The Pennsylvania State University
Schreyer Honors College

Department of Mathematics

Application of Computational Linear Algebraic Methods to a Reliability Simulation

Seth Henry
Spring 2015

A thesis
submitted in partial fulfillment
of the requirements for baccalaureate degrees
in Mathematics and Physics
with honors in Mathematics

Reviewed and approved* by the following:

Christopher Griffin
Sr. Research Assoc. & Assoc. Professor of Mathematics
Thesis Supervisor

Victoria Sadovskaya
Associate Professor of Mathematics
Honors Adviser

*Signatures are on file in the Schreyer Honors College

Abstract

In this paper, numerical methods for the solution of a reliability modeling problem are presented by finding the steady state solution of a Markov chain. The reliability modeling problem analyzed is that of a large system made up of two smaller systems each with a varying number of subsystems. The focus of this study is on the optimal choice and formulation of algorithms for the steady-state solution of the generator matrix for the Markov chain associated with the given reliability modeling problem. In this context, iterative linear equation solution algorithms were compared. The Conjugate-Gradient method was determined to have the quickest convergence with the Gauss-Seidel method following close behind for the relevant model structures. The successive over-relaxation method was studied as well, and some optimal relaxation parameters are presented.

Contents

List of Figures

1 Introduction

Determining the reliability of a system is a problem frequently encountered in Naval engineering. Frequently, the systems being simulated in these problems are far too complicated to be solved by hand, so computational methods must be employed.

In this paper, the stochastic process of a Markov chain is presented in a straightforward manner. These are then used to construct a reliability simulation of a general reliability problem. Then, various computational methods are presented to solve for the steady state solution of a reliability simulation. Then, a reliability problem of interest in some specific engineering applications is presented. The methods listed before are then tested on the given reliability to determine the optimal choice of algorithm. We show empirically that for large systems the Conjugate Gradient method for the normal equations is the most effective. However, at smaller system sizes it is more beneficial to use the Gauss-Seidel method.

2 Literature Review

This section will survey the necessary topics for a proper understanding of this thesis. Topics covered include Markov chains, reliability theory, and numerical analysis.

2.1 Markov Chains

Markov chains are a common method for simulation of complex systems. Ross, [3], defines a **Markov chain** as a *stochastic process* with a *finite state space* that has the *Markov property*. That is, whenever the process is in state i the *probability*, P_{ij} of moving to state j is dependent only upon the current state i.e., i. This formulation of a Markov chain assumes a discrete time setting, where transitions between states are made at regular intervals. Markov chains are usually represented by a **transition matrix**, $\mathbf{P}$, where P_{ij} is the probability of transition from state i to state j. This is:

$$\mathbf{P} = \begin{pmatrix} P_{00} & P_{01} & P_{02} & \dots & P_{0N} \\ P_{10} & P_{11} & P_{12} & \dots & P_{1N} \\ P_{20} & P_{21} & P_{22} & \dots & P_{2N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ P_{N0} & P_{N1} & P_{N2} & \dots & P_{NN} \end{pmatrix} \qquad 0 \leq i, j \leq N$$
$$\sum_{j=0}^{N} P_{ij} = 1, \forall i \qquad (0 \leq P_{ij} \leq 1) \tag{1}$$

The matrix representation of a Markov chain allows for linear algebraic techniques to be applied to their analysis. A *state distribution* is a row vector, $\mathbf{S}$, where

$$\mathbf{s} = \begin{pmatrix} P_0 & P_1 & P_2 & \dots & P_N \end{pmatrix}, \quad \sum_{i=0}^{N} P_i = 1, \quad P_i \geq 0, \forall i. \tag{2}$$

Each value in the vector identifies the probability that the system is in a given state. Then the one-step probability distribution of states, $\mathbf{s}'$, immediately following distribution $\mathbf{s}$ is

$$\mathbf{s}' = \mathbf{s} \cdot \mathbf{P}. \tag{3}$$

We define $P_{ij}^{(n)}$, the probability of a transition from i to j in n steps.

From [3]:

$$P_{ij}^{(n)} := P\{X_{n+k} = j | X_k = i\}, \qquad n \geq 0, \quad 0 \leq i, j \leq N$$

A state j is said to be *accessible* from state i if there is a possibility of eventually transitioning to state j after starting in state i, $\exists n(i,j)$ s.t. $P_{ij}^{(n(i,j))} > 0$. Two states, i and j, are said to *communicate* if and only if they are both accessible from each other. This is written as $i \leftrightarrow j$. The property of communication also holds under transitivity, that is if state i communicates with state j and state j communicates with state k, then state i communicates with state k. This fact implies that the set of states has an equivalence relation as both reflexivity and symmetry are trivially true. This creates equivalence classes of states in the chain. If all states live in the same equivalence class then the Markov chain is said to be *irreducible*.

A state is called *periodic* with period d if $P_{ii}^{(n)} = 0$ when $d \nmid n$. A state is called aperiodic if $d = 1$. A state is said to be *recurrent* if starting in state i the probability of reentering state i is 1. A state is considered to be *positive recurrent* if the expected time of reentry to the state is finite. In the case of a finite state Markov chain all recurrent states are positive recurrent. Both periodicity and recurrence are class properties so if it can be shown that they hold for one element of an equivalence class then they necessarily hold for all of them. If all the states of a Markov chain are in the same equivalence class and that class is both positive recurrent and aperiodic, then the Markov chain is considered *ergodic* [3].

The probabilities $P_{ij}^{(n)}$ satisfy the *Chapman-Kolmogorov Equations*. These are [3]

$$P_{ij}^{(n+m)} = \sum_{k=0}^{N} P_{ik}^{(n)} P_{kj}^{(m)} \qquad \forall n, m \geq 0. \tag{4}$$

Writing the n step transition matrix as $\mathbf{P}^{(n)}$, allows these equations to be rewritten in matrix form as

$$\mathbf{P}^{(n+m)} = \mathbf{P}^{(n)} \cdot \mathbf{P}^{(m)}.$$

This implies that

$$\mathbf{P}^{(n)} = \mathbf{P}^{n}. \tag{5}$$

That is, one can compute the n-step transition matrix by merely raising the initial transition matrix to the n-th power.

In the case of an ergodic Markov chain, *limiting probabilities*, π_j, can be

calculated. These are defined as

$$\pi_j := \lim_{n\to\infty} P_{ij}^n, \qquad j \geq 0.$$

Note that π_j is independent of i. This means that the steady state is independent of the starting state. Then by [3],

$$\pi_j = \sum_{i=0}^{N} \pi_i P_{ij}, \qquad j \geq 0, \quad \sum_{j=0}^{N} \pi_j = 1, \quad \pi_j > 0. \tag{6}$$

This is the same as calculating the left hand eigenvector of the dominant eigenvalue, $\lambda = 1$ (for all eigenvalues other than the dominant, $|\lambda| < 1$), or solving the equation

$$\boldsymbol{\pi} = \boldsymbol{\pi} \cdot \mathbf{P}. \tag{7}$$

This $\boldsymbol{\pi}$, is often called the *steady state solution*, and denotes the probability of being in state j after the system has run for a long time. Note that these values are independent of any initial distribution of states.

Despite the generality of the formulation of a Markov chain its modeling capabilities are rather restricted due to the requirement of discretized time. A *continuous-time Markov chain* can be used to overcome this problem. In this case, instead of letting the time domain take integer values it is allowed to take any real value ≥ 0. Since there are no individual time steps to analyze the transitions of system, distributions of transition rates must be used instead of simple transition probabilities. These *continuous-time Markov chains* must also have the *Markov property*. In a continuous-time setting this is

$$P\{X(t+s) = j | X(s) = i, X(u) = x(u), 0 \leq u < s\} = P\{X(t+s) = j | X(s) = i\}$$

where $X(t)$ denotes the random variable of the state at time t, $x(t)$ denotes the actual values of the state at time t, and i and j denote possible state values.

Since the time that transitions occur is not uniform and is itself randomly distributed, another random variable, T_i, can be introduced. T_i denotes the time spent in state i before transition. The Markovian property asserts that these T_i also be memoryless. The only continuous memoryless probability distribution is the **exponential distribution**. Therefore, $T_i \sim Exp(\lambda)$. Ross,

[3], defines its *probability density function* as:

$$f(t) = \begin{cases} \lambda e^{-\lambda t}, & t \geq 0 \\ 0, & t \leq 0 \end{cases} \tag{8}$$

The expected value of the exponential distribution is:

$$\mathbf{E}[T_i] = \lambda_i^{-1}.$$

This implies that λ_i denotes the average transition rate out of i. Looking at specific transition rates from i to j, if $p_{ij}(t, t+\Delta t)$ denotes the probability of transition from state i to state j in the interval $(t, t+\Delta t)$, then the transition rate from state i to state j can be written as [4]

$$\lambda_{ij}(t) := \lim_{\Delta t \to 0} \left\{ \frac{p_{ij}(t, t+\Delta t)}{\Delta t} \right\}, \quad \text{for} \quad i \neq j.$$

Writing this in matrix form yields

$$\mathbf{Q}(t) := \lim_{\Delta t \to 0} \left\{ \frac{\mathbf{P}(t, t+\Delta t) - \mathbf{I}}{\Delta t} \right\}. \tag{9}$$

In the case of a homogeneous continuous-time Markov chain, these are simply λ_{ij} and $\mathbf{Q}$. Thus, if the average transition rates between all states are known, then a **generator matrix** can be formed to represent the *continuous-time Markov chain*. This is written as

$$\mathbf{Q} = \begin{pmatrix} -s_0 & \lambda_{01} & \lambda_{02} & \dots & \lambda_{0N} \\ \lambda_{10} & -s_1 & \lambda_{12} & \dots & \lambda_{1N} \\ \lambda_{20} & \lambda_{21} & -s_2 & \dots & \lambda_{2N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \lambda_{N0} & \lambda_{N1} & \lambda_{N2} & \dots & 0 \end{pmatrix}. \tag{10}$$

Here s_i denotes the sum of all off-diagonal values in row i, and λ_{ij} denotes the average transition rate from i to j. Note that this construction forces the row-sum of each row to be zero. This also forces the matrix to be uniformly diagonally dominant, that is that the absolute value of the element on the diagonal s_i for each row i is greater than or equal to the sum of all other elements in this row. This will prove very useful later.

Within these *continuous-time Markov chains* there are embedded discrete *Markov chains* that can be used to gain a better understanding of the behavior of the system. The transition matrix for the *embedded Markov chain*

is calculated as:

$$\mathbf{P}_E = \begin{pmatrix} 0 & \lambda_{01}/s_0 & \lambda_{02}/s_0 & \dots & \lambda_{0N}/s_0 \\ \lambda_{10}/s_1 & 0 & \lambda_{12}/s_1 & \dots & \lambda_{1N}/s_1 \\ \lambda_{20}/s_2 & \lambda_{21}/s_2 & 0 & \dots & \lambda_{2N}/s_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \lambda_{N0}/s_N & \lambda_{N1}/s_N & \lambda_{N2}/s_N & \dots & 0 \end{pmatrix}. \tag{11}$$

Where λ_{ij} and s_i are defined as before. From [4], this can be calculated in matrix terms as:

$$\mathbf{P}_E = \mathbf{I} - [diag\{\mathbf{Q}\}]^{-1}\mathbf{Q}. \tag{12}$$

Since the probability distribution of states in the continuous-time case changes with time, they must be written as a function of time. This is done so as [4]

$$\pi_i(t) = P\{X(t) = i\}.$$

The *Chapman-Kolmogorov Equations* again allow for the computation of *steady state solutions*, $\boldsymbol{\pi}$, for a *continuous-time Markov chain*. In this situation, these are [4]

$$\frac{d\boldsymbol{\pi}(t)}{dt} = \boldsymbol{\pi}(t)\mathbf{Q}(t).$$

In the case of a homogeneous continuous-time Markov chain this simplifies to

$$\frac{d\boldsymbol{\pi}(t)}{dt} = \boldsymbol{\pi}(t)\mathbf{Q}. \tag{13}$$

In the case of a steady state, the derivative must be equal to 0. Therefore, the steady state solution can be calculated by solving the homogeneous equation [4]:

$$\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}. \tag{14}$$

Calculation of *steady state solutions* for both the discrete and continuous case are simple problems for small systems, but as the size of the system increases the relevant matrices can grow to enormous sizes, making computation difficult.

2.2 Reliability Theory

Reliability theory is the statistical and probabilistic study of the lifespan of various systems. The lifespan of a system is analyzed by recursively breaking the system into multiple subsystems, each with their own lifespan properties. Before discussing exactly how this is done, the idea of reliability must be de-

fined. **Leemis**, [2], defines the **reliability** of an *item* as the *probability* that it will *adequately perform* its specified *purpose* for a specified period of time under specified *environmental conditions*. This definition is quite abstract, and thus allows for a rather broad use of reliability theory in applied mathematical situations.

Within *reliability theory*, "the *structure function* defines the system as a function of the component states" [2]. Binary variables are sufficient to describe the state of individual components. The state of component i, x_i is defined as:

$$x_i = \begin{cases} 0, & \text{if component } i \text{ has failed} \\ 1, & \text{if component } i \text{ is functioning} \end{cases} \tag{15}$$

These x_i can be written as a *system state vector*:

$$\mathbf{x} = (x_1, x_2, \ldots, x_n). \tag{16}$$

The *structure function* for this system can then be written as a function of $\mathbf{x}$:

$$\phi(\mathbf{x}) = \begin{cases} 0, & \text{if the system has failed when the state vector is } \mathbf{x} \\ 1, & \text{if the system is functioning when the state vector is } \mathbf{x} \end{cases} \tag{17}$$

The two extreme situations for *structure functions* arise when a system is either completely in *parallel* or completely in *series*. The structure function for a completely *parallel* system is as follows:

$$\phi(\mathbf{x}_p) = ||\mathbf{x}_p||_\infty = 1 - \prod_{i=1}^{n}(1 - x_i). \tag{18}$$

Alternatively, the *structure function* for a system that is completely in series is:

$$\phi(\mathbf{x}_s) = \prod_{i=1}^{n} x_i \tag{19}$$

Another possible structure for a system is a *k-out-of-n system*. From [2], a *k-out-of-n system* only functions if k or more of its n components functions. This can be written as:

$$\phi(\mathbf{x}) = \begin{cases} 0, & \text{if } \sum_{i=1}^{n} x_i < k \\ 1, & \text{if } \sum_{i=1}^{n} x_i \geq k \end{cases} \tag{20}$$

These structure functions can now be expanded into a stochastic setting to give a *reliability function* for the system. Instead of using x_i to denote the

state of a system, now X_i is used to denote the *random state* of the system. That is [2]:

$$X_i = \begin{cases} 0, & \text{if component } i \text{ has failed} \\ 1, & \text{if component } i \text{ is functioning} \end{cases} \tag{21}$$

As before, a *random state vector* can be written as

$$\mathbf{X} = (X_1, X_2, \ldots, X_n). \tag{22}$$

Since these are all random variables, they have associated probabilities. These are referred to as the *reliability* of component i, written as:

$$p_i = P[X_i = 1]. \tag{23}$$

These reliabilities can be combined into a *reliability vector*:

$$\mathbf{p} = (p_1, p_2, \ldots, p_n). \tag{24}$$

The *system reliability*, r, can be defined as:

$$r(\mathbf{p}) = P[\phi(\mathbf{X}) = 1]. \tag{25}$$

Some systems have components which are repairable. From [2], "a *repairable item* may be returned to an operating condition after failure to perform a required function by any method other than replacement of the entire item." The reliability of systems including *repairable items* does not have a static value, so a dynamic process must be used to describe its behavior. One process that is effective in describing reliability systems is the *continuous-time Markov chain*.

By finding all possible combinations of the *random state vector* one can populate the state space of a *continuous-time Markov chain*. Any two of these states that differ by the values of only variable, then the two are connected. That is:

$$\mathbf{X} \leftrightarrow \mathbf{Y}, \qquad \text{if and only if } X_i = Y_i \text{ for all } i \text{ but one.}$$

The values used in the *generator matrix* for this *continuous-time Markov chain* will be either the *mean time between failure* (MTBF), frequently denoted as λ, or the *mean time to repair* (MTTR), frequently denoted as μ, for the component. If the component has failed during this transition it will be the MTBF, while if it has been repaired it will be the MTTR. Commonly, the *generator matrix* is structured such that all values above the diagonal

are multiples of λ and all values below are multiples of μ:

$$\mathbf{Q} = \begin{pmatrix} -s_0 & \lambda_{01} & \lambda_{02} & \dots & \lambda_{0N} \\ \mu_{10} & -s_1 & \lambda_{12} & \dots & \lambda_{1N} \\ \mu_{20} & \mu_{21} & -s_2 & \dots & \lambda_{2N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mu_{N0} & \mu_{N1} & \mu_{N2} & \dots & -s_N \end{pmatrix}.$$

This convention places higher functioning states in lower state values and lower functioning states in higher state values. It is also important to note that these state spaces can frequently be written so that the *generator matrix* can take a banded structure. This should be done whenever possible as many algorithms can exploit banded matrix structure to optimize convergence.

2.3 Numerical Analysis

In both the continuous-time, and the discrete-time settings, the steady state solution of a Markov chain can be found by solving a set of linear equations. This can be done easily for small matrices. However, as the number of states in a system increases, the difficulty of solving the steady state solution also increases. Frequently, numerical methods are used to solve such problems. Many algorithms exist to solve the matrix problems presented in previous sections, each with its own convergence properties.

The problem associated with the discrete-time Markov chain is that of solving for the left-hand eigenvector of a system. One algorithm designed for this purpose is the **power** method. From [1], the power method works by multiplying a vector by the given matrix repeatedly. This works because this repeated multiplication typically forces the vector to tend toward the dominant eigenvector. This is clear in the case of a discrete-time Markov chain as all eigenvalues other than the dominant value satisfy $|\lambda| < 1$, and the dominant eigenvalue $\lambda = 1$. Thus as the matrix is subsequently multiplied these values all tend toward 0, leaving the dominant eigenvector as the only nontrivial eigenvector solving:

$$\mathbf{A} \cdot \mathbf{v} = \mathbf{v}.$$

The algorithm can be written as such [1]:

Note the algorithm normalizes $\tilde{\mathbf{v}}$ to prevent any possible overflow rounding errors, and that the algorithm also stores intermediate approximations of the eigenvalue. Termination is usually set to occur when the differences

input : a matrix $\mathbf{A}$ and initial guess $\mathbf{v}_0$
output: the dominant eigenvector $\mathbf{v}$ of $\mathbf{A}$
for $k \leftarrow 1$ *until termination* **do**
  $\tilde{\mathbf{v}} = \mathbf{A}\mathbf{v}_{k-1}$
  $\mathbf{v}_k = \tilde{\mathbf{v}}/||\tilde{\mathbf{v}}||$
  $\lambda_1^{(k)} = \mathbf{v}_k^T \mathbf{A}\mathbf{v}_k$
end

Algorithm 1: Power Method

between two subsequent $\mathbf{v}_k$ is less than some given tolerance. That is:

$$|\mathbf{v}_k - \mathbf{v}_{k-1}| < \text{TOL}.$$

This signals the convergence of the algorithm. Usual tolerance values are chosen to be around 10^{-6}.

Despite its simplicity, this algorithm is extremely useful. Since it solves the eigenvalue and eigenvector problem it can be used either with a discrete-time Markov chain or an embedded Markov chain to solve for its steady state.

When working with continuous-time Markov chains instead of solving for the eigenvalues of the generator matrix, the equation (from equation (12))

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{0}.$$

must be solved. There are many different algorithms available to do this, but this paper will focus on the *relaxation methods* of Gauss-Seidel, Jacobi and Successive Over Relaxation. One important note in regards to the convergence of the relaxation methods is that the standard ones (Jacobi and Gauss-Seidel) will converge whenever the matrix $\mathbf{A}$ is uniformly diagonally dominant [1]. In addition to the relaxation methods, there will be discussion of Conjugate Gradient algorithm with some minor reference to the General Minimal Residual method.

The relaxation methods are all stationary methods that use a splitting of the matrix $\mathbf{A}$ to solve the equation $\mathbf{A}\mathbf{x} = \mathbf{b}$ for some given $\mathbf{A}$ and $\mathbf{b}$. That is [1] $\mathbf{A} = \mathbf{M} - \mathbf{N}$. This yields the equation $\mathbf{M}\mathbf{x} = \mathbf{N}\mathbf{x} + \mathbf{b}$ which can be rewritten as $\mathbf{x} = \mathbf{M}^{-1}\mathbf{N}\mathbf{x} + \mathbf{M}^{-1}\mathbf{b}$. This then allows for a *fixed point iteration*, that is taking an initial guess, $\mathbf{x}_0$ and applying this algorithm to this guess iteratively to achieve convergence to the correct value, to solve for $\mathbf{x}$:

$$\mathbf{x}_{k+1} = \mathbf{M}^{-1}\mathbf{N}\mathbf{x}_k + \mathbf{M}^{-1}\mathbf{b} = \mathbf{x}_k + \mathbf{M}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}_k).$$

Since this $\mathbf{x}_k$ is merely a guess and not the true value of $\mathbf{x}$, $(\mathbf{b} - \mathbf{A}\mathbf{x}_k) \neq \mathbf{0}$. We call this value the residual, $\mathbf{r}_k$. Rewriting $\mathbf{b} - \mathbf{A}\mathbf{x}_k = \mathbf{r}_k$ yields the iteration [1]:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{M}^{-1}\mathbf{r}_k. \tag{26}$$

One important note in regards to the convergence of the relaxation methods is that the standard ones (Jacobi and Gauss-Seidel) will converge whenever the matrix $\mathbf{A}$ is uniformly diagonally dominant.

The question at hand now, is how to define the matrix, $\mathbf{M}$. From [1], let $\mathbf{D}$ denote the diagonal matrix whose elements are the diagonal of $\mathbf{A}$, and let $\mathbf{L}$ and $\mathbf{U}$ denote the $n \times n$ lower and upper triangular matrices with the corresponding elements of $\mathbf{A}$ respectively. These two matrices can be used as the splitting matrix, $\mathbf{M}$, for two different methods.

Assigning $\mathbf{M} = \mathbf{D}$ yields the **Jacobi** method. Letting $\mathbf{N} = \mathbf{L} + \mathbf{U}$ yields the equality [4]:

$$\mathbf{A} = \mathbf{D} - (\mathbf{L} + \mathbf{U}).$$

This allows the iteration to be written as:

$$\mathbf{D}\mathbf{x}_{k+1} = (\mathbf{L} + \mathbf{U})\mathbf{x}_k.$$

which yields [4]:

$$\mathbf{x}_{k+1} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}_k. \tag{27}$$

This yields the iteration matrix for the Jacobi method:

$$\mathbf{H}_J = \mathbf{M}^{-1}\mathbf{N} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U}).$$

To execute the Jacobi method merely multiply a current guess $\mathbf{x}_k$ by the iteration matrix $\mathbf{H}_J$ to obtain the next guess, $\mathbf{x}_{k+1}$, then iterate. That is:

$$\mathbf{x}_{k+1} = \mathbf{H}_J\mathbf{x}_k.$$

This yields the eigenvector problem:

$$\mathbf{H}_J\mathbf{x} = \mathbf{x}. \tag{28}$$

Thus solving for the homogeneous solution of $\mathbf{A}$ is the same as solving for the right-hand eigenvector of $\mathbf{H}_j$. To analyze the nature of the eigenvalues we introduce Gerschgorin's theorem [5].

Theorem 2.1 (Gerschgorin's Circle Theorem)**.** *For an $n \times n$ matrix $\mathbf{A}$,*

define the discs

$$D_i = \left\{ z : |z - a_{ii}| \leq \sum_{j \neq i} |a_{ij}| \right\}$$

Then, each of the eigenvalues of $\mathbf{A}$ *is in at least one of the discs,* D_i*.*

When $\mathbf{A}$ is a zero column-sum matrix, Gerschgorin's Circle theorem asserts that no eigenvalue of $\mathbf{H}_J$ can be greater than 1 [4]. Therefore the dominant eigenvalue of $\mathbf{H}_J$ is one . This implies that the Jacobi method for a matrix $\mathbf{A}$ is equivalent to applying the Power method to the matrix $\mathbf{H}_J$.

Alternatively, assigning $\mathbf{M} = (\mathbf{D} - \mathbf{L})$ and $\mathbf{N} = \mathbf{U}$ yields the **Gauss-Seidel** method. This assignment allows for one improvement upon the Jacobi iteration. When the computer is executing the matrix-vector multiplication it has to do so using each individual component of these. Therefore some of the values $x_i^{(k+1)}$ of $\mathbf{x}_{k+1}$ will be calculated before others. This allows some use of these $x_i^{(k+1)}$ in the calculation of $\mathbf{x}_{k+1}$. This is beneficial because the current guess will be a more accurate guess than the previous. Gauss-Seidel iteration can be written as [4]:

$$x_i^{(k+1)} = \frac{1}{d_{ii}} \left(\sum_{j=1}^{i-1} l_{ij} x_j^{(k+1)} + \sum_{j=i+1}^{n} u_{ij} x_j^{(k)} \right), \qquad i = 1, 2, \ldots, n. \tag{29}$$

This can be written with matrices as:

$$(\mathbf{D} - \mathbf{L})\mathbf{x}_{k+1} = \mathbf{U}\mathbf{x}_k. \tag{30}$$

Which can be rewritten as:

$$\mathbf{x}_{k+1} = (\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}\mathbf{x}_k.$$

Which then yields the iteration matrix:

$$\mathbf{H}_{GS} = (\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}.$$

As before this implies that $\mathbf{x}$ is right-hand eigenvector of the equation

$$\mathbf{H}_{GS}\mathbf{x} = \mathbf{x}. \tag{31}$$

Since $\mathbf{H}_J$ has a dominant eigenvalue equal to one, the Stein-Rosenberg theorem asserts that the dominant eigenvalue of $\mathbf{H}_{GS}$ is also equal to one [4]. Thus the Gauss-Seidel method applied to a matrix $\mathbf{A}$ is equivalent to the Power method applied to $\mathbf{H}_{GS}$.

Occasionally a scaling factor, $\mathbf{D}^{-1}$, is multiplied to the matrix $\mathbf{A}$ to force

all elements along the diagonal to one. When this occurs, the iteration matrix is also scaled. It can be written as [4]

$$(\mathbf{I} - \mathbf{L}\mathbf{D}^{-1})^{-1}\mathbf{U}\mathbf{D}^{-1} = \mathbf{D}\big((\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}\big)\mathbf{D}^{-1}.$$

Since the matrix is only altered by a similar transformation this scaling has no effect on convergence of the Gauss-Seidel method.

Since values from the current iterate are being used in its calculation, the Gauss-Seidel method has both a forward and a backward form. The backward iteration can be written as [4]

$$(\mathbf{D} - \mathbf{U})\mathbf{x}_{k+1} = \mathbf{L}\mathbf{x}_k.$$

As a general rule of thumb forward iteration should be used when the majority of the elemental mass is below the diagonal and backward when the opposite is true. There are cases in which this rule fails though where the alternate form has a faster convergence.

The **Successive Over Relaxation** (SOR) method comes from taking the Gauss-Seidel method and multiplying it by a *relaxation parameter*, ω. The component-wise iteration for this method can be written as [4]

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \omega\left\{\frac{1}{d_{ii}}\left(\sum_{j=1}^{i-1} l_{ij}x_j^{(k+1)} + \sum_{j=i+1}^{n} u_{ij}x_j^{(k)}\right)\right\}, \qquad i = 1, 2, \ldots, n. \tag{32}$$

This can be written in matrix form as

$$\mathbf{x}_{k+1} = (1-\omega)\mathbf{x}_k + \omega\big\{\mathbf{D}^{-1}(\mathbf{L}\mathbf{x}_{k+1} + \mathbf{U}\mathbf{x}_k)\big\}. \tag{33}$$

Since this is merely a relaxed form of the Gauss-Seidel method a backward form of SOR can also be written. The iteration for backward SOR is

$$\mathbf{x}_{k+1} = (\mathbf{D} - \omega\mathbf{L})^{-1}\big[(1-\omega)\mathbf{D} + \omega\mathbf{U}\big]\mathbf{x}_k.$$

This gives the iteration matrix for SOR

$$\mathbf{H}_\omega = (\mathbf{D} - \omega\mathbf{L})\big[(1-\omega)\mathbf{D} + \omega\mathbf{U}\big]. \tag{34}$$

Note that in the case $\omega = 1$ SOR is identical to the Gauss-Seidel method. When $\omega < 1$ the process is called *underrelaxed*. Alternatively, when $\omega > 1$ the process is called *overrelaxed* [4].

Like the Jacobi and Gauss-Seidel methods SOR is the same as solving

the eigenvalue equation [4]

$$\mathbf{H}_\omega \mathbf{x} = \mathbf{x}.$$

However, for some values of ω the dominant eigenvalue is not equal to one. Thus SOR only is the same as the power method applied to $\mathbf{H}_\omega$ in those cases.

Selection of the relaxation parameter has a great effect on convergence of SOR. The method only converges when $0 < \omega < 2$. Due to its similarity to the power method the optimal choice of relaxation parameter is the value ω that minimizes the subdominant eigenvalue of $\mathbf{H}_\omega$. This minimization problem is different for every matrix thus there is no general way to find an optimal ω [4].

In addition to these iterative methods the **Conjugate Gradient** method can also be used to solve $\mathbf{Ax} = \mathbf{b}$. The conjugate gradient method can only be applied to matrices that are symmetric positive-definite, that is [4]:

$$\mathbf{A} = \mathbf{A}^T \quad \text{and} \quad \mathbf{x}^T \mathbf{A} \mathbf{x} > 0, \qquad \forall \mathbf{x} \neq 0.$$

This implies that solving $\mathbf{Ax} = \mathbf{b}$ is equivalent to minimizing

$$\phi(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} - \mathbf{b}^T \mathbf{x}. \tag{35}$$

Substituting $\mathbf{x} + \alpha \mathbf{v}$ for $\mathbf{x}$ yields

$$\phi(\mathbf{x} + \alpha \mathbf{v}) = \phi(\mathbf{x}) + \alpha (\mathbf{Ax} - \mathbf{b})^T \mathbf{v} + \frac{1}{2} \alpha^2 (\mathbf{v}^T \mathbf{A} \mathbf{v}).$$

This functional is minimized when α takes a value

$$\hat{\alpha} = \frac{(\mathbf{b} - \mathbf{Ax})^T \mathbf{v}}{\mathbf{v}^T \mathbf{A} \mathbf{v}}.$$

Thus the function's minimum is given by

$$\phi(\mathbf{x} + \alpha \mathbf{v}) = \phi(\mathbf{x}) - \frac{[(\mathbf{b} - \mathbf{Ax})^T \mathbf{v}]^2}{2 \mathbf{v}^T \mathbf{A} \mathbf{v}}. \tag{36}$$

This minimum is found via iterative methods.

These iterative methods work by first choosing some direction vector, $\mathbf{v}_k$, then finding the minimum of the functional in this direction. Then a new direction is chosen and then a new minimum found. This is repeated until the absolute minimum of the functional ϕ is found. Therefore, if α_k denotes

a scalar and $\mathbf{v}_k$ some vector the iteration can be written as [4]

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{v_k}.$$

In the method of conjugate gradients the search directions, $\mathbf{v}_i$ are chosen to create an $\mathbf{A}$-orthogonal system, or $\mathbf{v}_i^T \mathbf{A} \mathbf{v}_j = \delta_{ij}$. The algorithm can be found in pseudocode on the below.

input : a matrix $\mathbf{A}$, a vector $\mathbf{b}$, and initial guess $\mathbf{x}_0$
output: a solution to the equation $\mathbf{Ax} = \mathbf{b}$
$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{v}_0 = 0$
$\beta_0 = 0$
for $k \leftarrow 1$ *until termination* **do**
  $\mathbf{v}_k = \mathbf{r}_{k-1} + \beta_{k-1}\mathbf{v}_{k-1}$
  $\alpha_k = \mathbf{r}_{k-1}^T \mathbf{r}_{k-1} / \mathbf{v}_k^T \mathbf{A} \mathbf{v}_k$
  $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{v}_k$
  $\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{A} \mathbf{v}_k$
  $\beta_k = \mathbf{r}_k^T \mathbf{r}_k / \mathbf{r}_{k-1}^T \mathbf{r}_{k-1}$
end

Algorithm 2: Conjugate Gradient method

The conjugate gradient algorithm's largest downfall is the requirement that it can only be used for symmetric positive-definite matrices. One small adjustment to the matrix will allow the use of the conjugate gradient algorithm for any real matrix. Given a real matrix, $\mathbf{A}$, $\mathbf{A}^T\mathbf{A}$ is symmetric positive-definite. Therefore, the conjugate gradient method can be used to solve the *normal equations*, [4]

$$\mathbf{A}^T \mathbf{A} \mathbf{x} = \mathbf{A}^T \mathbf{b}.$$

Pseudocode for this altered version of the conjugate gradient method, referred to as *conjugate gradient for the normal equations* can be found on the following page. In the pseudocode a preconditioner $\mathbf{M} = \mathbf{I}$ is used. For purposes of this paper, from this point forward whenever the Conjugate Gradient method is referenced, the Conjugate Gradient method for the Normal Equations is actually being referenced. This is due to the fact that in the reliability problem of interest the generator matrix and its transpose are not symmetric and positive-definite.

input : a matrix $\mathbf{A}$, a vector $\mathbf{b}$, and initial guess $\mathbf{x}_0$
output: a solution to the equation $\mathbf{Ax} = \mathbf{b}$
$\mathbf{r}_0 = \mathbf{M}^{-1}(\mathbf{b} - \mathbf{Ax}_0)$
$\mathbf{v}_0 = 0$
$\beta_0 = 0$
$k = 0$
while $\mathbf{r}_k^T \mathbf{r}_k > TOL$ **do**
  $k = k + 1$
  $\mathbf{v}_k = \mathbf{A}^T \mathbf{M}^{-T} \mathbf{r}_{k-1} + \beta_{k-1} \mathbf{v}_{k-1}$
  $\alpha_k = \mathbf{r}_{k-1}^T \mathbf{r}_{k-1} / \mathbf{v}_k^T \mathbf{v}_k$
  $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{v}_k$
  $\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{M}^{-1} \mathbf{A} \mathbf{v}_k$
  $\beta_k = \mathbf{r}_k^T \mathbf{r}_k / \mathbf{r}_{k-1}^T \mathbf{r}_{k-1}$
end

Algorithm 3: Conjugate Gradient for the Normal Equations

3 A Reliability Problem

Within reliability theory, there are many different ways to formulate a reliability problem. Frequently, Naval engineers encounter systems structured as two primary identical systems working toward the same goal. Since the two systems are effectively identical they are both completing the same task and thus can be thought of as being in parallel (figure 1). The structure function

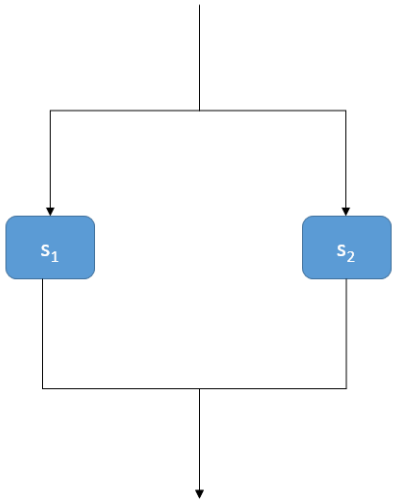

Figure 1: Diagram of two parallel systems

for this general system is quite simple and can be given by equation (16) with $n = 2$. However, these two identical systems are both made up of n identical subsystems. Varying levels of functionality can be applied to the primary systems depending upon the number of functioning subsystems (i.e. system still functioning as long as k subsystems are still functioning). This manifests in the definition of the states and thus has little effect on the convergence of an algorithm attempting to find the steady state solution. The ordering of these states can drastically effect the convergence of an algorithm though, as the ordering controls the shape and structure of the generator matrix.

An ordering that keeps the elemental mass of the matrix as close to the diagonal as possible is preferable. Stewart presents an example of an ordering for a two-subsystem system [4]. This is then extrapolated to be for n subsystems instead of two. An example of ordering and the generator matrix that it creates for a three-subsystem system is as follows. λ_i denotes the mean time between failure and μ_i denotes the mean time to repair for each subsystem within system $i = 1, 2$. Ordering: (written as ordered pairs

where each number corresponds to the number of functioning subsystems in that primary system)

$$(3,3),(3,2),(3,1),(3,0),(2,3),(2,2),(2,1),(2,0),(1,3),(1,2),(1,1),(1,0),$$
$$(0,3),(0,2),(0,1),(0,0).$$

That is, $(3,3)$ is the first state (state 0) and $(0,0)$ is the last state (state 15). This ordering follows decreasing order across both subsystems, but cycles through all possible values of subsystem two before changing subsystem one. Using subscripts to denote the values for the specific system, the generator matrix for this ordering can be constructed:

$$\mathbf{Q} = \begin{pmatrix}
-s_0 & 3\lambda_2 & 0 & 0 & 3\lambda_1 & 0 & \cdots & & & & & & & & \\
\mu_2 & -s_1 & 2\lambda_2 & 0 & 0 & 3\lambda_1 & 0 & \cdots & & & & & & & \\
0 & 2\mu_2 & -s_2 & \lambda_2 & 0 & 0 & 3\lambda_1 & 0 & \cdots & & & & & & \\
0 & 0 & 3\mu_2 & -s_3 & 0 & 0 & 0 & 3\lambda_1 & 0 & \cdots & & & & & \\
\mu_1 & 0 & 0 & 0 & -s_4 & 3\lambda_2 & 0 & 0 & 2\lambda_1 & 0 & \cdots & & & & \\
0 & \mu_1 & 0 & 0 & \mu_2 & -s_5 & 2\lambda_2 & 0 & 0 & 2\lambda_1 & 0 & \cdots & & & \\
\cdots & 0 & \mu_1 & 0 & 0 & 2\mu_2 & -s_6 & \lambda_2 & 0 & 0 & 2\lambda_1 & 0 & \cdots & & \\
 & \cdots & 0 & \mu_1 & 0 & 0 & 3\mu_2 & -s_7 & 0 & 0 & 0 & 2\lambda_1 & 0 & \cdots & \\
 & & \cdots & 0 & 2\mu_1 & 0 & 0 & 0 & -s_8 & 3\lambda_2 & 0 & 0 & \lambda_1 & 0 & \cdots \\
 & & & \cdots & 0 & 2\mu_1 & 0 & 0 & \mu_2 & -s_9 & 2\lambda_2 & 0 & 0 & \lambda_1 & 0 \\
 & & & & \cdots & 0 & 2\mu_1 & 0 & 0 & 2\mu_2 & -s_{10} & \lambda_2 & 0 & 0 & \lambda_1 \\
 & & & & & \cdots & 0 & 2\mu_1 & 0 & 0 & 3\mu_2 & -s_{11} & 0 & 0 & 0 \\
 & & & & & & \cdots & 0 & 3\mu_1 & 0 & 0 & 0 & -s_{12} & 3\lambda_2 & 0 \\
 & & & & & & & \cdots & 0 & 3\mu_1 & 0 & 0 & \mu_2 & -s_{13} & 2\lambda_2 \\
 & & & & & & & & \cdots & 0 & 3\mu_1 & 0 & 0 & 2\mu_2 & -s_{14} \\
 & & & & & & & & & \cdots & 0 & 3\mu_1 & 0 & 0 & 3\mu_2 \\
 & & & & & & & & & & \cdots & 0 & 3\mu_1 & 0 & 0 \\
 & & & & & & & & & & & \cdots & 0 & 3\mu_1 & 0 \\
 & & & & & & & & & & & & \cdots & 0 & 3\mu_1 \\
 & & & & & & & & & & & & & \cdots & 0 \\
 & & & & & & & & & & & & & & \cdots
\end{pmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix}. \tag{37}$$

Note the banded structure of this matrix which will ease computational methods of solution later. This banded structure is preserved for any number of subsystems, however the spacing between the two non-zero diagonals above and below the main diagonal will differ depending on n. For two subsystems there is one diagonal of zeros between them, for three (shown) there are two, and this pattern continues for n. That is the number of zero diagonals is $n-1$.

Once all λ and μ values are determined this generator matrix fully describes the Markov chain for this reliability problem. Theoretical values were assigned to these two parameters in this research. Unless otherwise stated

$$\lambda = \frac{1}{3600}$$

and

$$\mu = \frac{1}{60}.$$

That is the mean time between each components failure is assumed to be 3600 hours and the mean time to repair for each component was assumed to

be 60 hours. In practice these values are relatively accurate for some specific systems found at the Applied Research Lab.

Once the generator matrix is constructed, what remains is solving for the steady state solution of the given Markov chain. Computationally this can be done very quickly for small systems, however as the system grows in size this can become very difficult. Note how quickly the size of the generator matrix grows with respect to the number of subsystems. A system with n subsystems creates an $(n+1)^2 \times (n+1)^2$ matrix. Thus it is important to use efficient methods when calculating the steady state solution of this Markov chain. The following section analyzes computational speeds of various methods when applied to this system.

4 Computational Analysis

Since solving for the steady state solution of a continuous-time Markov chain is the same as solving the equation

$$\mathbf{Q} \cdot \mathbf{x} = \mathbf{0}$$

many different computational algorithms can be used. The general structure of these algorithms was outlined in the literature review portion of this paper. The power method cannot be directly used to solve any such systems as it is actually an eigenvalue solution, however both the Gauss-Seidel and the Successive Over Relaxation method can be thought of as altered versions of this algorithm.

All testing on the convergence of these algorithms with the given system was done via MATLAB. Convergence rates will obviously differ depending on which programming language is used and what packages were used within that language. In any case possible, the operation $\mathbf{x} = \mathbf{A}\backslash\mathbf{b}$ was used over $\mathbf{x} = \texttt{inv}(\mathbf{A}) \cdot \mathbf{b}$ the $\backslash$ command has much faster runtime than $\texttt{inv}(*)$.

To gain a general understanding of the convergence of some of these algorithms a test was conducted to see how each of their convergence times grow with the number of subsystems in each parallel component. First an algorithm to construct the generator matrix for the given n-subsystem reliability problem was constructed. The algorithm to do so should only require inputs of the number of subsystems or components, the mean time between failure and the mean time to repair for each of these components. From there it should construct the banded generator matrix shown in equation (34). This algorithm is included on the following page. Before the algorithms can be applied to the generator matrix to solve for steady a steady state solution, an initial guess must be determined. To eliminate any possible variability in convergence times a normalized uniform vector of size $(n+1)^2$ was used as the initial guess, $\mathbf{x}_0$.

Using MATLAB's internal clock, the Gauss-Seidel, Generalized Minimal Residual, and Conjugate Gradient algorithms were all tested. To do so, a script was written to first construct the generator matrix and initial guess, then apply each of these three algorithms to that pair while measuring the elapsed time with the built-in `tic` and `toc` commands. This was done for systems with the number of subsystems ranging from one to fifty. To ensure accuracy in the convergence times the test was conducted 30 times and the convergence rates were averaged. When looking at all 30 trials, values from the first trial had consistently longer run-times, most likely due to some sort of initialization, so these values were thrown out and the subsequent

input : total number of subsystems n, parameter values for both parallel systems $\lambda_1, \lambda_2, \mu_1$ and μ_2
output: the generator matrix $\mathbf{Q}$, of the given reliability problem
$\texttt{nodes} = (n+1)^2$
Initialize $\mathbf{Q}$ as a matrix of size $\texttt{nodes} \times \texttt{nodes}$
$\texttt{modBase} = n + 1$
for $k \leftarrow 0$ *to* $\texttt{nodes} - 1$ **do**
  if $k + 1 \leq \texttt{nodes}$ **then**
    $q_{k,k+1} = \texttt{mod}[-k, \texttt{modBase}] \cdot \lambda_2$
  end
  if $k + 1 + n \leq \texttt{nodes}$ **then**
    $q_{k,k+1+n} = (n - \texttt{ceil}[\frac{k}{\texttt{modBase}}] + 1) \cdot \lambda_1$
  end
  if $k - 1 > 0$ **then**
    $q_k k, k-1 = \texttt{mod}[k-1, \texttt{modValue}] \cdot \mu_2$
  end
  if $k - 1 - n > 0$ **then**
    $q_{k,k-1-n} = (\texttt{ceil}[\frac{k}{\texttt{modValue}}] - 1) \cdot \mu_1$
  end
  $q_{k,k} = -\sum_j q_{k,j}$
end

Algorithm 4: Generator Matrix Constructor

29 trials were averaged. The standard deviations for the average for the Gauss-Seidel, General Minimal Residual and Conjugate Gradient methods as a percentage of the actual average at 50 subsystems were 2.53%, 1.69% and 1.08% respectively. Plots are included, as figures 1 and 2, of the time elapsed versus the number of subsystems.

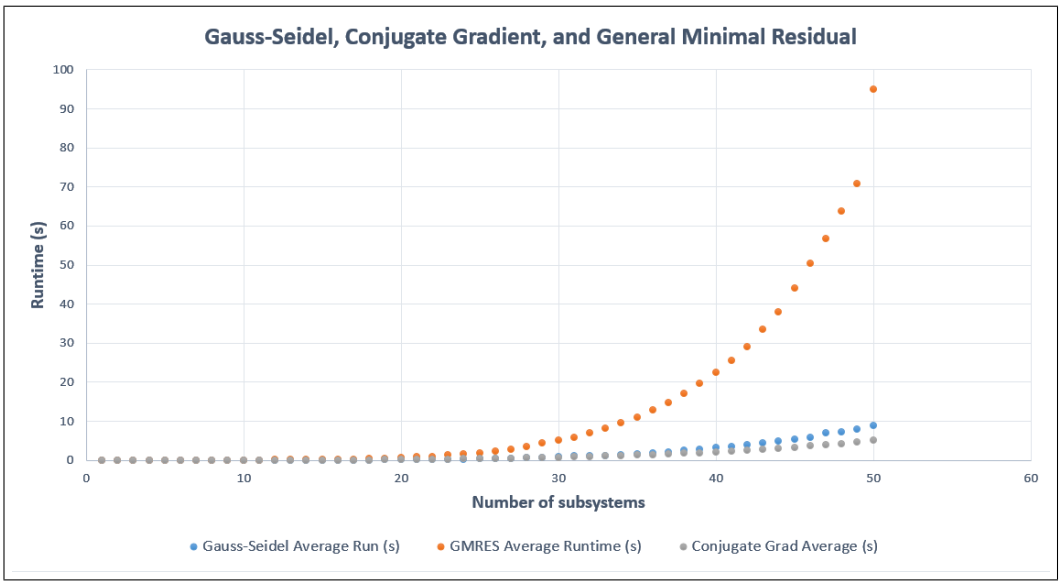

Figure 2: Initial test of the Gauss-Seidel, Conjugate Gradient, and General Minimal Residual methods applied to the generator matrix with 30 subsystems

As expected, at smaller matrix sizes all three had very similar convergence times. However, as the size began to grow the General Minimal Residual method diverged from both Gauss-Seidel and the Conjugate gradient and had a much longer runtime, thus eliminating the General Minimal Residual as a good algorithmic choice for the given problem. Looking more closely at the Gauss-Seidel and Conjugate Gradient methods it can be seen that the Conjugate Gradient method has a relatively significant advantage over the Gauss-Seidel as the number of subsystems approaches 50. At 50 subsystems it takes the Gauss-Seidel method about 9 seconds to converge where as it only takes the Conjugate Gradient method about 5 seconds. This speed implies a decent scalability of the problem as the matrix size with 50 subsystems is $(50+1)^2 \times (50+1)^2 = 2601 \times 2601$.

Since in the first test the Conjugate Gradient algorithm had the fastest convergence time other conjugate direction methods were tested as well. The Biconjugate Gradient and Conjugate Gradient squared methods were both

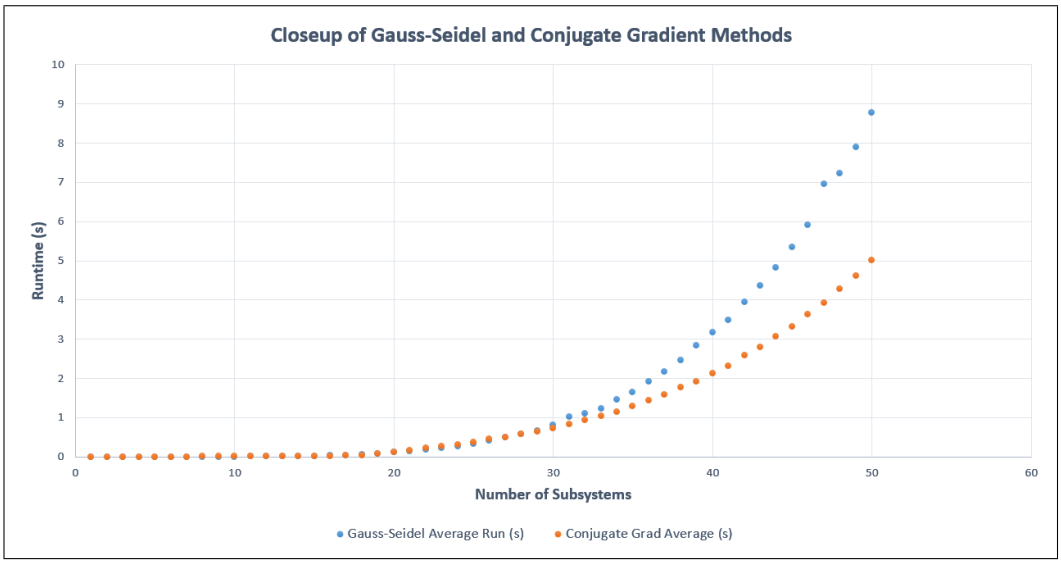

Figure 3: A closer view of the running times of the Gauss-Seidel and Conjugate Gradient methods

implemented in MATLAB so they could be directly compared with the Conjugate Gradient method. Again, using the `tic` and `toc` commands and MATLAB's internal clock, the run-times of the Conjugate Gradient, Biconjugate Gradient and Conjugate Gradient Squared methods were recorded for systems with a number of subsystems ranging from 1 to 50. These were then taken over 30 trials, the first trial was thrown out, and the subsequent 29 trials were averaged together. Plot for these trials are included in figures 4 and 5.

Like the General Minimal Residual method, the Conjugate Gradient Squared method's convergence time diverged from that of the Conjugate Gradient and Biconjugate Gradient methods. The Biconjugate Gradient method had an interesting almost stepwise change in its convergence times. As n approaches 50, note how the convergence time only has a significant change at iterations which have an even number of subsystems. At iterations with an odd number of subsystems the convergence time stays at about the same value as the previous even iteration, in some instances it even decreases slightly.

After looking at Conjugate Direction methods, the Successive Over Relaxation method was also analyzed. The convergence rate of this algorithm has a lot more variance than any of the previously discussed algorithms. This is due to the relaxation parameter, ω. The algorithm will converge for $0 \leq \omega < 2$, therefore any of these infinite possible values for ω could be

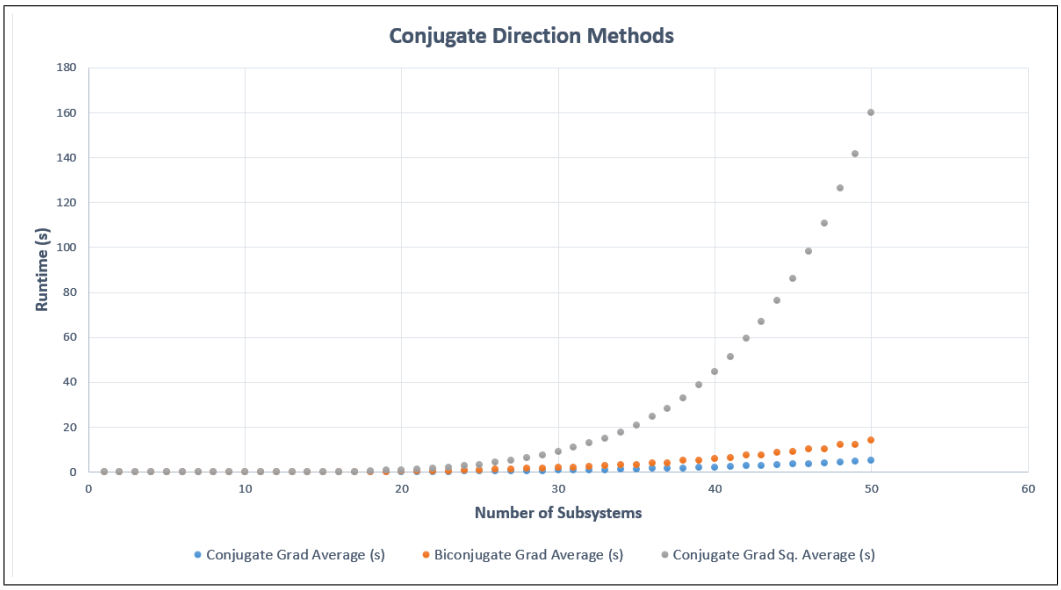

Figure 4: Comparison of running times of Conjugate Gradient, Biconjugate Gradient and Conjugate Gradient Squared methods applied to $\mathbf{Q}^T$

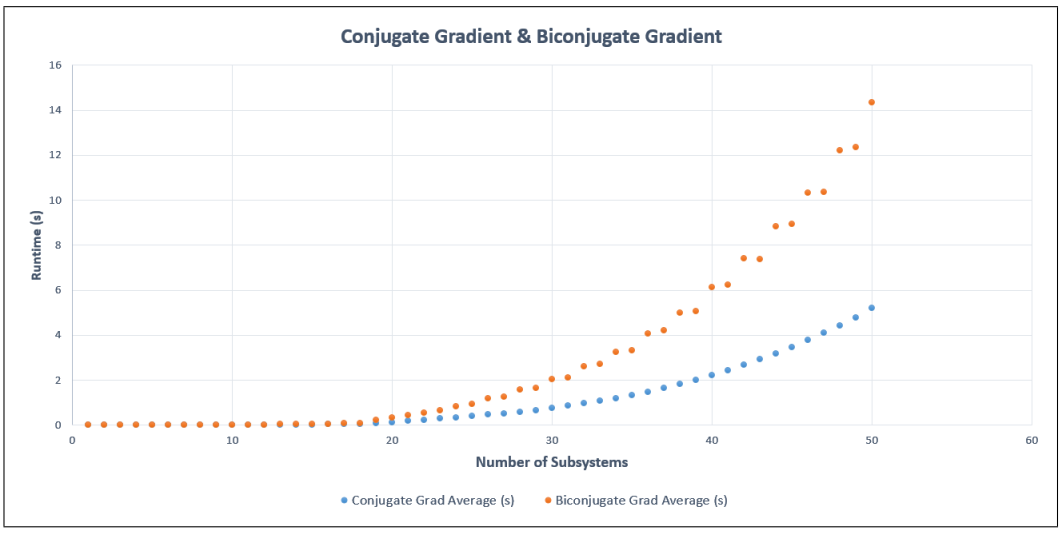

Figure 5: A closer view of the Conjugate Gradient and Biconjugate Gradient methods

used to calculate the steady state solution of the given reliability problem. However, since speed is the key , the right ω value must be chosen to achieve a desirable result. As noted in section 2.3, since the Successive Over Relaxation method is the same as the Power method with an iteration matrix of $\mathbf{H}_\omega$, (32), the optimal ω value will be the one that forces the subdominant

eigenvalue of $\mathbf{H}_\omega$ to be minimal [4]. This is a rather difficult problem as calculation of the eigenvalue of a matrix is not computationally trivial and there are effectively an infinite number of possible choices for ω. Due to the floating point nature and finite precision of computational systems there are not an actually infinite number of values for ω as once the precision of the computer is reached all smaller deviations of ω will not be stored computationally as a different value for ω.

To gain a better understanding of the behavior of the subdominant eigenvalue of $\mathbf{H}_\omega$ an initial test was conducted with 35 subsystems. This was done by first creating the generator matrix for the reliability system with 35 subsystems using the Generator Matrix Constructor algorithm. Then, starting with an ω-value of 1 and increasing in increments of 0.001, $\mathbf{H}_\omega$ was determined for values of ω ranging from 1.001 to 1.5. Then MATLAB's built-in command `eig`$(*)$ was used to obtain a vector, $\mathbf{u}$, of the eigenvalues of $\mathbf{H}_\omega$, Then the subdominant eigenvalue from this vector was taken and stored in another vector of size 500. Due to the form of the generator matrix, the subdominant eigenvalue is also the second eigenvalue. A short algorithm to conduct this test is included below. With the Successive Over Relaxation

input : a generator matrix $\mathbf{Q}$, a range of possible values for the relaxation parameter $\omega \in [\alpha, \beta]$, and a mesh size `mesh`
output: a vector of subdominant eigenvalues of $\mathbf{H}_\omega$, the SOR iteration matrix, $\mathbf{v}$
`iters` $= (\beta - \alpha)/$`mesh`
for $k \leftarrow 1$ *to* `iters` **do**
  $\mathbf{H}_\omega = (\mathbf{D} - \omega\mathbf{L})\big[(1-\omega)\mathbf{D} + \omega\mathbf{U}\big]$
  $\mathbf{u} =$ `eig`$(\mathbf{H}_\omega)$
  $v_k = u_1$
end

Algorithm 5: Testing ω-values in Successive Over Relaxation

method, the choice of an ω-value can force some of the eigenvalues of the iteration matrix, $\mathbf{H}_\omega$, to be complex. Since there is a possibility that some of these subdominant eigenvalue are complex, the magnitude of each of these values must be used instead of the actual value. To obtain this, MATLAB's built in function `abs`$(*)$ was applied to the vector $\mathbf{v}$,

$$\mathbf{r} = \texttt{abs}(\mathbf{v}).$$

This vector, $\mathbf{r}$, was then plotted using MATLAB's built-in `plot`$(*)$ command was used. A simple transformation of the x-axis transforms this to be a plot

with respect to ω,

$$\omega = 1 + \texttt{mesh} \cdot x. \tag{38}$$

A plot for the case with 35 subsystems is included in figure 6. Note the

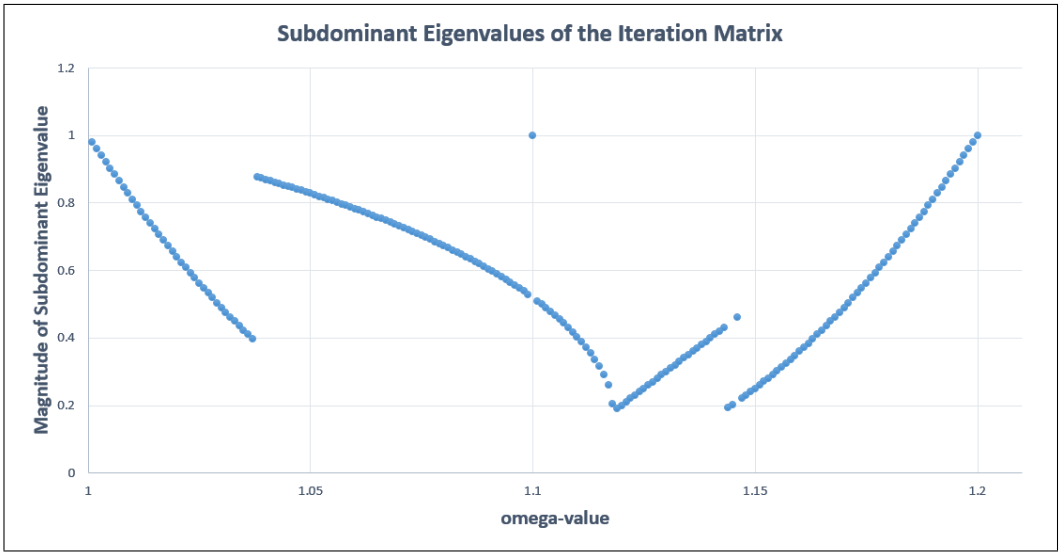

Figure 6: Effects of ω on the magnitude of the subdominant eigenvalue of the matrix $\mathbf{H}_\omega$

initial decline with a sharp spike immediately following. This local minimum is where the optimal ω-value is found. With this level of precision,

$$\omega = 1.037.$$

Analyzing the point at which complex behavior is found yields

$$\omega \geq 1.065 \quad \Longrightarrow \quad \lambda_2^{(e)} \in \mathbb{C},$$

where $\lambda_2^{(e)}$ is the subdominant eigenvalue of $\mathbf{H}_\omega$. Note that the two additional local minimum are after this point. Since these values of ω will yield an undesirable underflow in the calculation of a solution they can be ignored.

This underflow forces some intermediate computations of the eigenvector to register as negative. This has an observable effect on the solution given via Successive Over Relaxation, and can cause the eigenvector to converge to a value that is not trivially different from the correct solution. This is obviously an undesirable result and thus those choosing to use Successive Over Relaxation to solve the given reliability problem should use caution in this selection process. Alternatively, the Gauss-Seidel or conjugate gradient

methods could be used instead.

5 Conclusion and Future Work

5.1 Conclusions

Computational methods are an effective way of gaining an understanding of the limiting probabilities of Markov chains, which are an effective way of representing many problems found within reliability theory. However, due to differences in structure, many of these methods will have noticeable differences in convergence rates.

Ultimately, the Conjugate Gradient method has the fastest convergence when applied to the generator matrix of the parallel reliability system of interest. Thus for large systems, it is most useful to apply the Conjugate Gradient method and check if the steady state solution supplied by this algorithm is feasible. However, the Conjugate Gradient method will only surely converge in the case of a symmetric positive definite matrix. Since this is not definitely the case with the given system and testing shows that in the case of small matrices the choice of algorithm has little effect on the convergence time as all algorithms converge extremely quickly it would make sense to use an algorithm that is sure to converge in any case. Thus the Gauss-Seidel method would be the most effective choice in the small matrix case.

5.2 Future Work

To gain a full understanding of the reliability of a system the transient states, $\boldsymbol{\pi}(t)$, must be determined. This is a much more difficult problem than that of finding the steady states of a solution but would be very useful in engineering applications. It would allow engineers to not only understand the probability of their system having a long lifetime, but they would also have an understanding of its functionality at any point in time.

The Markov chains presented here could also be extended to the idea of a Markov decision process. This would allow the extension of the reliability problem to include situations in which the subsystems could be repaired before breaking. This would allow engineers to create an effective scheduling procedure to determine at what time components should be repaired instead of simply waiting until they fail.

References

[1] Uri M. Ascher and Chen Greif, *A first course in numerical methods*, SIAM, 2011.

[2] Lawrence M. Leemis, *Reliability: probabilistic models and statistical methods*, second edition ed., Lawrence M. Leemis, 2009.

[3] Sheldon M. Ross, *Introduction to probability models*, Academic Press, 2003.

[4] William J. Stewart, *Introduction to the numerical solution of markov chains*, Princeton University Press, 1994.

[5] Eric W. Weisstein, *Gershgorin circle theorem*, `http://mathworld.wolfram.com/GershgorinCircleTheorem.html`, From *Mathworld*–A Wolfam Web Resource, Accessed 03/26/15.

Academic Vita

Seth Henry
smh5777@psu.edu

Education

The Pennsylvania State University, Spring 2015

Bachelor of Science in Mathematics

Bachelor of Science in Physics

Honors in Mathematics

Honors and Awards

Schreyer Honors College Gateway Scholar, Summer 2012

Mathematics Scholarship – Lampeter Strasburg High School, Spring 2011

Engineering Design team project winner sponsored by Xerox, Fall 2011

Ean Hong Memorial Scholarship - Triangle Fraternity, Fall 2013

Publications

2015 Spring Simulation Multi-Conference – The Society for Modeling & Simulation International. Published a short paper and poster titled "Convergence of Linear Algebraic Reliability Simulation."

Activities

Triangle Fraternity – A Fraternity of engineers, scientists, and architects. Member Fall 2012 - present, recruitment chair Fall 2013 - Spring 2014