 0 & 1/8 & 1/4 & 1/2 \\ 0 & 1/16 & 1/8 & 1/4 \end{pmatrix}, \quad (27)$$

with the eigenvalues for the 4×4 Gauss-Seidel matrix B given by $\lambda_1 = 0$, $\lambda_2 = 0$, $\lambda_3 = 3/8 + \sqrt{5}/8$, and $\lambda_4 = 3/8 - \sqrt{5}/8$.

The eigenvalues for each case for both Jacobi and Gauss-Seidel iterative methods for the error iteration matrix B are summarized in a Table 1.

Size of B	2×2	3×3	4×4
Jacobi	$1/2, -1/2$	$0, \sqrt{2}/2, -\sqrt{2}/2$	$-1/4 + \sqrt{5}/4, -1/4 - \sqrt{5}/4$
Gauss-Seidel	$0, 1/4$	$0, 0, 1/2$	$1/4 + \sqrt{5}/4, 1/4 - \sqrt{5}/4$ $0, 0, 3/8 + \sqrt{5}/8, 3/8 - \sqrt{5}/8$

Table 1: Eigenvalues for the matrix B .

5.1 The effects of randomness on B

Consider the case of Jacobi iteration on a grid of four uniformly spaced points in which there are two interior nodes. This is the simplest possible geometry in which random iteration can be used (as the case of one interior point is degenerate, in that the sole interior point will always be chosen since it is the only point). In the usual Jacobi update procedure node x_1 is updated, i.e., $u_1^{(k)}$ is obtained, followed by node x_2 , i.e., $u_2^{(k)}$ is obtained. The procedure is repeated in that at each iteration k , the nodes 1, then 2 are updated. The update of the nodes can be expressed a sequence of binary numbers, 0101010101010... in which each pair represents an update of the variables u_1, u_2 such that the k th pair of numbers represents the k th update of the variables $u_1^{(k)}, u_2^{(k)}$. Note that as k becomes large the sequence 10101010101... obtained by shifting the first sequence to the right by 1 is equivalent (to within one iteration), and so the only factor determining the progress of the iteration for large k is the number of pairs 01 which occurring in any order.

In random Jacobi on two interior nodes, the update can proceed, for example as the sequence 0010101101110... in which the numbers 0 and 1 are not always paired, however a complete update will only be completed when a pair of two different numbers has occurred. This is because successive occurrences of two or more repeated numbers will have no effect for Jacobi iteration since the value of that node will not change (for example, if it is node 0, then its value is determined only by the boundary node on its left, and node 1 one on its right which was not updated, consequently the value of node 0 will remain the same). Thus, the progress of random Jacobi will be determined by the frequency of the occurrence of successive pairs 0, and 1. If the frequency of occurrence of these pairs is r , then the convergence rate will be λ^{rk} in contrast to a convergence rate of λ^k for non-random Jacobi, where λ is the maximum eigenvalue of B . For the 2×2 case this means the error should decrease as $(1/2)^{rk}$.

Nodes, n	Factor, r
2	0.666613
3	0.545448
4	0.479938
5	0.437935
8	0.367951
10	0.341484
20	0.278013
50	0.222396
100	0.192798
500	0.147370
800	0.137575

Table 2: Tabulation of the frequency of occurrence (r) of strings of length n which occur in randomly generated strings taken from n objects.

The probability of finding occurrences consisting of pairs of numbers, $\{0, 1\}$ drawn from an infinite string comprised all of the digits 0 or 1 randomly selected (with equal probability for each digit), can be computed analytically to be $2/3$. The computation of this value analytically for numbers larger than two, i.e., finding the probability of the occurrence of strings of digits from 0 to n in infinitely long strings consisting of numbers drawn randomly from the set $\{0, 1, \dots, n\}$ can be computed by simulation, and the numerical results are presented in in Table 2. These simulations were done on strings comprised of over 48 million digits, in which the search for substrings was done so as to maximize the count (i.e., the string was search with an offset from 0 to $n - 1$, and the maximum number of substrings of length n was recorded). To improve the statistics of the numbers presented in Table 2, the computations were averaged over several runs. Having noted this, the importance of these numbers is not in the details of the numbers, but in the trend shown as the number of interior nodes, n increases: While the factor r , or frequency of the occurrence of the strings goes down as n increases, the rate of decrease diminishes, giving what appears to be an asymptotic rate of decrease of $r = 0.395x^{-0.157}$.

For the Jacobi iteration done on a grid consisting of 4 points, and thus comprising a 2×2 matrix to be inverted, the effects of random iteration on the reduction of the error due to iterating the error matrix B is then measurable as $(1/2)^{(2/3)^k}$. This is demonstrated in the Fig. ?? which shows the plot or the error in Jacobi iteration in the case of attempting to solve the model problem $-u_{xx} = 1$. In the figure the straight line on the lower right is the convergence curve for the non-random iteration, and in the upper right there are two

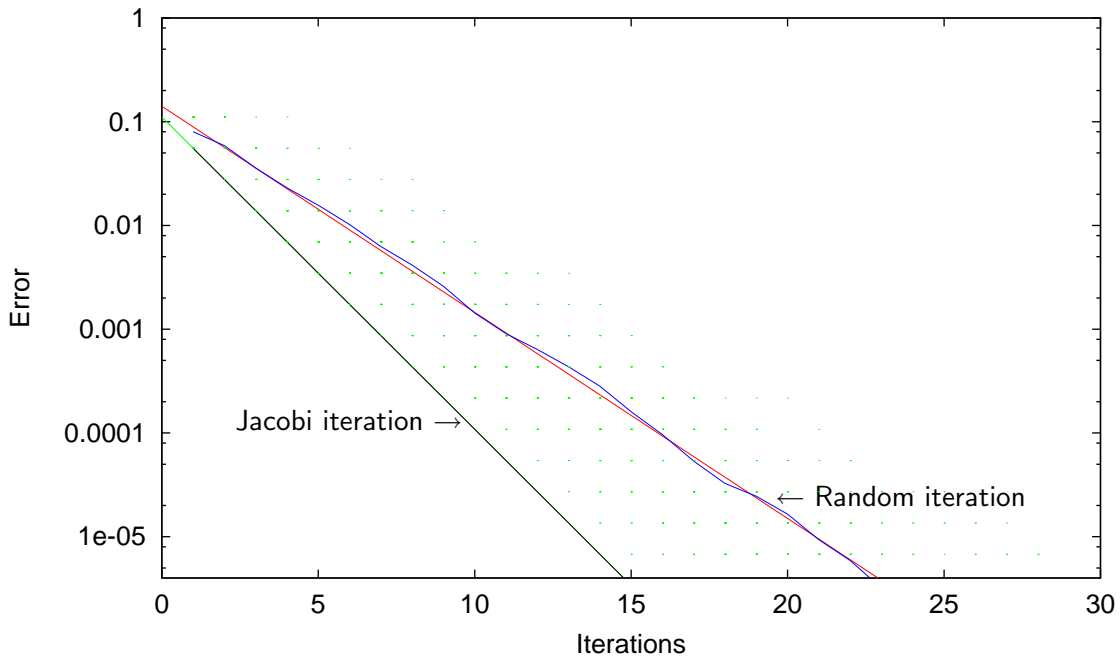


Figure 5: Comparison of random and non-random Jacobi iteration showing the analytical estimate of the reduction in the error on solving $-u_{xx} = 1$ on four nodes (i.e., inverting a 2×2 matrix). Note that the analytical result, $e_k = e_0(1/2)^{(2/3)k}$ agrees with the numerical results for the random iteration. The dots in the figure are the actual errors occurring in each of the individual computational studies.

curves show, which overlay each other quite well. The first of these is the curve showing the convergence rate of random Jacobi (averaged over 50 runs) compared to the theoretical curve based on the analytically obtained estimate the $e_k = e_0(1/2)^{(2/3)k}$. In this study the value of e_0 , the initial error, was set to the value obtained from the experimental average of the 50 random runs. On inspecting this graph, the only difference between the random iteration curve, and the analytically derived curve is that the latter is straight, while the numerically obtained average performance curve for the random Jacobi iteration has some wiggle, which is consistent with the random manner in which the iteration proceeds.

In attempting to extending these results to the next most difficult case, i.e., random Jacobi iteration using five nodes (the 3×3 matrix), the results in Table 2 do not extend directly. This is because on three digits, $\{0, 1, 2\}$ the effects of randomly updating the the nodes has the possibility of having partial updates change the value of an updated node, even when a complete string of 0's, 1's, and 2's has not yet occurred. Consider the substring $\{0, 1, 1\}$ which occurs, say on iteration k , followed by the substring $\{0, 1, 1\}$ which occurs on iteration $k + 1$. Because node 0 and 1 are updated on iteration k , on

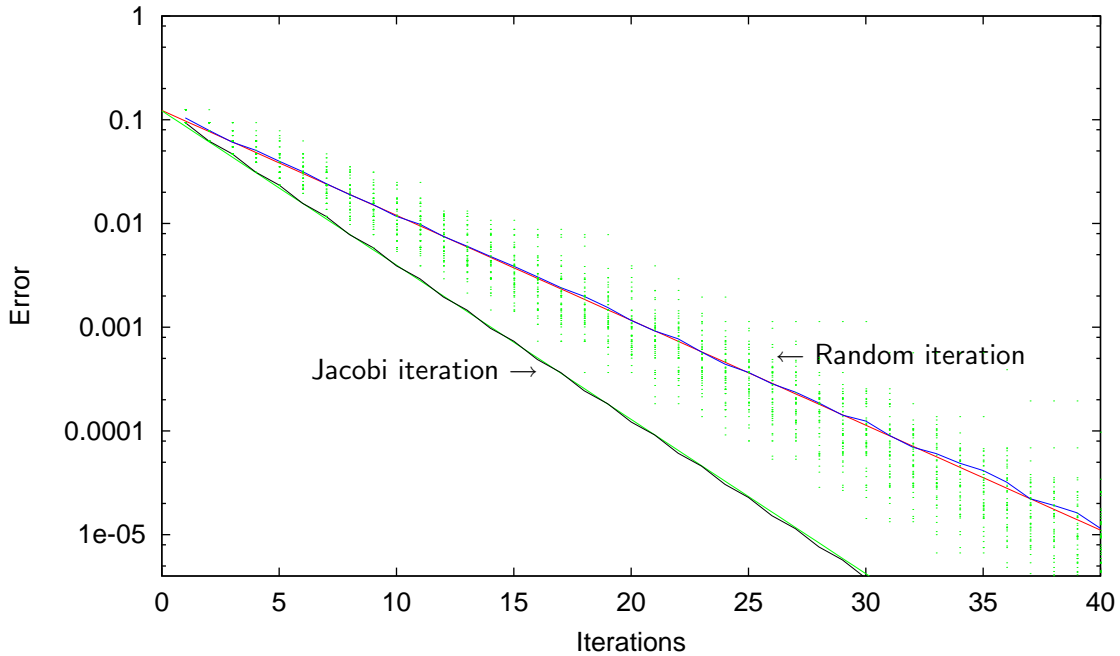


Figure 6: Comparison of random and non-random Jacobi iteration showing the analytical estimate of the reduction in the error on solving $-u_{xx} = 1$ on five nodes (i.e., inverting a 3×3 matrix). Note that the analytical result, $e_k = e_0(1/2)^{(2/3)k}$ agrees with the numerical results for the random iteration. The dots in the figure are the actual errors occurring each of the individual computational studies.

iteration $k + 1$, the update of node 0 does change in values since it depends on the left boundary node (which does not change), and on node 1, which has changed on iteration k . Thus partial updates can have an effect, unlike in the much simplified 2×2 case, and clearly this result extends to all larger matrices which are being inverted. Nevertheless, the results in Table 2 provide a lower bound on the effects of random iteration, i.e., using these results will provide an estimate of the degradation of the performance of random Jacobi iteration which is more pessimistic than would be observed.

As shown in Fig. 6, this is indeed what is observed. For example in the 3×3 case, the maximum eigenvalue is $\sqrt{2}/2 \approx 0.71$ and the rate of reduction of the error is seen to proceed as $e_k = e_0 0.71^{0.68k}$, where the reduction factor is 0.68, and not 0.54 as would be predicted from Table 2. The value 0.68 was obtained by experimentally fitting the curve of the average decrease in the error which occurs during the random iteration process.

In attempting to extend these results to Gauss-Seidel iteration, the complexity of the combinatorial problem increase substantially, however before this attempt is pursued, it is important to examine the early results presented for the convergence of the random it-

eration results in Fig. 3 and Fig. 4, and contrast these with the results obtained in in Fig. 1 and Fig. 2. Although the Gauss-Seidel iteration is significantly faster than Jacobi, the result in the random iteration case is quite the opposite: the degradation in the performance of Gauss-Seidel appears to be much greater than the performance of Jacobi when randomness is introduced. The implications for parallelizing these schemes is important when the amount of degradation in the Gauss-Seidel scheme is examined. From Fig. 3 or Fig. 4 it is obvious that the performance is more than 1000 times worse for the 1000 grid problem being solved when the error is computed down to machine precision (about 10^{-7} to 10^{-8}), and thus the advantage of having 1000 parallel processors, each dedicated to a node, is entirely negated by the the dramatic drop in performance of random Gauss-Seidel. In contrast for random Jacobi, while severe, seems to be only half as large as for Gauss-Seidel.

6 Conclusion

Clearly, neither algorithm, as presented, is a suitable candidate for massively parallel architectures, however what is successful is the mathematical analysis which has shown that the drop off in the performance of Jacobi and Gauss-Seidel iteration can be understood in the context of the modification of the error iteration matrix B , and its eigenvalues. Most importantly, it can be seen that the loss in performance does not scale well with the number of nodes n , and thus results in a substantial loss in efficiency which effectively negates any benefits of parallelizing the basis Jacobi or Gauss-Seidel algorithms using an entirely random approach to the update procedure.

Having done the analysis, though, it is intriguing to speculate, in regard to future work, about the possibilities of implementing the basic random update algorithm in a manner in which the computations are synchronized every so often, so as to improve the performance. In addition, this analysis has raised an interesting combinatorial problem regarding computing analytical estimates for the frequency of occurrence of complete, and partial substrings inside infinitely long strings of digits.

References

- [1] G. BIRKHOFF AND R. LYNCH, *Numerical Solution of Elliptic Problems*, SIAM, 1984.
- [2] W. BRIGGS, *A Multigrid Tutorial*, SIAM, 1987.
- [3] R. BURDEN AND J. FAIRES, *Numerical Analysis*, PWS Publishing Company, 1993.
- [4] P. DUCHATEAU AND D. ZACHMANN, *Schaum's Outline Series: Theory and Problems of Partial Differential Equations*, McGraw-Hill, Inc., 1986.

- [5] L. HAGEMAN AND D. YOUNG, *Applied Iterative Methods*, Academic Press, Inc., 1981.
- [6] R. HILL, *Elementary Linear Algebra*, Saunders College Publishing, 1996.
- [7] D. KINCAID AND W. CHENEY, *Numerical Analysis: Mathematics of Scientific Computing*, Brooks/Cole Publishing Company, 1991.
- [8] B. NOBLE AND J. DANIEL, *Applied Linear Algebra*, Prentiss-Hall, Inc., 1988.
- [9] A. QUARTERONI AND A. VALLI, *Numerical Approximation of Partial Differential Equations*, Springer-Verlag Berlin Heidelberg, 1984.
- [10] G. STRANG, *Introduction to Applied Mathematics*, Wellesley-Cambridge Press, 1986.