Various Methods for Solving Tridiagonal Systems

Hung Doan

March 6, 2018

Abstract

When solving systems of equations, specifically tridiagonal systems, there are various methods that may be used. I will be discussing direct methods such as Guassian Elimination with and without partial pivoting and LU factorization on tridiagonal systems, and iterative methods such as Jacobi, Guass-Seidel, and Successive Overrelaxation method.

1 Introduction

Consider the tridiagonal matrix T given by

$$T = \begin{bmatrix} b_1 & a_1 & 0 & 0 \\ c_1 & b_2 & a_2 & 0 \\ 0 & c_2 & b_3 & a_3 \\ 0 & 0 & c_3 & b_4 \end{bmatrix}.$$

Given that the LU factorization of T is

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{c_1}{d_1} & 1 & 0 & 0 \\ 0 & \frac{c_2}{d_2} & 1 & 0 \\ 0 & 0 & \frac{c_3}{d_3} & 1 \end{bmatrix}, \qquad U = \begin{bmatrix} d_1 & a_1 & 0 & 0 \\ 0 & d_2 & a_2 & 0 \\ 0 & 0 & d_3 & a_3 \\ 0 & 0 & 0 & d_4 \end{bmatrix}$$

with $d_1 = b_1$ and $d_{i+1} = b_{i+1} - \frac{a_i c_i}{d_i}$.

The above is an LU factorization for the 4x4 matrix T. However, this LU factorization works for tridiagonal matrices of arbitrary size. I will describe in detail the LU factorization for tridiagonal matrices of arbitrary length. Specifically, I will be working on tridiagonal matrices with the parameters $a_i = c_i = -1$, for $i = 1, ..., n-1$, and $b_i = 2$ for $i = 1, ..., n$. Plugging in these parameters, the generalized tridiagonal matrix has the form

$$T = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \cdots & 0 \\ 0 & -1 & 2 & -1 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix}.$$

For different values of n, I will be using Guassian Elimination with and without partial pivoting, tridiagonal LU factorization, Jacobi, Gauss-Seidel, and successive overralation to solve $Ax = b$ given that $b = [1, 0, ..., 0, 1]$.

2 Solving $Ax = b$

2.1 Gauss Elimination without Partial Pivoting

Consider the augmented matrix

$$\begin{bmatrix} a_{11} & a_{12} & ... & a_{1n} & b_1 \\ a_{21} & a_{22} & ... & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & ... & a_{mn} & b_m \end{bmatrix}.$$

In order to solve for x using Gaussian Elimination, we must first transform this matrix in to an upper triangular matrix and then perform backwards substitution.

To transform this matrix in to an upper triangular matrix, begin by zeroing out the first column by performing the following steps,

$$R_1 \rightarrow R_1$$
$$R_2 - \frac{a_{21}}{a_{11}} R_1 \rightarrow R_2$$
$$R_3 - \frac{a_{31}}{a_{11}} R_1 \rightarrow R_3$$
$$R_i - \frac{a_{i1}}{a_{11}} R_1 \rightarrow R_i$$

Our matrix will now have the form

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & ... & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & ... & a_{2n}^{(2)} & b_2^{(2)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a_{m2}^{(2)} & ... & a_{mn}^{(2)} & b_m^{(2)} \end{bmatrix}.$$

If $l_{i1} = \frac{a_{i1}}{a_{11}}$, $i = 2, ..., m$, then $a_{ij}^{(2)} = a_{ij} - l_{i1} a_{1j}$ for $i = 2, ..., n$ and $b_i^{(2)} = b_i - l_{i1} b_1$ for $i = 2, ..., m$. In general, if $l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$, $i = k+1, ..., n$ then $a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik}^{(k)} a_{kj}^{(k)}$ and $b_i^{(k+1)} = b_i^{(k)} - l_{ik}^{(k)} b_k^{(k)}$ for $i = k+1, ..., m, j = k+1, ..., n$.

Now that the augmented matrix is upper triangular, perform backward substitution to obtain the value for x.

2.2 Guass Elimination with Partial Pivoting

Gaussian Elimination without partial pivoting can run in to problems if any element along the diagonal is zero. To remedy this, we can perform partial pivoting. To perform partial pivoting, at the start of the k^{th} step of Gaussian-elimination, interchange rows k and r. Select r so that

$$|ar_k^{(k)}| = \max |a_{ik}^k|, \qquad \text{for } k \leq i \leq n$$

If Guassian elimination with partial pivoting is used to compute the upper triangularization

$$M_{n-1} P_{n-1} ... M_2 P_2 M_1 P_1 T = U.$$

2.3 Tridiagonal LU Factorization

Consider the tridiagonal matrix T given by

$$T = \begin{bmatrix} b_1 & a_1 & 0 & 0 \\ c_1 & b_2 & a_2 & 0 \\ 0 & c_2 & b_3 & a_3 \\ 0 & 0 & c_3 & b_4 \end{bmatrix}.$$

with LU Factorization given by

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{c_1}{d_1} & 1 & 0 & 0 \\ 0 & \frac{c_2}{d_2} & 1 & 0 \\ 0 & 0 & \frac{c_3}{d_3} & 1 \end{bmatrix}, \qquad U = \begin{bmatrix} d_1 & a_1 & 0 & 0 \\ 0 & d_2 & a_2 & 0 \\ 0 & 0 & d_3 & a_3 \\ 0 & 0 & 0 & d_4 \end{bmatrix}$$

with $d_1 = b_1$ and $d_{i+1} = b_{i+1} - \frac{a_i c_i}{d_i}$.

To verify that this is indeed the LU factorization of T, multiply L and U to obtain

$$LU = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{c_1}{d_1} & 1 & 0 & 0 \\ 0 & \frac{c_2}{d_2} & 1 & 0 \\ 0 & 0 & \frac{c_3}{d_3} & 1 \end{bmatrix} \begin{bmatrix} d_1 & a_1 & 0 & 0 \\ 0 & d_2 & a_2 & 0 \\ 0 & 0 & d_3 & a_3 \\ 0 & 0 & 0 & d_4 \end{bmatrix} = \begin{bmatrix} d_1 & a_1 & 0 & 0 \\ c_1 & \frac{c_1 a_1}{d_1} + d_2 & a_2 & 0 \\ 0 & c_2 & \frac{c_2 a_2}{d_2} + d_3 & a_3 \\ 0 & 0 & c_3 & \frac{c_3 a_3}{d_3} + d_4 \end{bmatrix}$$

where

$$\begin{aligned} d_2 = b_2 - \frac{a_1 c_1}{d_1} &\implies b_2 = d_2 + \frac{a_1 c_1}{d_1} \\ d_3 = b_3 - \frac{a_2 c_2}{d_2} &\implies b_3 = d_3 + \frac{a_2 c_2}{d_2} \\ d_4 = b_4 - \frac{a_3 c_3}{d_3} &\implies b_4 = d_4 + \frac{a_3 c_3}{d_3}. \end{aligned}$$

Substituting the values along the diagonal, we have that

$$T = \begin{bmatrix} b_1 & a_1 & 0 & 0 \\ c_1 & b_2 & a_2 & 0 \\ 0 & c_2 & b_3 & a_3 \\ 0 & 0 & c_3 & b_4 \end{bmatrix}.$$

In order to generalize the LU factorization for T, let T be given by

$$T = \begin{bmatrix} b_1 & a_1 & 0 & \cdots & 0 \\ c_1 & b_2 & a_2 & \cdots & 0 \\ 0 & c_2 & b_3 & a_3 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & c_{n-1} & b_n \end{bmatrix}.$$

Let

$$a_i = T_{i,i+1} \text{ for } i = 1, ..., n-1$$

$$c_i = T_{i+1,i} \text{ for } i = 1, ..., n-1$$

$$b_i = T_{i,i} \text{ for } i = 1, ..., n.$$

To find d,

$$d_1 = b_1$$

$$d_{i+1} = bi + 1 - a_i \frac{c_i}{d_i} \text{ for} i = 1, ..., n-1.$$

To find L,

$$L_{i,i} = 1 \text{ for } i = 1, ..., n$$

$$L_{i+1,i} = \frac{c_i}{d_i} \text{ for } i = 1, ..., n-1.$$

To find U,

$$U_{i,i} = d_i \text{ for } i = 1, ..., n$$

$$U_{i,i+1} = a_i \text{ for } i = 1, ..., n-1.$$

The above values of L and U will provide the LU factorization for a tridiagonal matrix of arbitrary length.

2.4 The Jacobi Iterative Method

So far, we have only explored direct methods for solving for x. The Jacobi method is an iterative method for solving for x. To do this, start with an initial guess, $x^{(0)}$, and construct a sequence $x^{(1)}, x^{(2)}, ..., x^{(k)}$ of approximations to the solution. The idea is that this sequence will converge to the actual value of x. However, the sequence will only converge if the matrix, T, is diagonally dominant. Fortunately,

$$T = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \cdots & 0 \\ 0 & -1 & 2 & -1 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

is diagonally dominant for any T of arbitrary length.

Consider

$$\begin{bmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & ... & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

Each equation in this system can be written as

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2$$

$$\vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n = b_n.$$

Solving for x_i we have

$$x_1^{(k+1)} = \frac{b_1 - a_{12}x_2^{(k)} - ... - a_{1n}x_n^{(k)}}{a_{11}}$$

$$x_2^{(k+1)} = \frac{b_2 - a_{21}x_1^{(k)} - ... - a_{2n}x_n^{(k)}}{a_{22}}$$

$$\vdots$$

$$x_n^{(k+1)} = \frac{b_n - a_{m1}x_1^{(k)} - ... - a_{mn}x_n^{(k)}}{a_{nn}}.$$

In general, we have

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}}{a_{ii}}.$$

If T is diagonally dominant, x_i will converge to x. When using this method, I ran these iterations until

$$\left\| x^{(k)} - x^{(k+1)} \right\| < 10^{-8}.$$

2.5 The Gauss-Seidel Iterative Method

This method is very similar to the Jacobi method except it uses new values of x_i as they come up whereas the Jacobi does not. Using the same system as we did in the Jacobi, when solving for x_i we have the following formulas:

$$x_1^{(k+1)} = \frac{b_1 - a_{12}x_2^{(k)} - ... - a_{1n}x_n^{(k)}}{a_{11}}$$

$$x_2^{(k+1)} = \frac{b_2 - a_{21}x_1^{(k+1)} - ... - a_{2n}x_n^{(k)}}{a_{22}}$$

$$\vdots$$

$$x_n^{(k+1)} = \frac{b_n - a_{m1}x_1^{(k+1)} - ... - a_{mn}x_n^{(k+1)}}{a_{nn}}.$$

As we obtain values for $x_i^{(k+1)}$ we use them in the proceeding formulas to solve for $x_{i+1}^{(k+1)}, x_{i+2}^{(k+1)}, ..., x_n^{(k+1)}$. By doing this, we can converge to x with much fewer iterations. The generalized formula is given by

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}}{a_{ii}}.$$

2.6 The Successive Over-relaxation Method

This is another improvement of the Jacobi method. In addition to the improvements made by the Gauss-Seidel method, a relaxation parameter, w, is added to provide additional improvements. If $w > 1$, we have over relaxation. When $w = 1$, this is simply the Gauss-Seidel method.

Consider the Gauss-Seidel method

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}}{a_{ii}}.$$

Then,

$$r_i^{(k)} = b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}$$

$$x_i^{(k+1)} = x_i^{(k)} + \frac{1}{a_{ii}}r_i^{(k)}.$$

By adding the over relaxation parameter w, we obtain

$$r_i^{(k)} = b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}$$

$$x_i^{(k+1)} = x_i^{(k)} + \frac{w}{a_{ii}}r_i^{(k)}$$

which is the Successive Over-relaxation method. To achieve different results, I tested $w = 0.5, 1, 1.5$.

2.7 Determining the Eigenvalues of Matrix T

I want to show that for a matrix A,

$$A = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \cdots & 0 \\ 0 & -1 & 2 & -1 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix},$$

the eigenvalues are

$$2 - 2\cos\left(\frac{k\pi}{n+1}\right), k = 1, 2, ..., n$$

with corresponding eigenvectors

$$x = \left[\sin\left(\frac{k\pi}{n+1}\right), \sin\left(\frac{2k\pi}{n+1}\right), ..., \sin\left(\frac{nk\pi}{n+1}\right)\right]^T.$$

To do this, consider

$$
\begin{aligned}
&Ax-\lambda x=(A-\lambda I)x\\
&=\left(\begin{bmatrix}2&-1&0&\cdots&0\\-1&2&-1&\cdots&0\\0&-1&2&-1&0\\\vdots&\vdots&\ddots&\ddots&\ddots\\0&0&0&-1&2\end{bmatrix}\right.\\
&\left.-\begin{bmatrix}2-2\text{cos}\left(\frac{k\pi}{n+1}\right)&0&0&\cdots&0\\0&2-2\text{cos}\left(\frac{k\pi}{n+1}\right)&0&\cdots&0\\0&0&2-2\text{cos}\left(\frac{k\pi}{n+1}\right)&0&0\\\vdots&\vdots&\vdots&\ddots&\vdots\\0&0&0&0&2-2\text{cos}\left(\frac{k\pi}{n+1}\right)\end{bmatrix}\right)\begin{bmatrix}\text{sin}\left(\frac{k\pi}{n+1}\right)\\\text{sin}\left(\frac{2k\pi}{n+1}\right)\\\text{sin}\left(\frac{3k\pi}{n+1}\right)\\\vdots\\\text{sin}\left(\frac{nk\pi}{n+1}\right)\end{bmatrix}\\
&=\begin{bmatrix}2\text{cos}\left(\frac{k\pi}{n+1}\right)&-1&0&\cdots&0\\-1&2\text{cos}\left(\frac{k\pi}{n+1}\right)&-1&\cdots&0\\0&-1&2\text{cos}\left(\frac{k\pi}{n+1}\right)&-1&0\\\vdots&\vdots&\ddots&\ddots&\ddots\\0&0&0&-1&2\text{cos}\left(\frac{k\pi}{n+1}\right)\end{bmatrix}\begin{bmatrix}\text{sin}\left(\frac{k\pi}{n+1}\right)\\\text{sin}\left(\frac{2k\pi}{n+1}\right)\\\text{sin}\left(\frac{3k\pi}{n+1}\right)\\\vdots\\\text{sin}\left(\frac{nk\pi}{n+1}\right)\end{bmatrix}\\
&=\begin{bmatrix}2\text{cos}\left(\frac{k\pi}{n+1}\right)\text{sin}\left(\frac{k\pi}{n+1}\right)-\text{sin}\left(\frac{2k\pi}{n+1}\right)\\-\text{sin}\left(\frac{k\pi}{n+1}\right)+2\text{cos}\left(\frac{k\pi}{n+1}\right)\text{sin}\left(\frac{2k\pi}{n+1}\right)-\text{sin}\left(\frac{3k\pi}{n+1}\right)\\-\text{sin}\left(\frac{2k\pi}{n+1}\right)+2\text{cos}\left(\frac{k\pi}{n+1}\right)\text{sin}\left(\frac{3k\pi}{n+1}\right)-\text{sin}\left(\frac{4k\pi}{n+1}\right)\\\vdots\\-\text{sin}\left(\frac{(n-1)k\pi}{n+1}\right)+2\text{cos}\left(\frac{k\pi}{n+1}\right)\text{sin}\left(\frac{nk\pi}{n+1}\right)\end{bmatrix}
\end{aligned}
$$

Using the identities,

$$\text{cos}(\alpha)\text{sin}(\beta)=\frac{1}{2}(\text{sin}(\alpha+\beta)-\text{sin}(\alpha-\beta))$$

$$\text{sin}(-\theta)=-\text{sin}(\theta)$$

we obtain

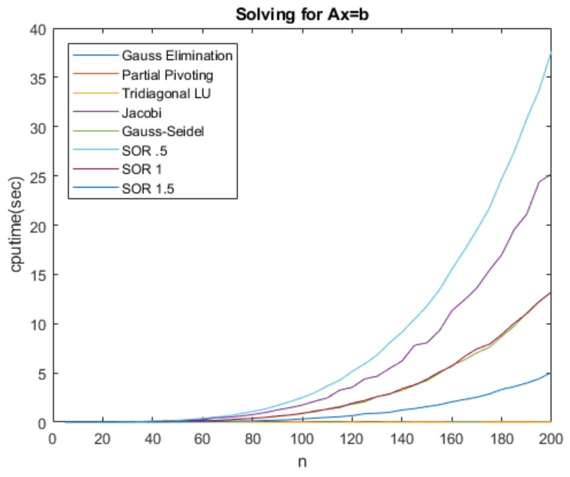
$$\begin{bmatrix}\text{sin}\left(\frac{2k\pi}{n+1}\right)-\text{sin}\left(0\right)-\text{sin}\left(\frac{2k\pi}{n+1}\right)\\-\text{sin}\left(\frac{k\pi}{n+1}\right)+\text{sin}\left(\frac{3k\pi}{n+1}\right)-\text{sin}\left(\frac{-k\pi}{n+1}\right)-\text{sin}\left(\frac{3k\pi}{n+1}\right)\\\vdots\\-\text{sin}\left(\frac{(n-1)k\pi}{n+1}\right)+\text{sin}\left(\frac{(n+1)k\pi}{n+1}\right)-\text{sin}\left(\frac{-(n-1)k\pi}{n+1}\right)\end{bmatrix}=\begin{bmatrix}0\\0\\\vdots\\0\end{bmatrix}$$

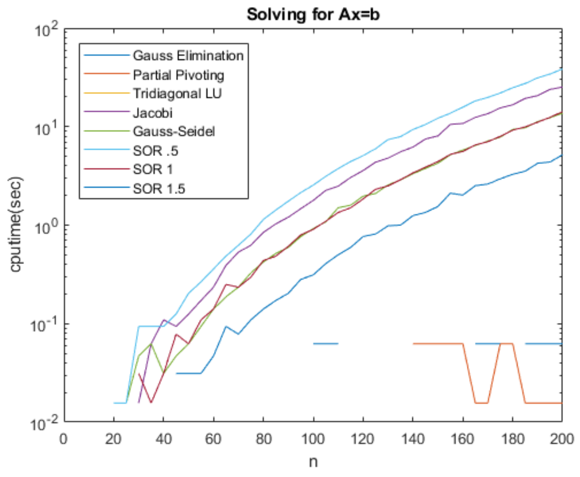
Therefore, $Ax=\lambda x$ so the eigenvalues of A are $\lambda_k=2-2\text{cos}\left(\frac{k\pi}{n+1}\right), k=1,2,...,n$.

3 Results

3.1 Overall Results

All of the above methods used for solving tridiagonal systems of equations were successful in computer the values of x. In order to test the performance of each method, I tested the speed at which each method can compute the value of x with matrices ranging from 0 to 200 in size. Here are the highlighted results.

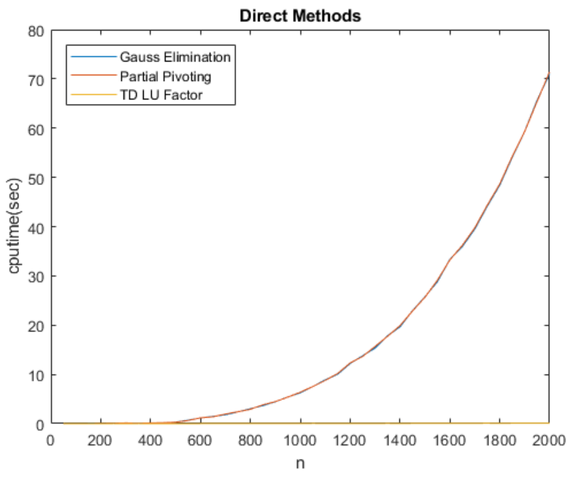


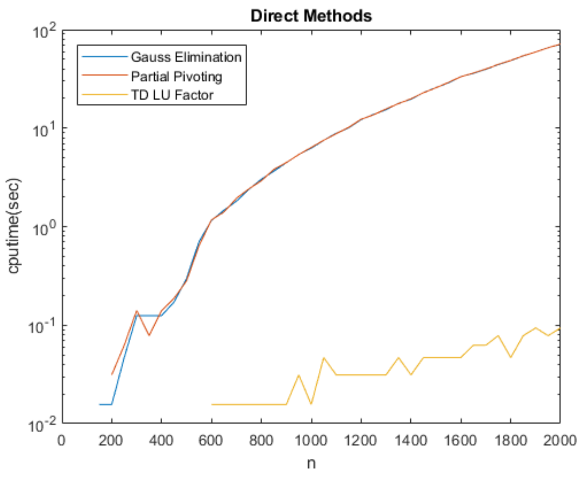


All of the direct methods' cputime are almost negligible compared to those of the indirect methods. We cannot determine which of the direct methods is the fastest on these graphs, but we can determine the slowest methods for solving for x. At the top, the slowest method is the method of "underrelaxation" when $w = .5$. The Jacobi is the next slowest, followed by Gauss-Seidel and SOR when $w = 1$, and finally the fastest iterative method tested was SOR when $w = 1.5$.

3.2 Direct Methods

Since the cputime of matrices of size $n <= 200$ was negligible, I tested the direct methods separately and tested matrix sizes up to $n = 2000$. Here are the results:

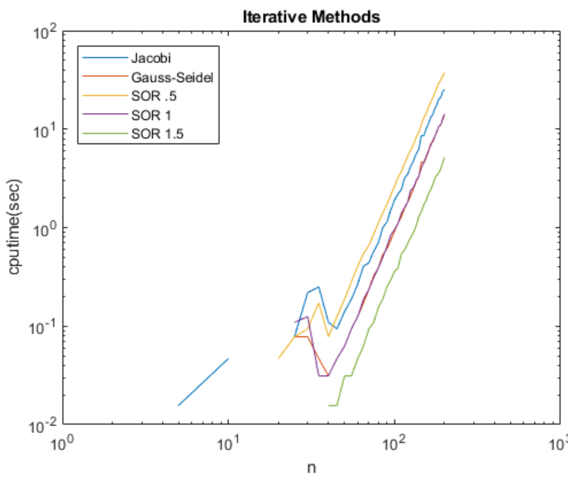


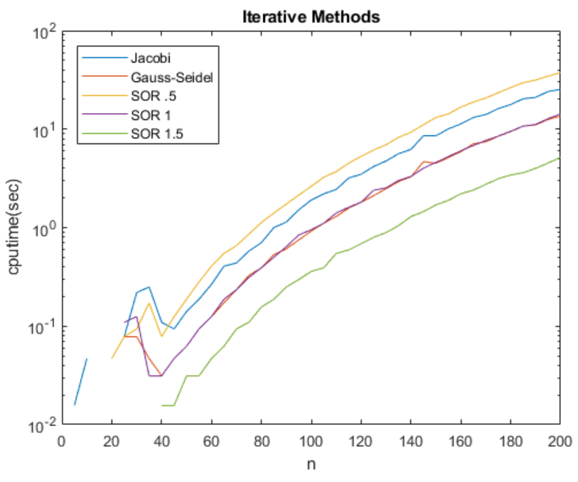


The clear winning for computing tridiagonal matrices is not surprisingly the tridiagonal LU factorization method. This method only uses order n multiplications whereas regular LU factorization is of order n^3. We can also see that Gauss Elimination with and without pivoting are about the same speed. It is important to note that Gauss Elimination without pivoting does not always work, but on the tridiagonal system constructed above, there are never zeros along the diagonal so we would expect not to need pivoting in order to solve the system.

3.3 Iterative Methods

These are the slowest methods for solving for $Ax = b$ on tridiagonal systems. The following are the all of the tested iterative methods:





Starting with the Jacobi, we would expect all of the methods to be faster since they are all improvements on the Jacobi. SOR with $w = .5$ is slower than the Jacobi because it is the underrelaxation version of the Jacobi so we would expect it to be slower. We can see the improvements made by the Gauss-Seidel. At $n = 200$, Gauss-Seidel is about seconds faster than the Jacobi simply by using new x values as they come. SOR with $w = 1$ is the same speed as Gauss-Seidel because they are actually the same exact method of iteration. Lastly, the fastest method of solving for $Ax = b$ using iterations is SOR with $w = 1.5$. This is because this method is the Gauss-Seidel method, but with an overrelaxation of $w = 1.5$ which will obtain the value of x with fewer iterations. For all of these methods, the value of x was obtained such that $\left\|x^k - x^{(k+1)}\right\| < 10^{-8}$.

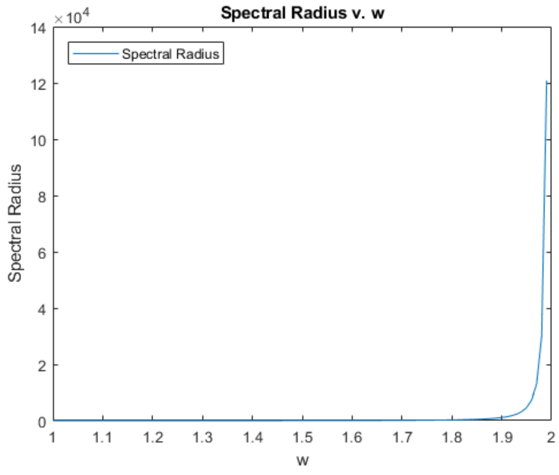
3.4 SOR and Spectral Radius

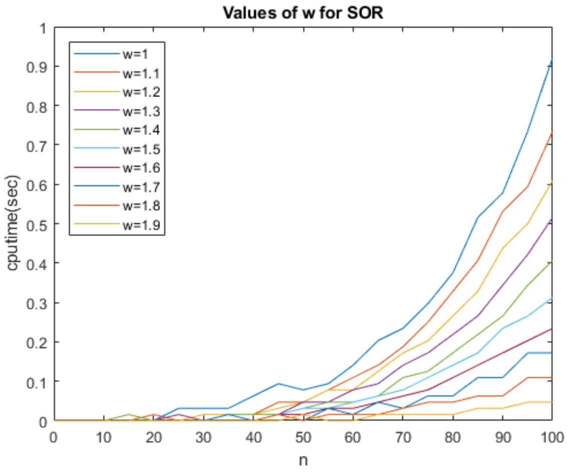
The spectral radius is for a matrix A is defined as

$$\rho(A) = \max_{1 \le i \le n} |\lambda_i(A)|$$

where $\lambda_i(A)$ is an eigen value of A. Since $\lambda_i = 2 - 2\cos\left(\frac{k\pi}{n+1}\right), k = 1, 2, ..., n$ and $-1 \le \cos\left(\frac{k\pi}{n+1}\right) \le 1$, then as $n \to \infty$, $\rho(A) \to 4$.

By graphing $\rho(B_{SOR})$ as a function of w, we can obtain an optimal value for our relaxation parameter.





We can see that when $w \to 2$, $\rho(B_{SOR}) \to \infty$. By graphing various values of w as w approaches 2, we can see the improvements made on cputime. Starting at $w = 1$ which is the Gauss-Seidel method of iterating, we see that we are able to cut down the cputime to a tenth of what it was originally.