Programming Report 3

Executive Summary

In this program, we implement and test various iterative methods for solving matrix systems of the form

$$\mathbf{Ax} = \mathbf{b}.$$

These methods avoid an explicit computation of the inverse or a factorization of A to solve the system. Thus, we approach our solution in an iterative fashion. This is useful in situations where for example the aforementioned computations are inefficient or costly, or sometimes impossible (each when we only observe the action of A, and not the matrix itself). Some of these iterative methods are stationary methods, in the sense that the pre-conditioner is chosen at the beginning of the algorithm and stays identical. Non-stationary methods on the other hand choose a pre-conditioner actively. We implement and test the code on Python. For testing, we thoroughly probe theoretical results concerning each method's reliability and efficiency. For reliability, we test the algorithms on problems where we know theoretical solutions (i.e. convergence) is guaranteed, and also on problems where such results may or may not hold. We control this by perturbing essential properties of our system. For certain algorithms, we reduce our testing to simpler situations which give complete information about our algorithms, and also construct systems ourselves for more thorough testing. For efficiency, we test our algorithms against theoretical guarantees of rates of convergence. Finally, we also implement and test more concrete situations where these algorithms are applied. We present the implementation method, the relevant mathematics, our testing methodology, and complete empirical results in this report.

Objectives

We will follow the task flow of the assignment, and conduct three tasks independently, though with much cross reference for the code implementation. The mathematics of these tasks is described as necessary, since most of this discussion is redundant in our implementation. We take the implementation of these methods from the class notes, and test numerical and theoretical results. Furthermore, for each task, we first present representative problems with the appropriate results and plots, and then we run a larger parameterized set of experiments an present overall statistics from these experiments. The tasks are as follows, and the significance of each we will explain in detail:

1. **Behavior of Richardson's Family, Steepest Descent, and Conjugate Gradient in Eigen-Coordinate System**: We reduce our testing to a simpler form, and implement the methods on Python. We then demonstrate theoretical convergence results, and explain our observations. Finally, we test the problem by varying the dimension of the system, the distribution of our entries, algorithm parameters, and impose structure on the problem, and present and explain our findings with respect to theoretical results.
2. **Behavior of Stationary Methods: Jacobi, Gauss-Seidel, and Symmetric Gauss-Seidel**: We use dense system and associated primitives for these three stationary methods in the Richardson Family of methods, each with it's own preconditioner. We take 8 representative matrices for a particular type of problem, and test our methods on each of these matrices and present our results.
3. **Behavior of All Methods for Large Sparse Symmetric Positive Definite Matrix Problems** We modify the algorithms in Task 2 so they are optimized for Large Sparse SPD Matrix Systems, and run a large parameterized set of experiments and present our results.

You may refer to each of the above descriptions as necessary. Henceforth, we abbreviate as follows: Richardson's Family (RF), Steepest Descent (SD), Conjugate Gradient (CD), Jacobi, Gauss-Seidel (GS), and Symmetric Gauss-Seidel (Symmetric GS).

Note on Implementation

We implement the code on python using a Jupyter notebook. This allows us to use different code blocks for each individual task, and also allows us to conduct each kind of test individually, for each task. We can also conduct interactive testing and display each of these tests separately. The outputs and the tests are displayed after each test/task and can be run independently. We use the NumPy library to construct and use data structures, and various other libraries for error analysis, testing, and visualization (NumPy, SciPy, Pandas, and Matplotlib). We will cite our usage of libraries wherever they have been used. Most relevant plots will be included in this report. Certain parts of the experiments automatically save graphs, tables, and plots in the current directory, and this functionality can be toggled with a Boolean variable. This should be kept in mind when running the source files on your own machine!

Task 1

Preliminaries (Mathematical Background)

We implement three algorithms: the simplest stationary method, i.e. Richardson's method without preconditioning, and two non-stationary methods, Steepest Descent and Conjugate Gradient. We restrict ourselves to symmetric positive definite systems for these tasks since this ensures convergence for each method, and in fact the non-stationary methods require this restriction for derivation and error analysis. Richardson's family of methods work also for non-SPD systems and there is a vast literature regarding this usage, but this is beyond the scope of this report. For these three methods, we can simply considerably to the eigen-basis using the following definition and theorem:
Definition: A matrix $B \in \mathbb{C}^{n\times n}$ is called **diagonalizable** if it has n linearly independent eigenvectors, x_i, $1 \le i \le n$, associated with eigenvalues λ_i, $1 \le i \le n$. Then we can show that:

- $Bx_i = x_i\lambda_i, \quad 1 \le i \le n$ in matrix form

$$X = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}, \quad \Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$$

$$BX = X\Lambda \implies B = X\Lambda X^{-1} \quad \text{and} \quad X^{-1}BX = \Lambda$$

- A symmetric matrix is diagonalizable with $Q = X$ and $Q^T = X^{-1}$.

For symmetric positive definite matrices, we additionally get that the matrix Λ of the eigenvalues of A is real and positive, and the columns of the matrix Q are the n orthonormal eigenvectors of A. This then leads to the following observation:

$$\begin{aligned} Ax &= b \\ (Q\Lambda Q^T)x &= b \\ \Lambda\tilde{x} &= \tilde{b} \end{aligned}$$

where $\tilde{x} = Q^T x$ and $\tilde{b} = Q^T x$. Thus we simply get a change of basis of the system with the full-rank matrix Q^T. Thus each such system with $\Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$ and $\lambda_i > 0$ is determined by the original SPD system. We can then restrict our analysis to such systems. Note that simplifies our storage and computations considerably since a matrix-vector product with a diagonal matrix is the Hadamard product, which is implemented as element-wise multiplication of two vectors.

For error analysis, we will restate the following theorems:

Theorem
If $A \in \mathbb{R}^{n\times n}$ is symmetric positive definite and $b \in \mathbb{R}^n$ is given, then the sequence $\{x_k\}$, from the stationary iterative method

$$x_{k+1} = x_k + \alpha_{\text{opt}} r_k, \quad \alpha_{\text{opt}} = \frac{2}{\lambda_{\min} + \lambda_{\max}}$$

satisfies

$$\|e^{(k+1)}\|_A \le \frac{\kappa - 1}{\kappa + 1}\|e^{(k)}\|_A$$

where $e^{(k)} = x_k - A^{-1}b$ and $\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$.
Theorem
If $A \in \mathbb{R}^{n\times n}$ is symmetric positive definite and $b \in \mathbb{R}^n$ is given, then the sequence $\{x_k\}$, from Steepest Descent

$$x_{k+1} = x_k + \alpha_k r_k, \quad \alpha_k = \frac{r_k^T r_k}{r_k^T A r_k},$$

satisfies

$$\|e^{(k+1)}\|_A \leq \frac{\kappa - 1}{\kappa + 1}\|e^{(k)}\|_A$$

where $e^{(k)} = x_k - A^{-1}b$ and $\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$.

Theorem 9.8

For CG, $e^{(k)}$ is bounded in terms of κ, the condition number of A, and the initial error $e^{(0)}$ by

$$\|e^{(k)}\|_A \leq 2\alpha^k \|e^{(0)}\|_A$$

$$\alpha = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}.$$

Note that these results are asymptotic, especially in practice. Also note that for CG, the error damping bound goes down with k, i.e. the number of steps taken in the algorithm.

Implementation

The workspace for the algorithm is $3n = O(n)$. This is because we start with a solution x^* and a positive diagonal matrix represented as a positive vector D and compute their (Hadamard/element-wise) product to generate b for our system. We do this so we can compute actual errors for our analysis. Our algorithm only requires D, b and an initial guess x_0 as input, and $x*$ is used only to compute error, and for an initial interactive testing part where we set our convergence condition to be $x_k = x^*$, i.e. the exact solution is the output, or we can also require the exact error to be close enough for our answer to be correct up to some decimal places (this avoids numerical bottlenecks). For our experimental convergence criterion, we use $\frac{\|r_k\|}{\|r_0\|} < tol$ where tol is a pre-set tolerance for the residual (the value of which we will probe during the testing and choose proactively) and r_k is the residual after k iterations.

Our algorithm returns the following outputs (see source code):

1. The number of iterations = num
2. The final x output = $x_{k=num}$
3. An array of errors
4. An array of residuals

For our interactive tests, we present first the defining information about the problem in a particular experiment. This includes, but is not limited to the following (see source code):

- The size of the system = n
- The spectrum of the matrix = $spec(A)$
- The spectral radius = $\rho(A)$
- The condition number = $\kappa(A)$
- The initial guess = x_0
- The true solution of the system = x*(this is known apriori)
- The tolerance used = tol variable (as discussed above).

We set a maximum number of iterations for our convergence, and assign a convergence Boolean variable according to if the number of iterations were less than the maximum iterations allowed i.e convergence=TRUE if $num < MAXITER$ for each algorithm. We compute the condition number κ as $\kappa(A) = \lambda_{max}/\lambda_{min}$, where the eigenvalues are exactly the elements of our vector D. In the algorithm, to ensure convergence, we have a step-size variable α in Richardson and Steepest descent. It can derived that there is a bound on α to ensure convergence, and it also has an optimal value, $\alpha_{opt} = \frac{2}{\lambda_{min}+\lambda_{max}}$. This derivation is done in the lecture notes. Thus, these variables are simple enough to compute.

To allow for a uniform comparison of the three algorithms, we use a uniform testing criteria, i.e the tolerance, MAXITER, and problem variables etc. are kept the same for each algorithm. We also provide a randomizer for each input variable for general experimentation. We then present a table of results for our algorithms, which includes the following results:

1. If convergence occurred (True/False)
2. Number of Iterations (num)
3. Initial Error (float)
4. Final Error (float)
5. Final Residual (float)
6. Total Work (Total Cost of the Algorithm)

In the tables presented in this report, we show all floating points in scientific format for brevity. The total cost of each run is computed using formulas derived by hand and implemented as functions in Python. Each algorithm has a one-time cost and an iterative cost associated to it. These can be calculated using Table 1 and going over each algorithm in the Python file.

Computation	**Complexity**
Vector Addition	n
Hadamard Product	n
Inner Product	$2n-1$
Scalar Multiplication	n

Table 1: Computation Cost (n = dimension/size)

Keep in mind each arithmetic operations adds 1 to the complexity. We ignore costs that are uniform across our algorithms e.g. the cost of increasing the iteration counter variable by 1 for each iteration the loop, or the cost of checking the convergence criterion (in general these may be costly and ill-conditioned operations e.g. computation of the 2-norm of large vectors, followed by one division and one comparison). We can then compute the one-time and iterative costs of each algorithm and use the following formula to compete the total work of algorithm A:

$$\text{Total Work}_A = \text{One-time Cost} + num_A \times (\text{Iterative Cost})$$

where num_A is the total number of iterations for Algorithm A. The respective costs are reproduced in Table 2 below.

Algorithm	**One-time Cost**	**Iterative Cost**
RF	$2n+2$	$5n$
SD	$3n$	$9n-1$
CG	$4n-1$	$11n$

Table 2: Algorithm Costs

We then present plots for error and convergence analysis. This includes a gird of plots where each row is for a particular algorithm and column 1 is plots of error against number of steps and column 2 is plots of residual against number of steps. These provide a visual representation of convergence - the last error and residual can be cross-referenced from the table of results to ensure convergence occurred. The shapes of these curves also provide insight to the convergence rate of each algorithm and the actual efficiency can then be deduced from the number of iterations in the table of results, as well as the total cost associated with the algorithm. The notes and lectures (Sets 7, 8, 9, and 10; Study Question Sets 4 and 5; and written problems from Homework 3) have discussed and derived the bounds on the behavior of the damping of the error and residuals. For RF and SD each step must reduce the error norm by at the factor

$$\frac{\kappa-1}{\kappa+1}$$

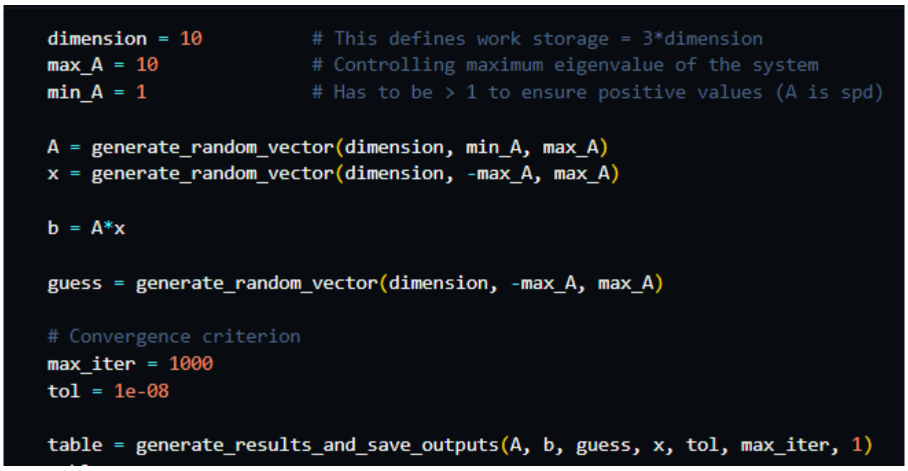
with SD's single step reduction being no less and typically better than RF. CG's one step damping factor is

$$\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}.$$

For CG, the performance, in exact arithmetic, is also related to the number of distinct eigenvalues, e.g., the identity matrix as a single eigenvalue, 1, with multiplicity n, as well as the eccentricity of the spectrum as measured by κ. We also present the graphs for the damping factors of each run for each algorithm and also plot the theoretical upper-bounds for each.

Experimental Design

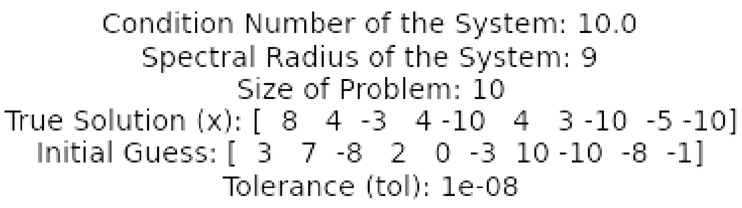
We conduct interactive experiments and discuss the results. We can also treat this subsection as sample results presentation. We initially restrict ourselves to simple random problems. For our problem generation, we compute a random integer-valued positive vector D and arbitrary integer valued vector x using NumPy functionality, and then compute their product to generate b. We also compute a random initial guess vector. We then manually set convergence control and run our algorithms. The code for the problem generation looks as follows:



```
dimension = 10          # This defines work storage = 3*dimension
max_A = 10              # Controlling maximum eigenvalue of the system
min_A = 1               # Has to be > 1 to ensure positive values (A is spd)

A = generate_random_vector(dimension, min_A, max_A)
x = generate_random_vector(dimension, -max_A, max_A)

b = A*x

guess = generate_random_vector(dimension, -max_A, max_A)

# Convergence criterion
max_iter = 1000
tol = 1e-08

table = generate_results_and_save_outputs(A, b, guess, x, tol, max_iter, 1)
```

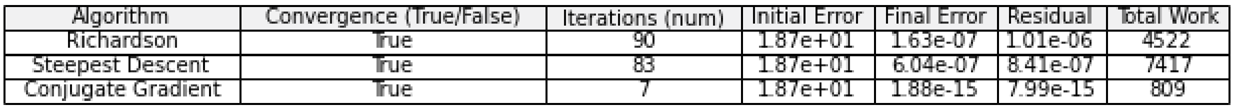
Figure 1: Random Problem Generator

The characteristics and results for this **uniform random experiment** look like:



```
Condition Number of the System: 10.0
Spectral Radius of the System: 9
Size of Problem: 10
True Solution (x): [ 8   4  -3   4 -10   4   3 -10  -5 -10]
Initial Guess: [ 3   7  -8   2   0  -3  10 -10  -8  -1]
Tolerance (tol): 1e-08
```

Figure 2: Problem Characteristics



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Residual	Total Work
Richardson	True	90	1.87e+01	1.63e-07	1.01e-06	4522
Steepest Descent	True	83	1.87e+01	6.04e-07	8.41e-07	7417
Conjugate Gradient	True	7	1.87e+01	1.88e-15	7.99e-15	809

Figure 3: Results Table

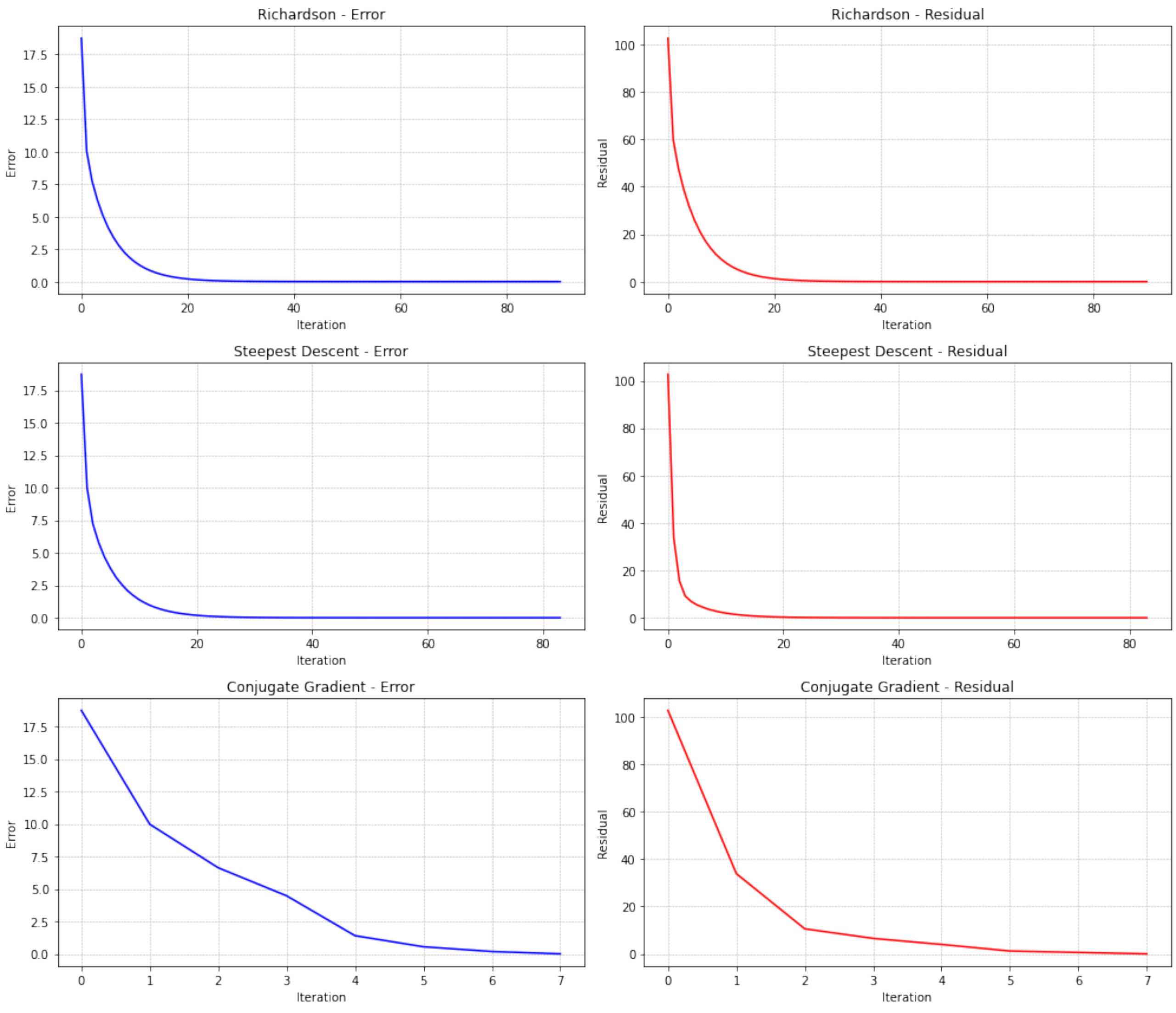


Figure 4: Error and Residual Convergence

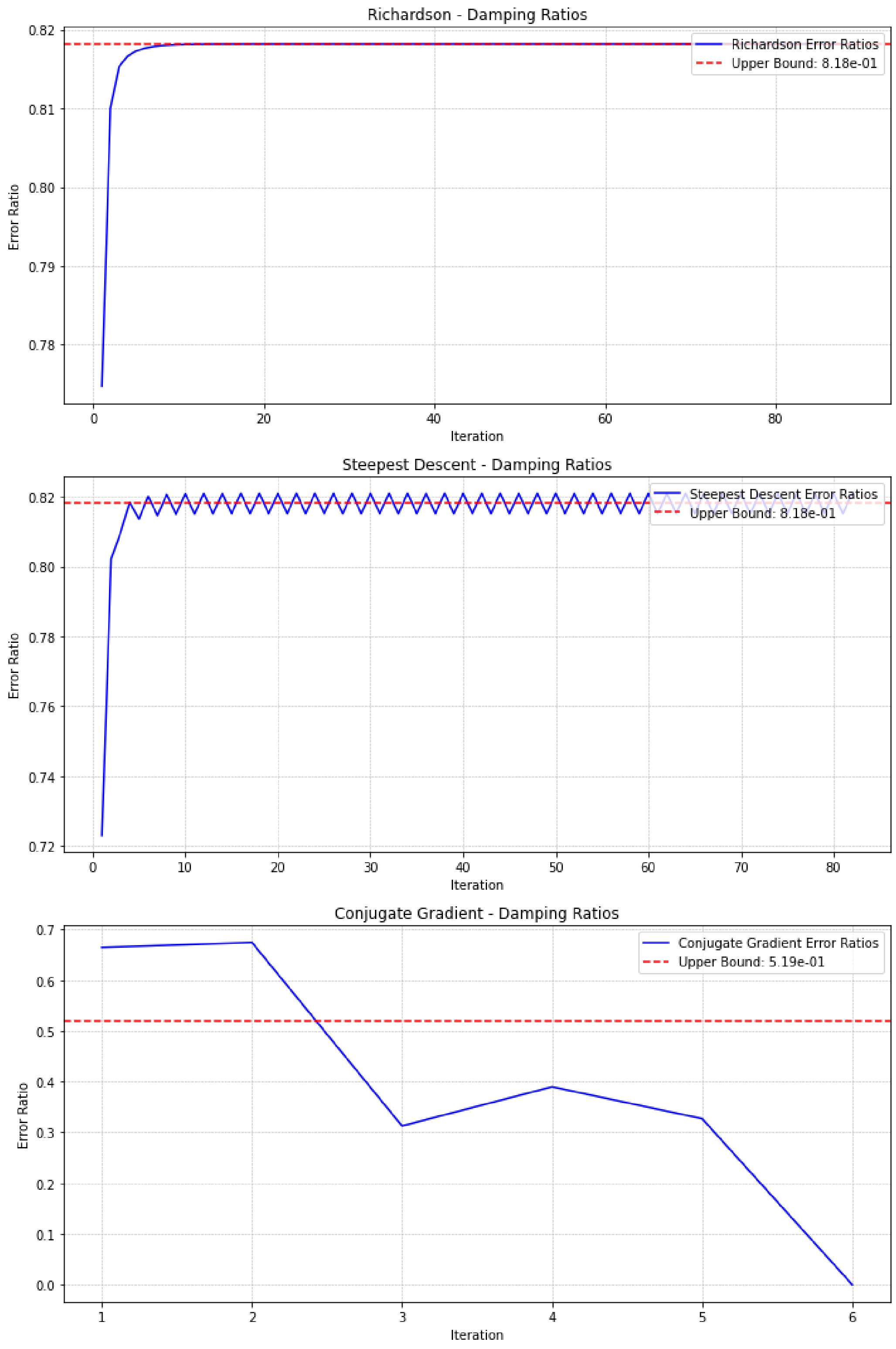


Figure 5: Damping Factors

I chose this as a sample experiment as it shows general conclusions and trends that we will observe in the larger parameterized set of experiments. This is a small and simple enough random system of integer values and a small spectrum (as controlled by the size of the vector D). We can present another such 'typical' experiment where we avoid numerical pitfalls, and can easily observe theoretical predictions.

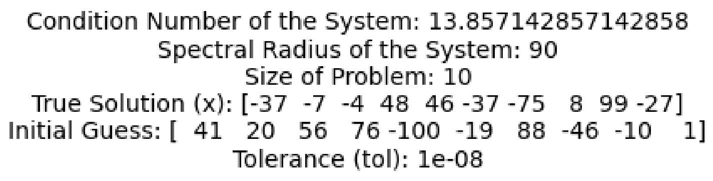
Under these conditions, each algorithm behaves as expected by theoretical predictions, and we can make the following hypotheses (these are not presented as actual results at this point):

Hypotheses:

1. The algorithms converge super-linearly (or may even have quadratic convergence).
2. SD outperforms RF in number of iterations generally, but is worse than RF in terms of total work done and accuracy.
3. RF and SD have comparable performance, while CG vastly outperforms both RD and SD, in terms of both total cost, number of iterations and accuracy.
4. The error and residuals for each algorithm show similar (relative) behavior.
5. Theoretical bounds and results on damping ratios hold up to numerical pitfalls and finite-precision modalities.
6. The dimension of the system, the sizes and ranges of values used for each problem variable, the spectrum of the system, and the tolerance and maximum number of iterations allowed are all dependent variables for performance of each algorithm.

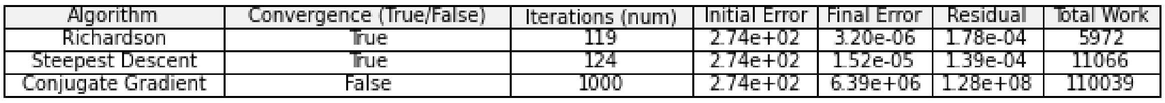
1 through 5 can be tested for more confidence within this section, and will also be tested by proxy when we conduct experiments for the last hypothesis. 6 will also be tested thoroughly for each variable, besides for the last two convergence variables, since this result will follow immediately from testing done within this section (it is also apparent from the design of any iterative algorithm). Let us now present results for different problems!

For a **larger-valued random experiment** (values $\in [-100, 100]$) of the same dimension, we get:



```
Condition Number of the System: 13.857142857142858
Spectral Radius of the System: 90
Size of Problem: 10
True Solution (x): [-37  -7  -4  48  46 -37 -75   8  99 -27]
Initial Guess: [  41   20   56   76 -100  -19   88  -46  -10    1]
Tolerance (tol): 1e-08
```

Figure 6: Problem Characteristics



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Residual	Total Work
Richardson	True	119	2.74e+02	3.20e-06	1.78e-04	5972
Steepest Descent	True	124	2.74e+02	1.52e-05	1.39e-04	11066
Conjugate Gradient	False	1000	2.74e+02	6.39e+06	1.28e+08	110039

Figure 7: Results Table

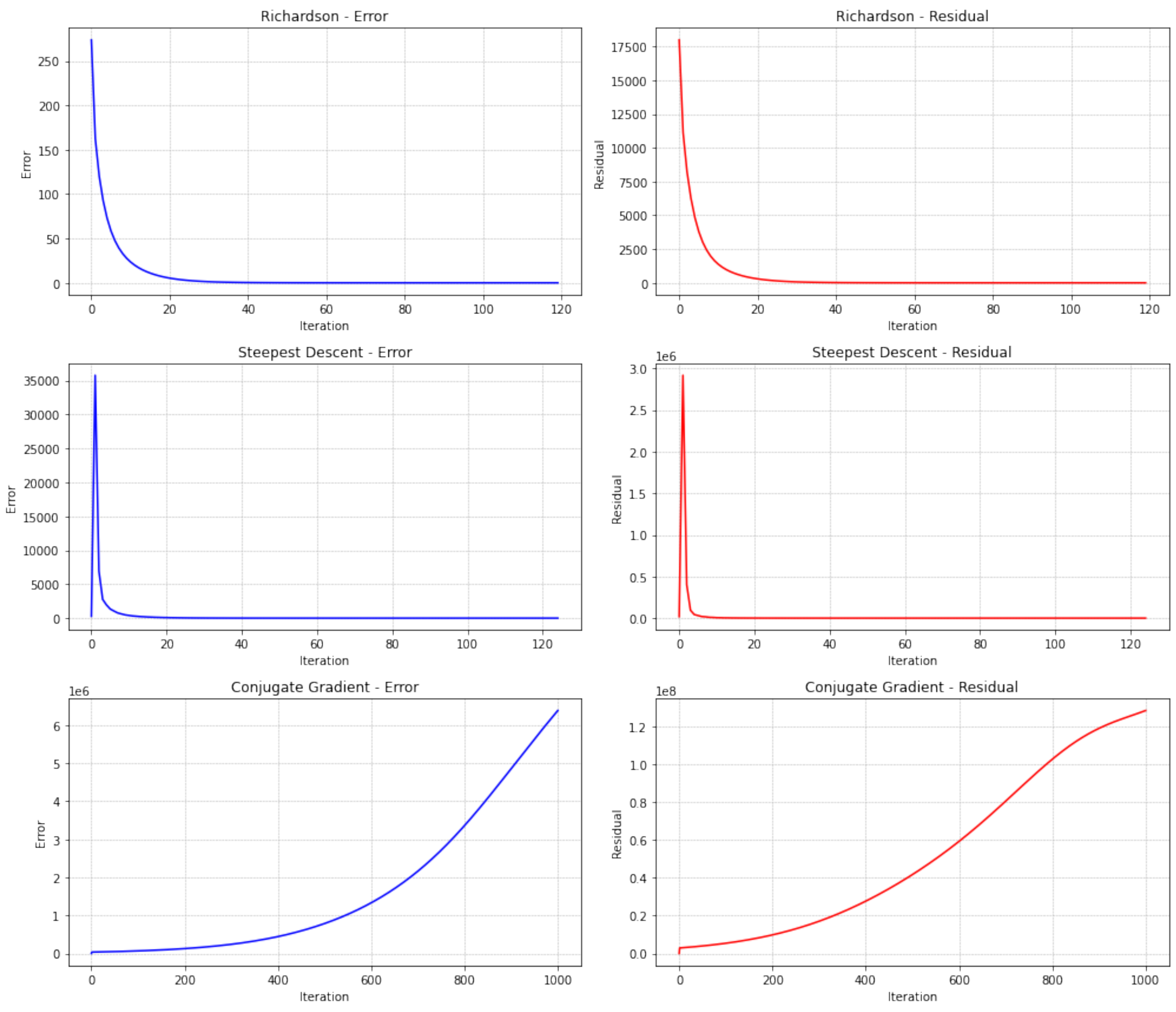


Figure 8: Error and Residual Convergence

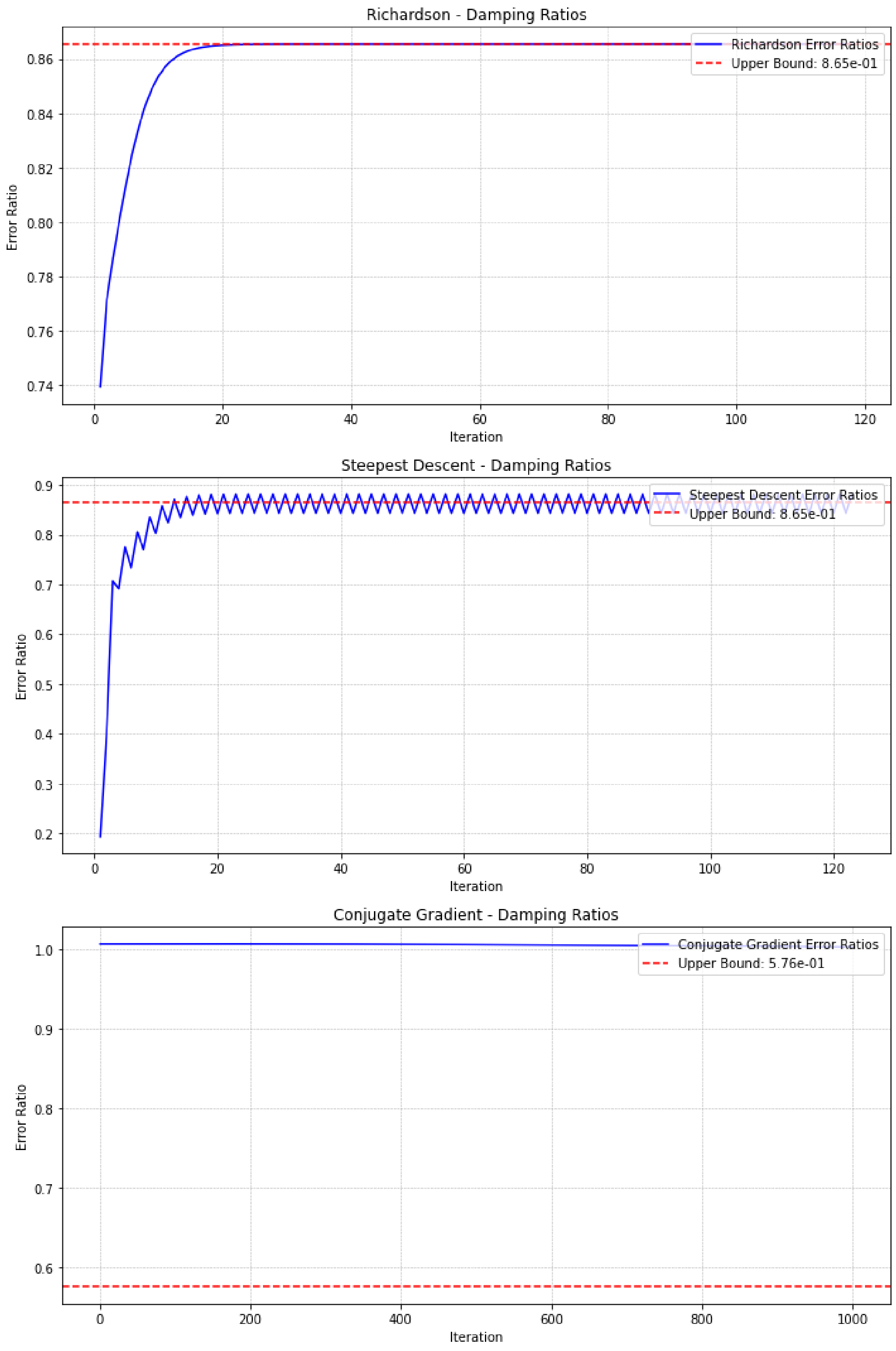
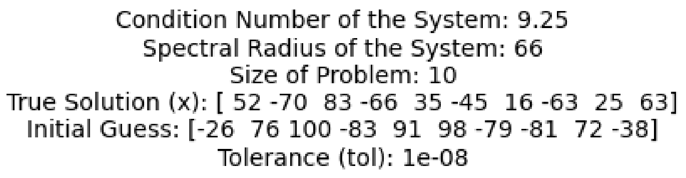


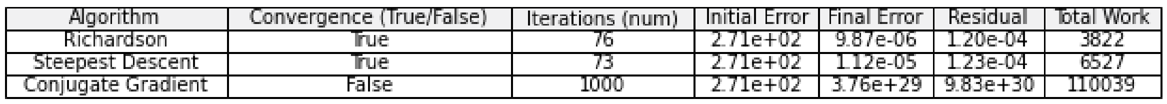
Figure 9: Damping Factors

Here we can see similar results for RF and SD, but CG is observed to diverge. Numerically, we can find malicious systems where CG can diverge, and it is also sensitive to perturbation. In this case, Richardson was the best algorithm in terms of accuracy as well as number of iterations and total work done, outperforming Steepest Descent. This weakens hypothesis number 2 and in fact, we observe the converse result! Let us run another such random experiment:



```
Condition Number of the System: 9.25
Spectral Radius of the System: 66
Size of Problem: 10
True Solution (x): [ 52 -70  83 -66  35 -45  16 -63  25  63]
Initial Guess: [-26  76 100 -83  91  98 -79 -81  72 -38]
Tolerance (tol): 1e-08
```

Figure 10: Problem Characteristics



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Residual	Total Work
Richardson	True	76	2.71e+02	9.87e-06	1.20e-04	3822
Steepest Descent	True	73	2.71e+02	1.12e-05	1.23e-04	6527
Conjugate Gradient	False	1000	2.71e+02	3.76e+29	9.83e+30	110039

Figure 11: Results Table

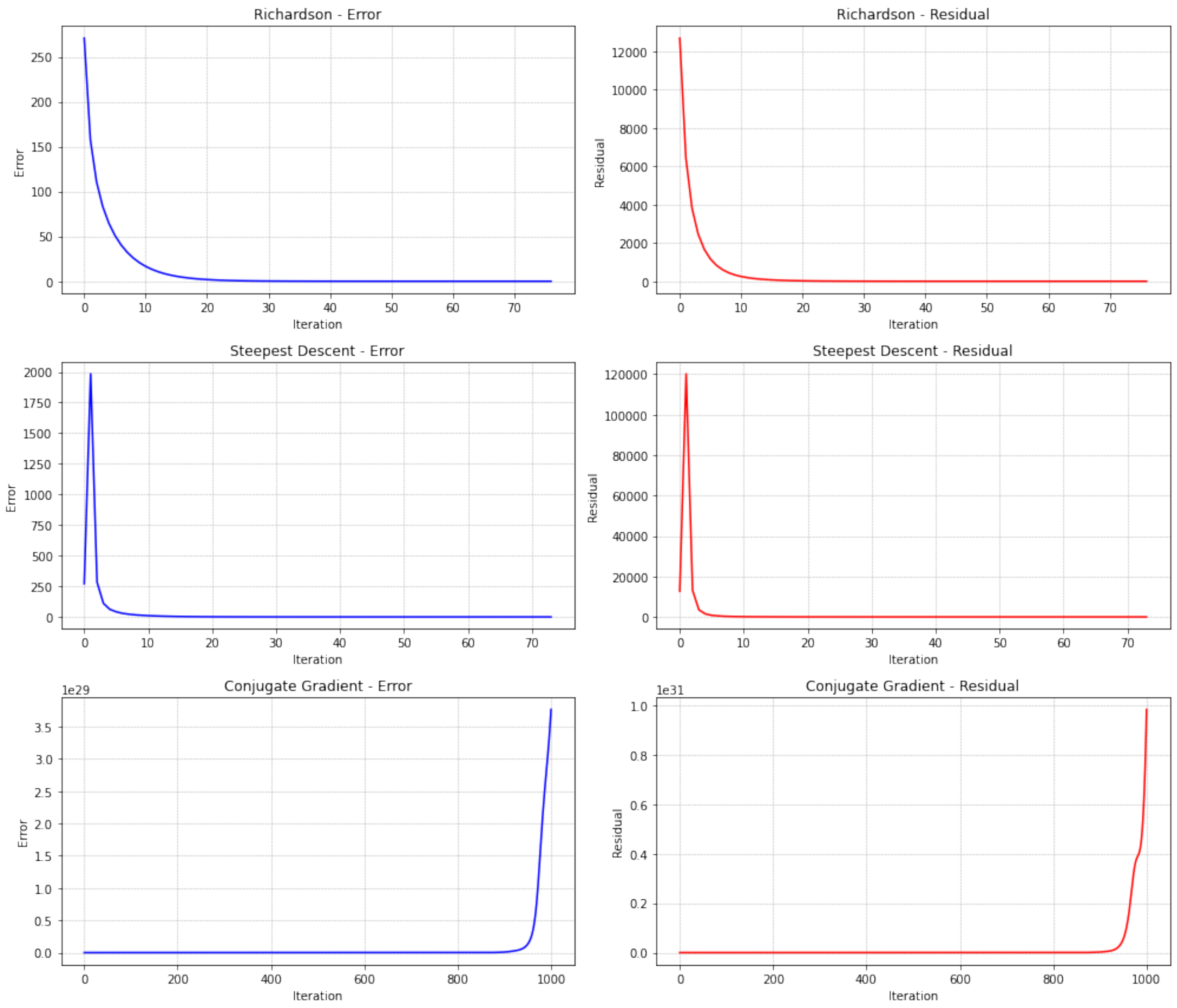


Figure 12: Error and Residual Convergence

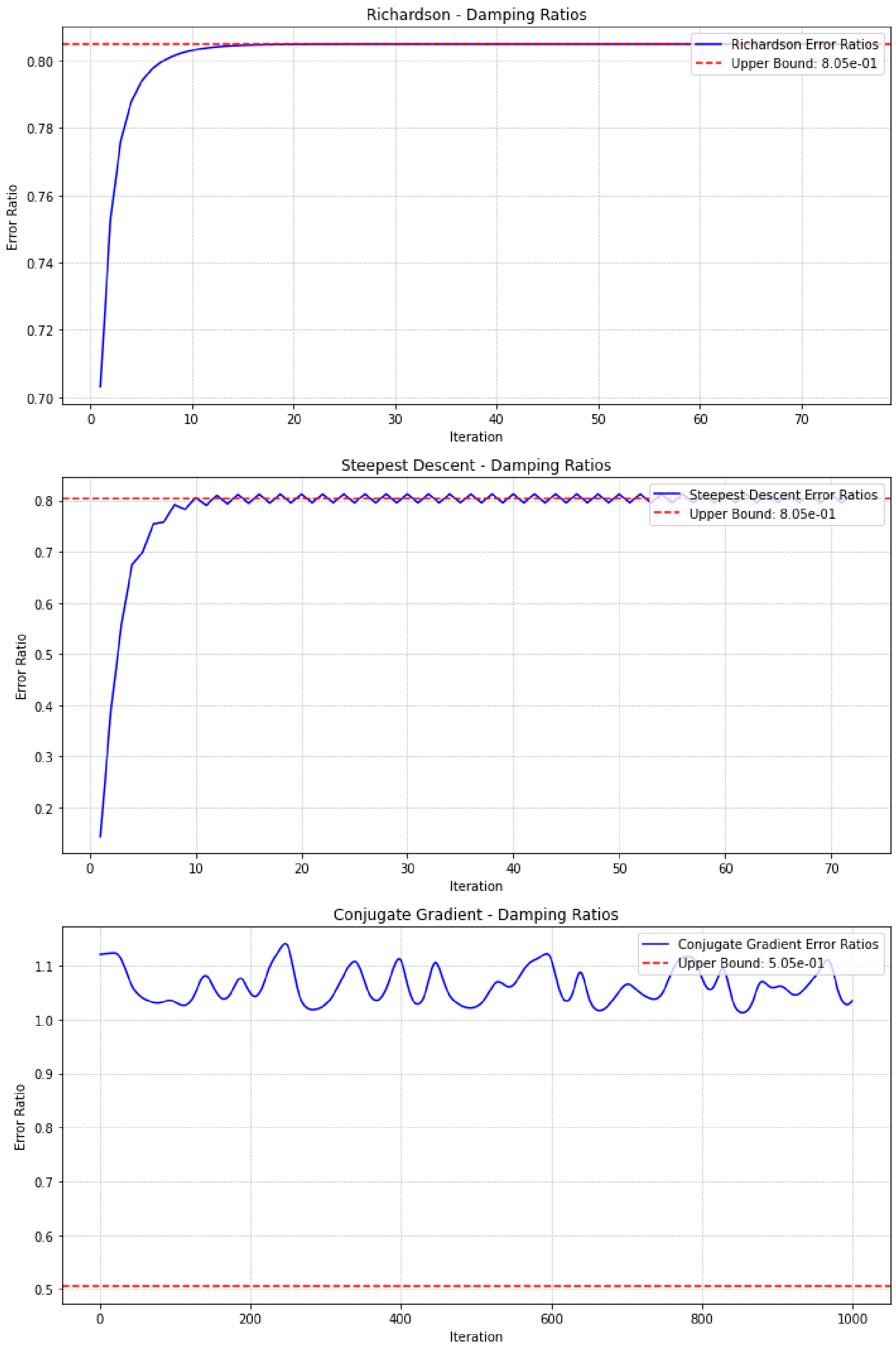
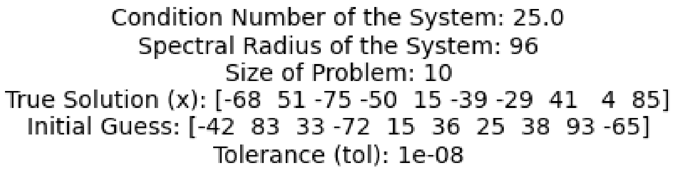


Figure 13: Damping Factors

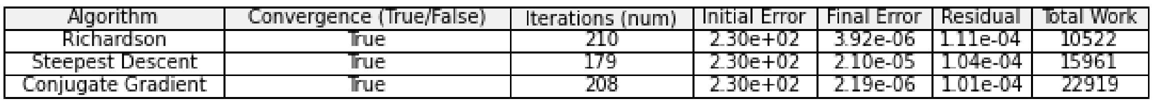
We have similar observations as with the first larger-valued problem. We also make more observations: The damping factors for RF and SD approach their theoretical bounds very quickly and then becomes constant. RF also begins with much smaller damping than SD i.e. in the beginning SD has stronger damping of the error for a brief period (about 7-8 iterations). SD also shows a curious 'saw-tooth' behavior around the bound oscillating above and below it by a small amount quite uniformly.

Furthermore, SD also shows a uncharacteristic and big jump in error at the very first or first few iterations and then shows similar convergence as we expect. CG in this case diverged again, but had an exponential blow up in error at the end of the maximum number of iterations (1000 in this case) allowed. The error ratio also showed a relatively constant behavior around 1.1. SD also outperformed RF in number of iterations by very little, but RF outperformed SD in terms of both accuracy and total work by relatively significant amounts. This again supports the converse result for this particular hypothesis. The larger-valued experiments show even more curious results. For another one of these experiments were CG does appear to converge, we get the following results:



Condition Number of the System: 25.0
Spectral Radius of the System: 96
Size of Problem: 10
True Solution (x): [-68 51 -75 -50 15 -39 -29 41 4 85]
Initial Guess: [-42 83 33 -72 15 36 25 38 93 -65]
Tolerance (tol): 1e-08

Figure 14: Problem Characteristics



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Residual	Total Work
Richardson	True	210	2.30e+02	3.92e-06	1.11e-04	10522
Steepest Descent	True	179	2.30e+02	2.10e-05	1.04e-04	15961
Conjugate Gradient	True	208	2.30e+02	2.19e-06	1.01e-04	22919

Figure 15: Results Table

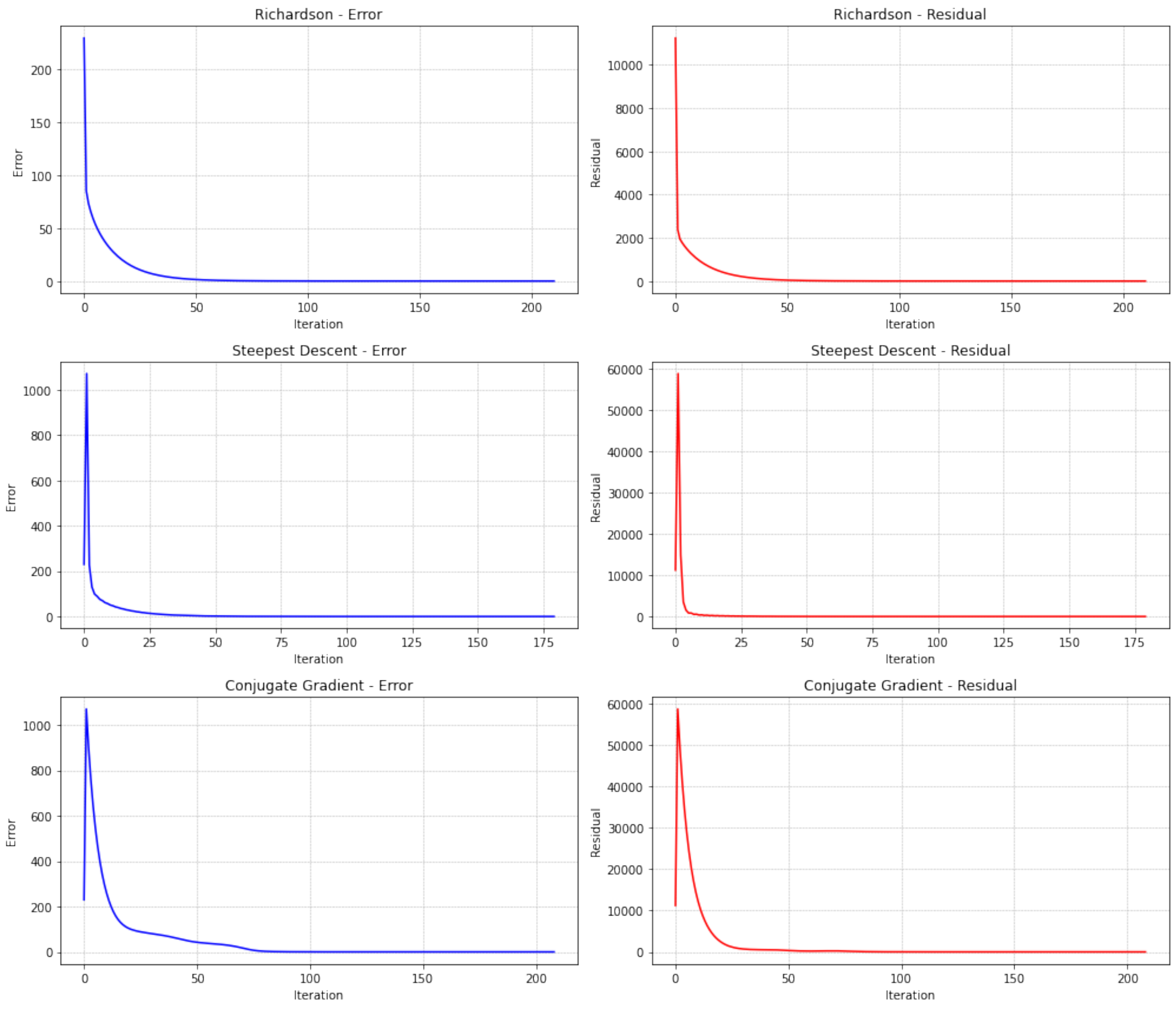


Figure 16: Error and Residual Convergence

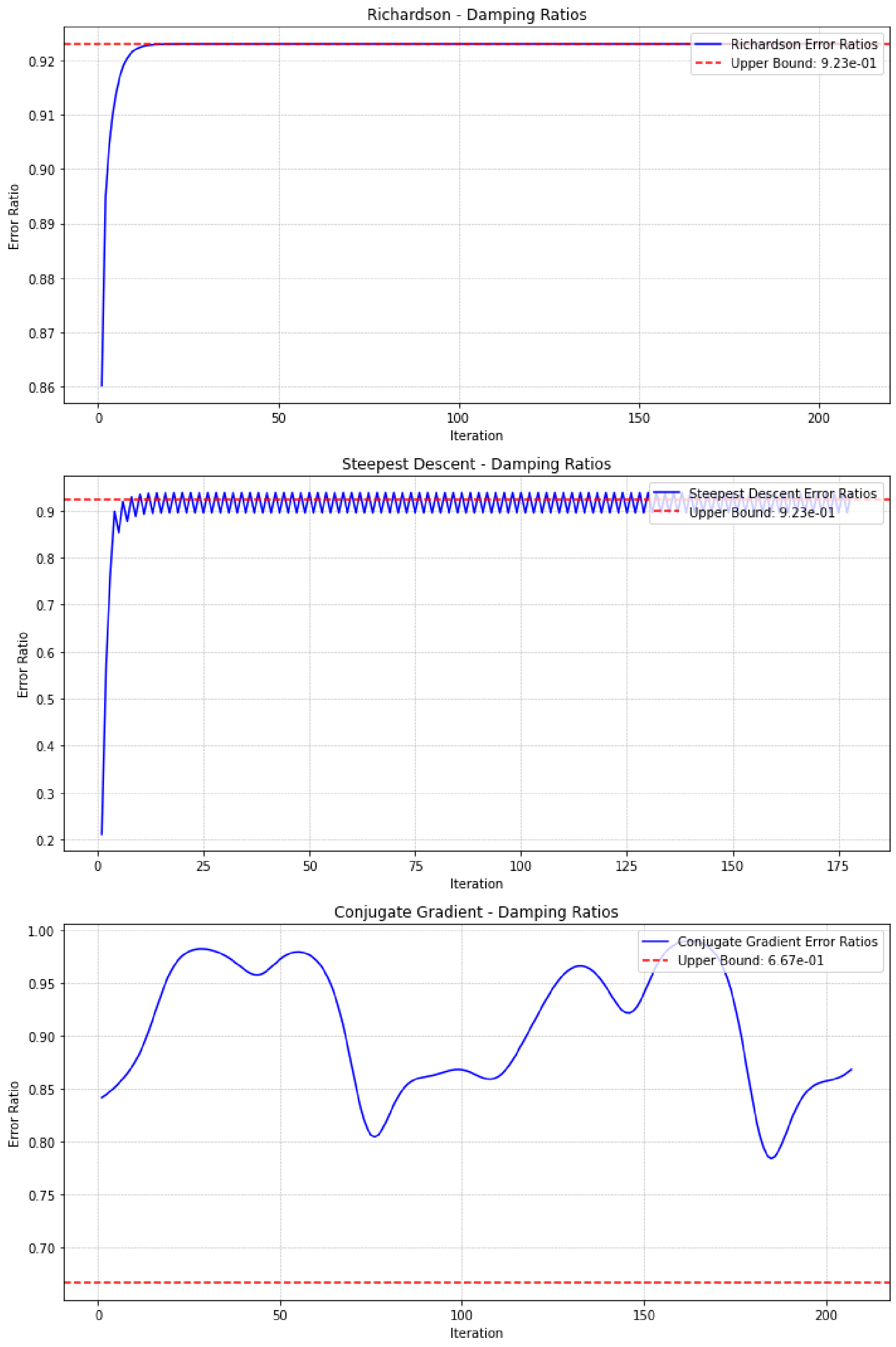
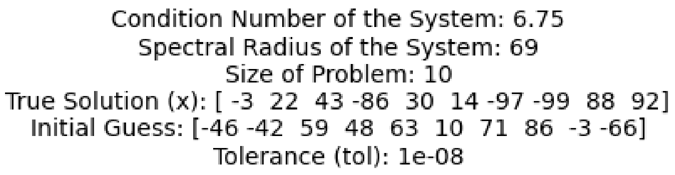


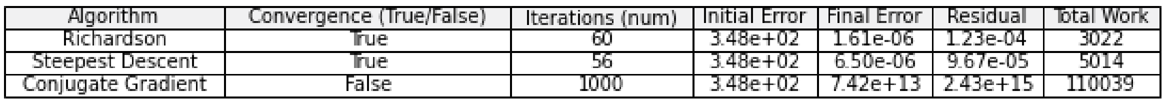
Figure 17: Damping Factors

Note the much larger condition number for the system (about twice of the first two of such experiments). Here, CG shows similar a similar error profile to SD, but rather than converging very quickly, or diverging, it instead converges at about the rate of RF and SD. CG however, is still more accurate but takes almost twice the work. RF continues to outperform SD (except in number of iterations, though it is close). The damping for CG showed a similar periodic/oscillating structure, but this time it was bounded by 1, resulting in convergence. However, the damping vastly exceeded the theoretical bound it should have attained, in fact never once going below or close to the bound. There does seem to be lower and upper bounds for the damping factor in both experiments (divergent and convergent). We generate one more experiment for completeness:



```
Condition Number of the System: 6.75
Spectral Radius of the System: 69
Size of Problem: 10
True Solution (x): [ -3  22  43 -86  30  14 -97 -99  88  92]
Initial Guess: [-46 -42  59  48  63  10  71  86  -3 -66]
Tolerance (tol): 1e-08
```

Figure 18: Problem Characteristics



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Residual	Total Work
Richardson	True	60	3.48e+02	1.61e-06	1.23e-04	3022
Steepest Descent	True	56	3.48e+02	6.50e-06	9.67e-05	5014
Conjugate Gradient	False	1000	3.48e+02	7.42e+13	2.43e+15	110039

Figure 19: Results Table

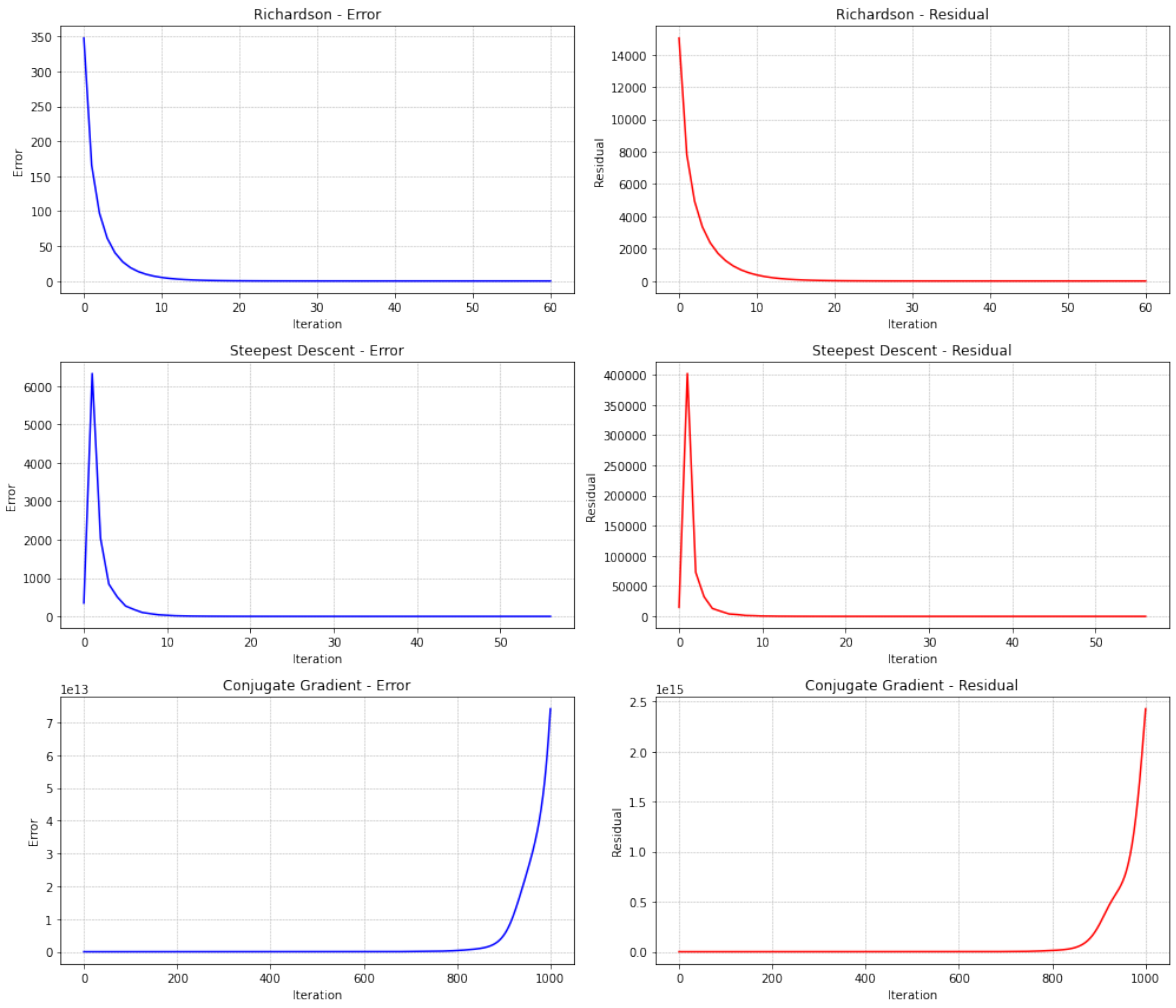


Figure 20: Error and Residual Convergence

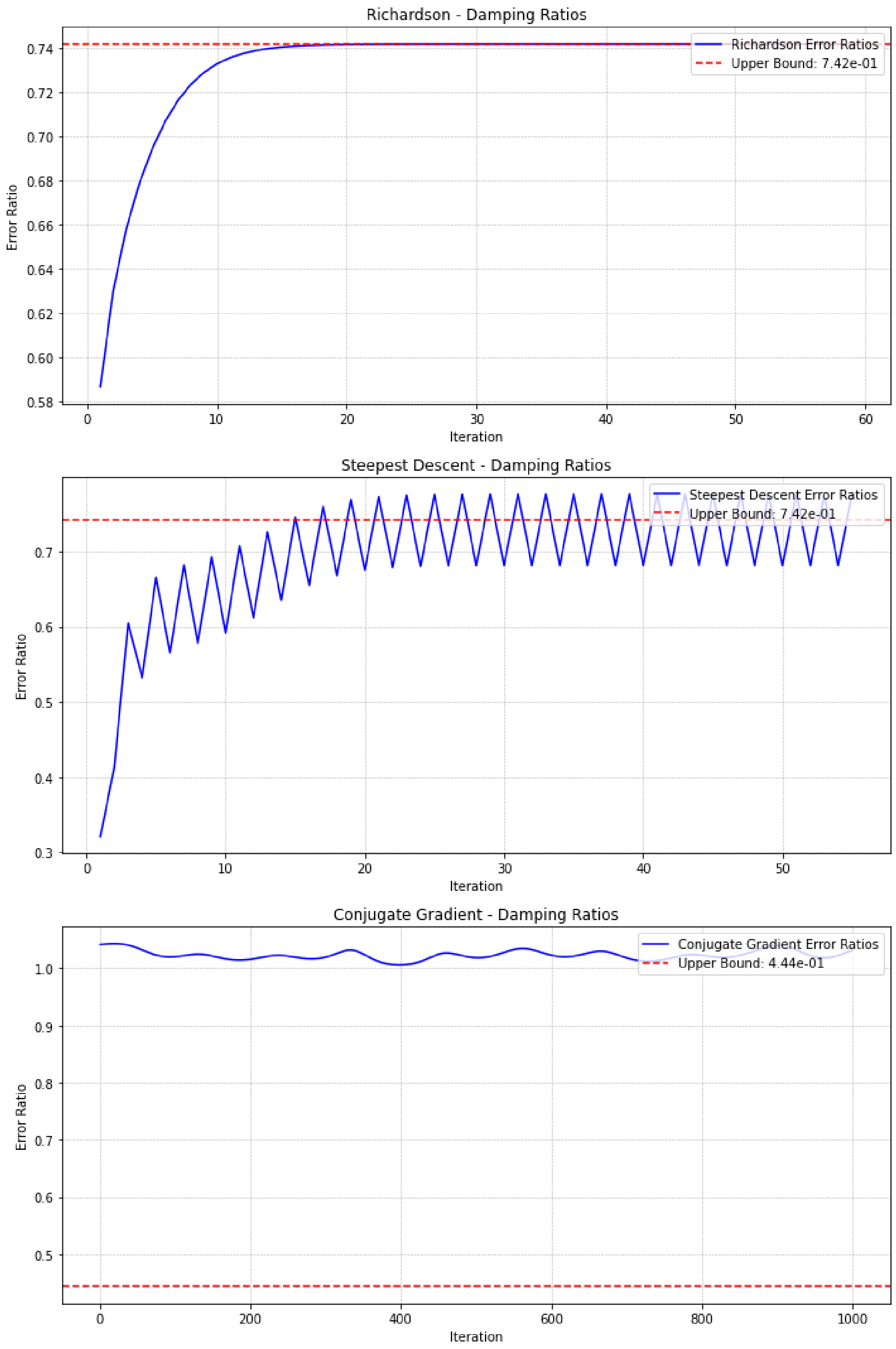


Figure 21: Damping Factors

Important Conclusion 1:
By running more experiments, we can conclude that typically for such large-valued random experiments, CG diverges, even when the condition number is smaller. One explanation for this is the higher amount of ill-conditioned operations that CG must perform. These are numerically very sensitive, and can cause perturbation of the system. The conjugate gradient method can theoretically be viewed as a direct method, as in the absence of round-off error it produces the exact solution after a finite number of iterations, which is not larger than the size of the matrix. In practice, the exact solution is never obtained since the conjugate gradient method is unstable with respect to even small perturbations, e.g., most directions are not in practice conjugate, due to a degenerative nature of generating the Krylov subspaces. Also note that larger values results in a larger value of initial error as well, as we can see in the upcoming control experiment.

We now restrict ourselves to larger-sized systems, i.e. problems with **larger dimension**. Using this approach, we can slowly build towards simulating complicated problems from practical applications. We first run experiments where the dimension is large (e.g. 100) but the values are relatively smaller still (in $[-10, 10]$). Note this has the effect of controlling the largest possible condition number too, i.e for this experiment $\kappa(A) \leq 10$. Also note the smaller value of initial error in this experiment. These experiments can now serve as a control for our results in the previous experiment:

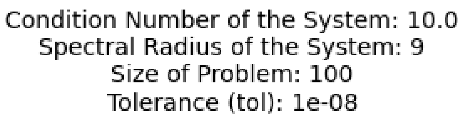
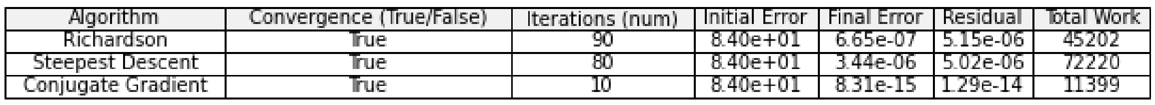

Condition Number of the System: 10.0
Spectral Radius of the System: 9
Size of Problem: 100
Tolerance (tol): 1e-08

Figure 22: Problem Characteristics



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Residual	Total Work
Richardson	True	90	8.40e+01	6.65e-07	5.15e-06	45202
Steepest Descent	True	80	8.40e+01	3.44e-06	5.02e-06	72220
Conjugate Gradient	True	10	8.40e+01	8.31e-15	1.29e-14	11399

Figure 23: Results Table

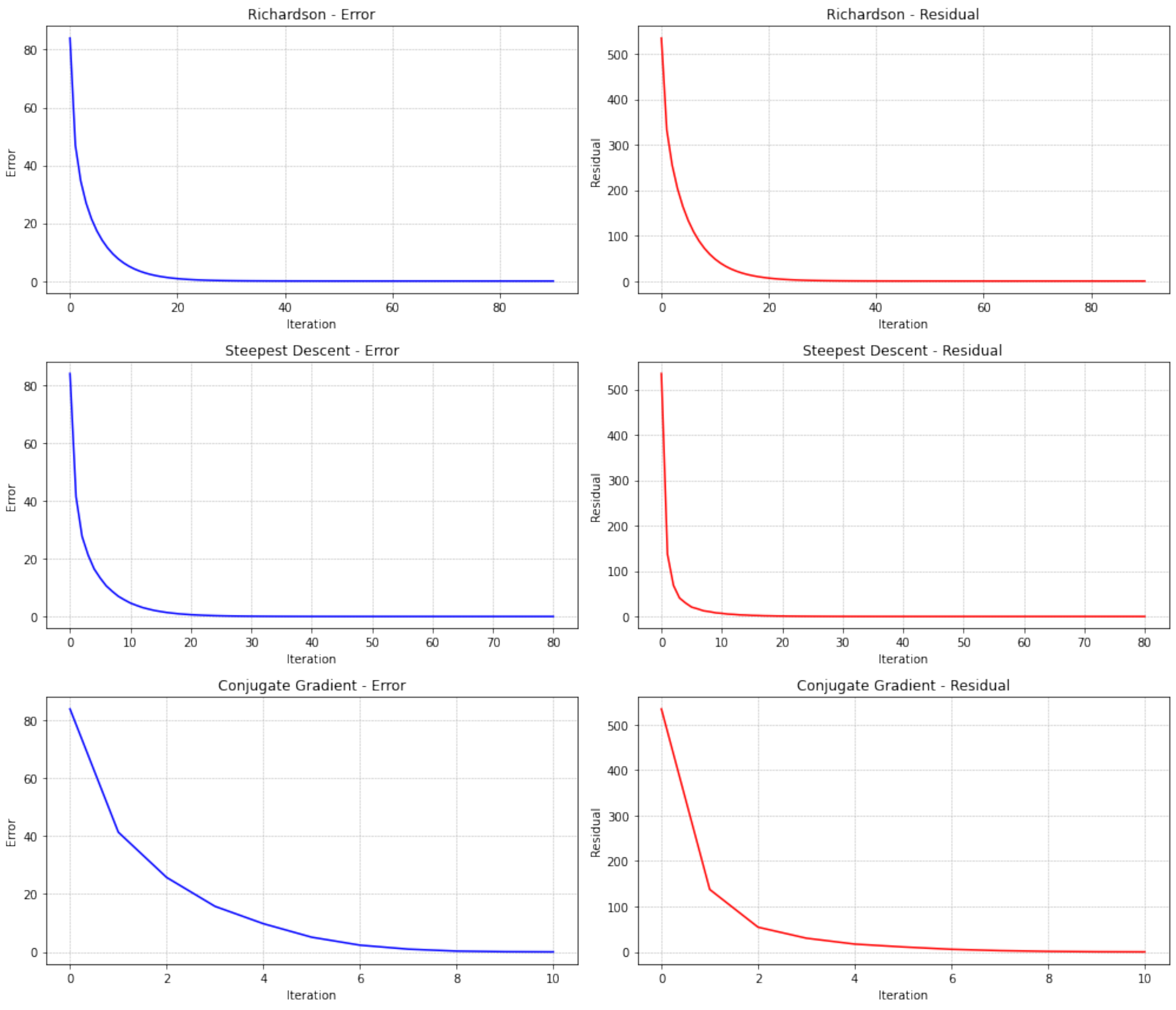


Figure 24: Error and Residual Convergence

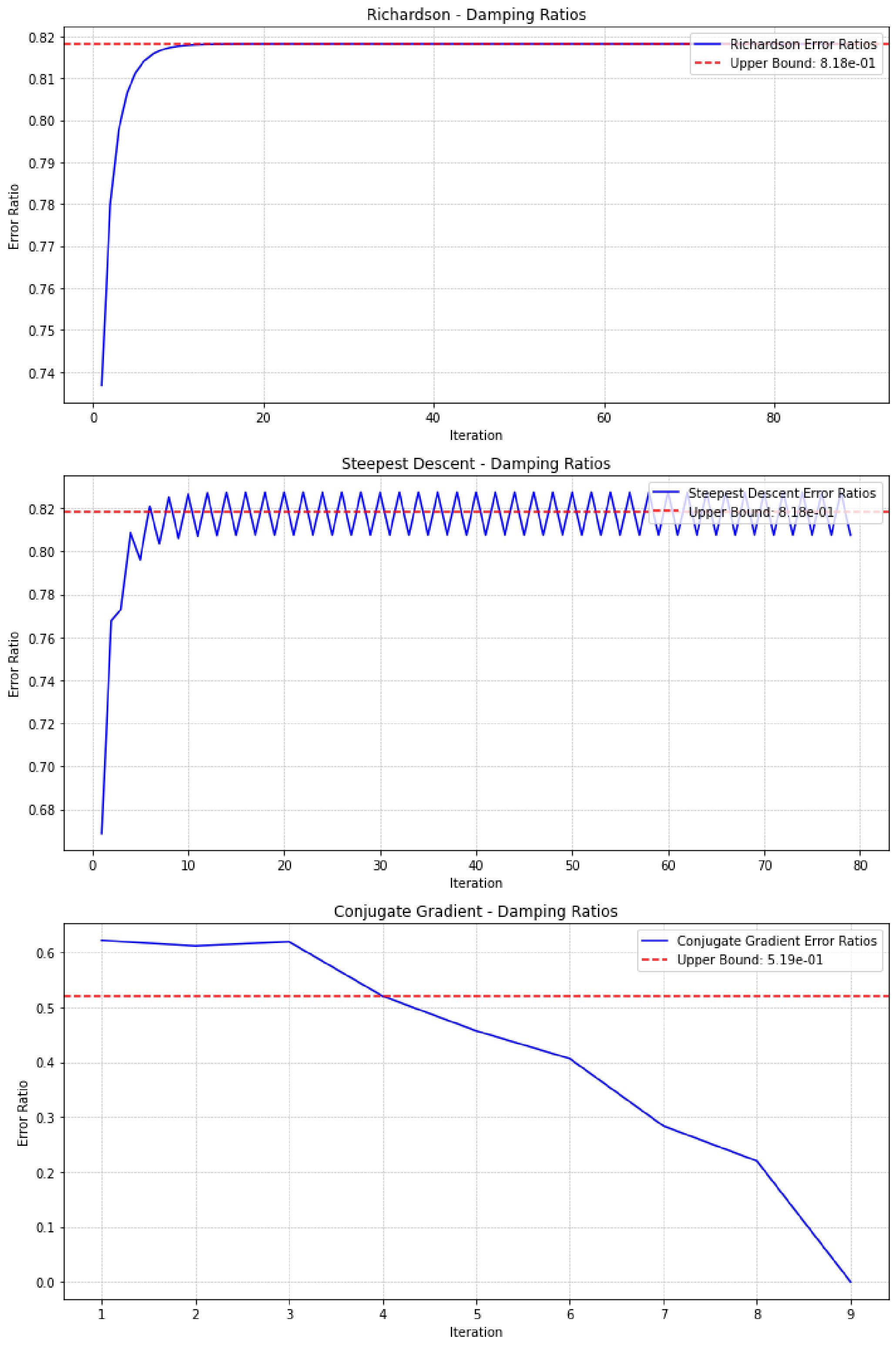


Figure 25: Damping Factors

Important Conclusion 2:
We again observe a more typical situation, where CG converges in at most $n = 100$ steps. In fact, it converges much quicker (10 steps in this case). By virtue of this, it takes order's of magnitude less total work than RF and SD. It also accurate up to twice as many decimal places. RF again outperforms SD except with iteration number. A larger sample of such experiments with these conditions will allow us to conclude that Richardson is typically better than Steepest descent and less prone to finite-precision errors. CG outperforms both methods vastly, but is very sensitive to and unreliable under numerical perturbation.

When SD does converge, we see another idiosyncratic behavior in terms of error damping i.e. that it tends to go down with iterations. This is consistent with theoretical results regards to convergence in CG: the damping factor actually goes down with k, where k is the number of iterations. This is also a reason for it's faster convergence. This fact can be probed more exactly, but is beyond the scope of this assignment. We only provide our observations as evidence to this fact. Note also the convergence behavior of these algorithms are asymptotic in general, which is why for CG for example the error damping starts above the theoretical bound and then decreases rapidly below the theoretical bound, and in fact keeps going down as predicted.

Let's see another such example where we make this even more apparent.

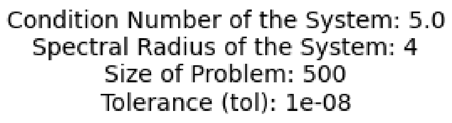
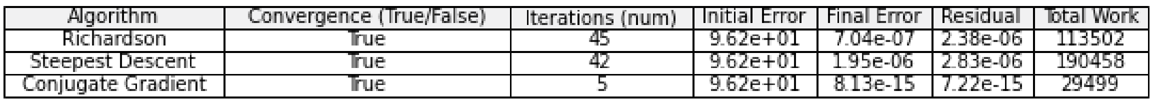

Condition Number of the System: 5.0
Spectral Radius of the System: 4
Size of Problem: 500
Tolerance (tol): 1e-08

Figure 26: Problem Characteristics



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Residual	Total Work
Richardson	True	45	9.62e+01	7.04e-07	2.38e-06	113502
Steepest Descent	True	42	9.62e+01	1.95e-06	2.83e-06	190458
Conjugate Gradient	True	5	9.62e+01	8.13e-15	7.22e-15	29499

Figure 27: Results Table

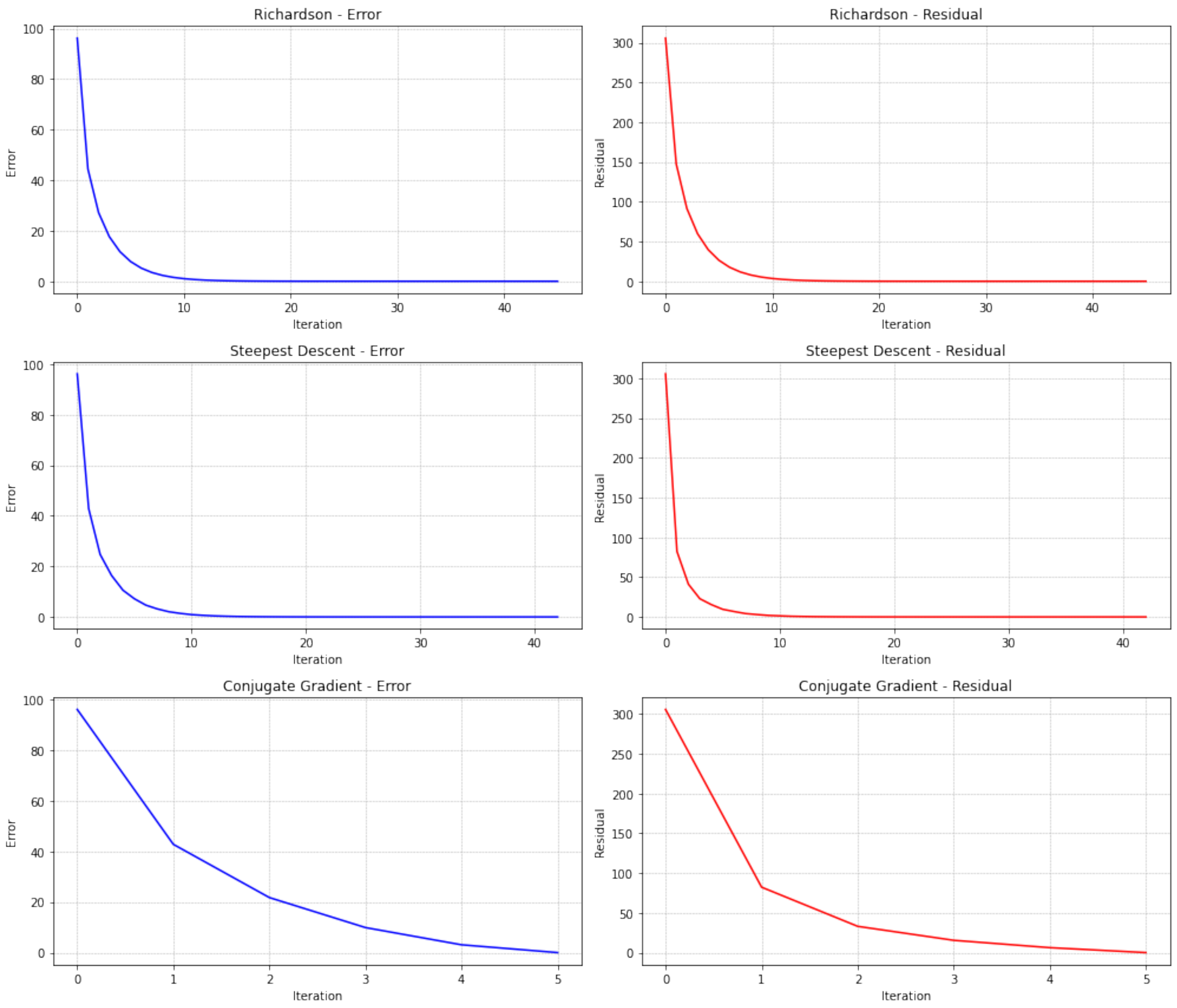


Figure 28: Error and Residual Convergence

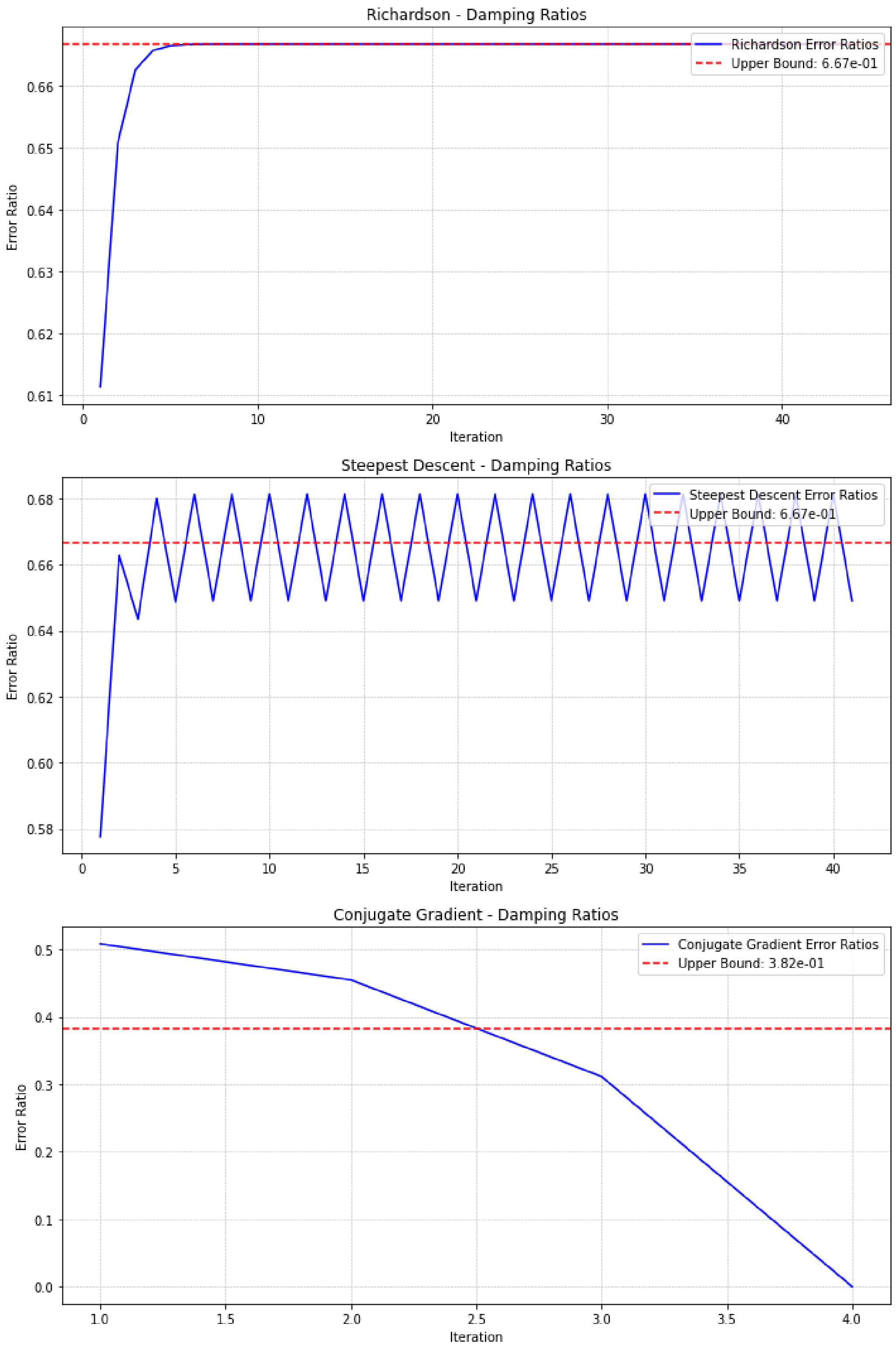
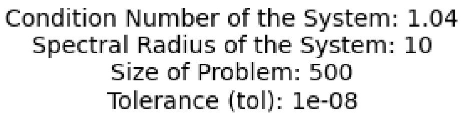


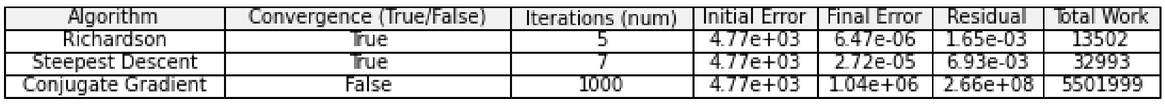
Figure 29: Damping Factors

The same observations can be made for this run. We can now also start to hypothesize that the number of iterations of all three algorithms correlates to the condition number of the system. In this case, because we end up choosing eigenvalues randomly, we can directly force certain ranges for the eigenvalues. Let us now choose bigger values for the spectrum but force them to be in a smaller range. For this experiment I chose to set dimension to 500 and force the values to be 250 and 260 (therefore forcing a small condition number).



Condition Number of the System: 1.04
Spectral Radius of the System: 10
Size of Problem: 500
Tolerance (tol): 1e-08

Figure 30: Problem Characteristics



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Residual	Total Work
Richardson	True	5	4.77e+03	6.47e-06	1.65e-03	13502
Steepest Descent	True	7	4.77e+03	2.72e-05	6.93e-03	32993
Conjugate Gradient	False	1000	4.77e+03	1.04e+06	2.66e+08	5501999

Figure 31: Results Table

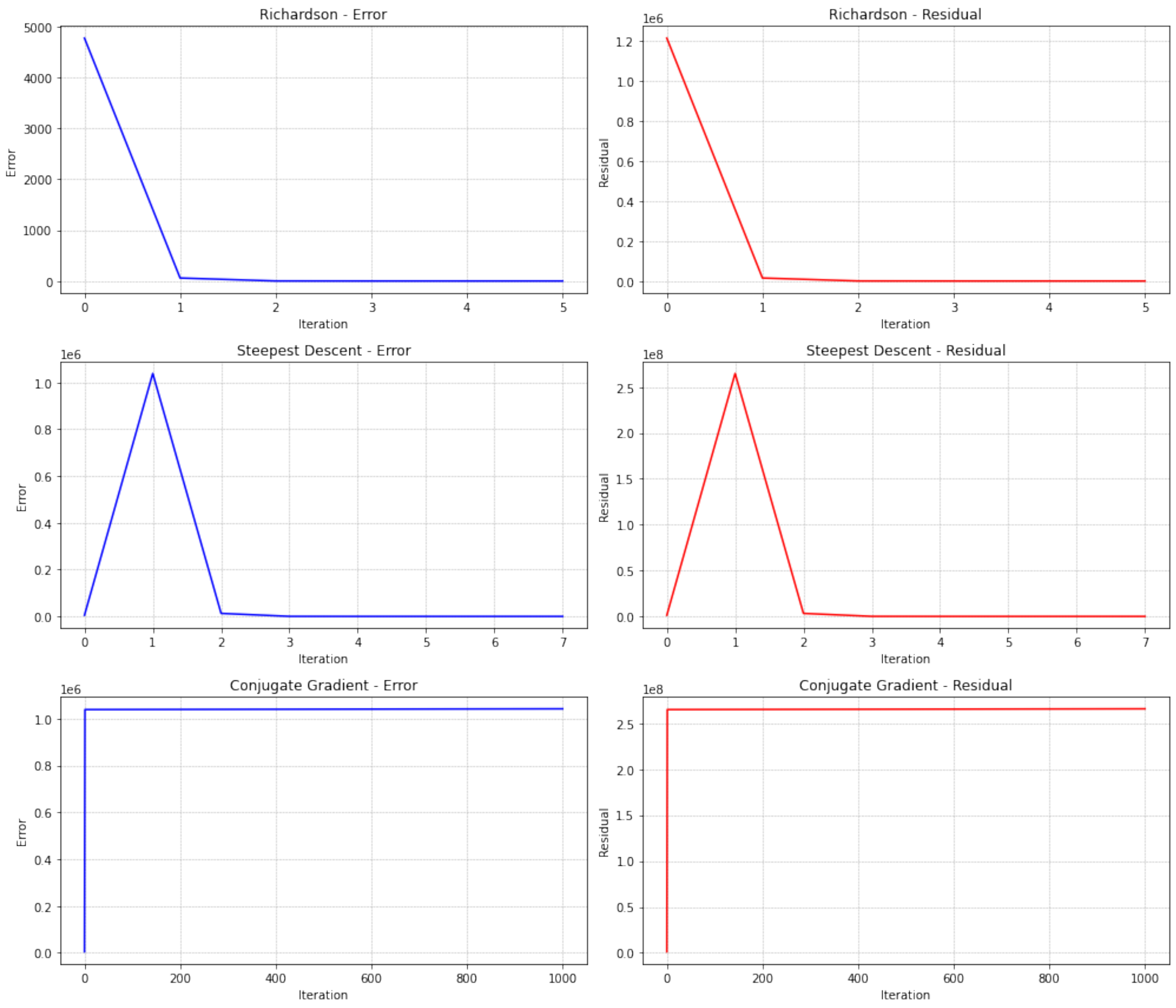


Figure 32: Error and Residual Convergence

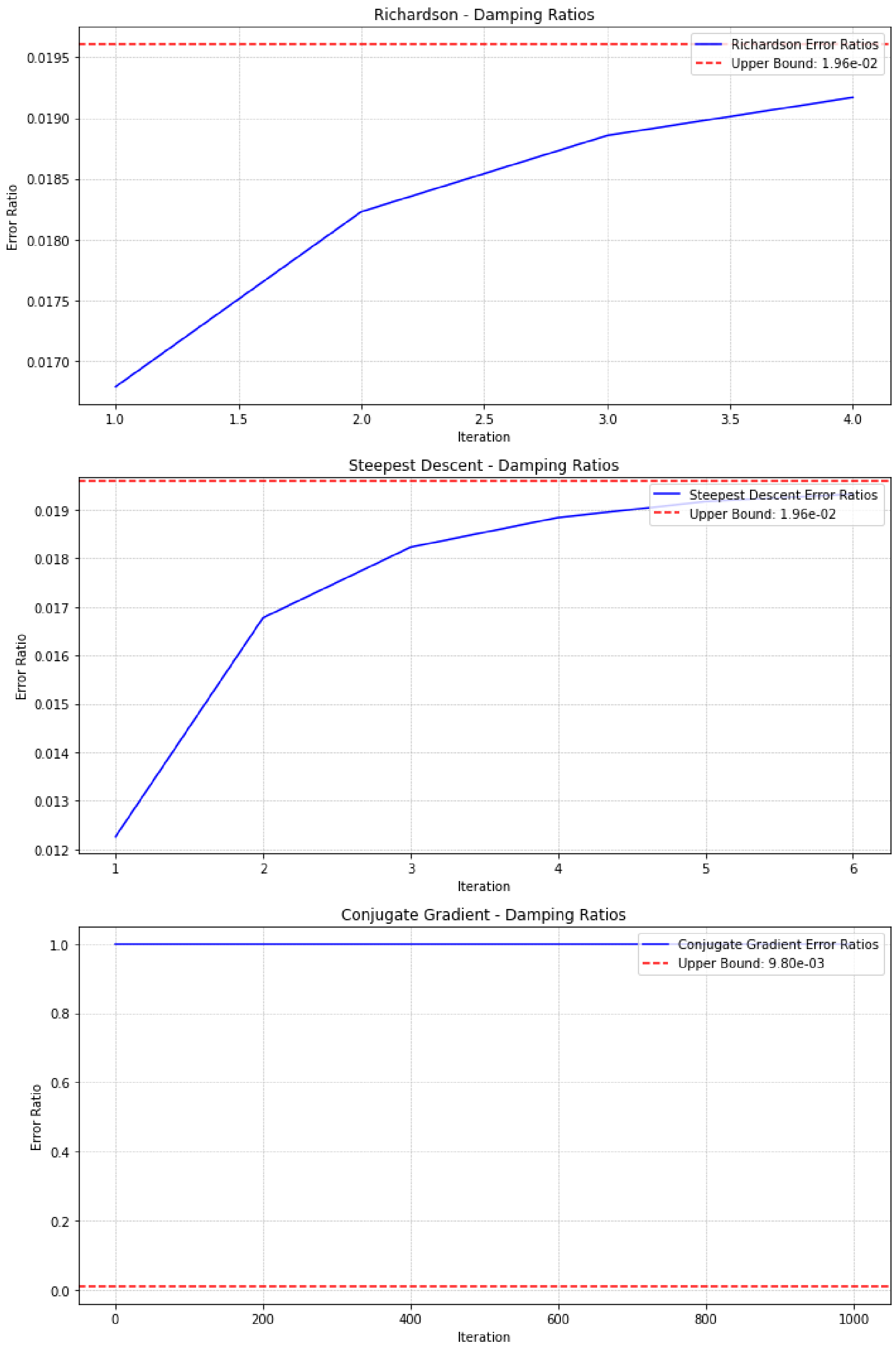


Figure 33: Damping Factors

Important Conclusion 3:
CG diverged for this experiment too. This is very strong evidence for Conclusion 1 - that it is numerical degeneration of Krylov space computations that is leading to this sensitivity of CG. RF and SD converge remarkably fast for this problem, providing strong evidence that it is indeed the condition number that determines performance. We are also ready to conclude that RF generally outperforms SD. The residual and error plots are less meaningful due to the small amount of iterations, but it is interesting to note that they still follow the same trends in a more discrete way. This confirms the theoretical results even more.

By repeated experimenting, we can find that after a certain threshold of numerical complexity, CG behaves abnormally as we saw for the larger-valued system experiments. It is therefore less reliable than the other methods. Keep in mind that for these experiments, I am choosing a random initial guess vector and a random b vector by choosing a random solution vector x^* (as can be seen from the initial error values in the tables). For the last experiment, I will consider a very large system with a very large condition number (spectral radius). This will help us finalize our hypotheses more generally, and also prove convergence criterion. The experiments are as follows:

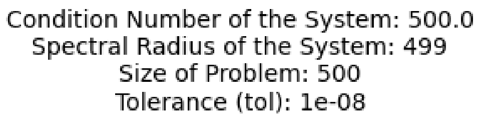
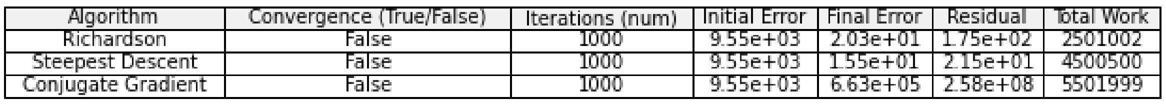

Condition Number of the System: 500.0
Spectral Radius of the System: 499
Size of Problem: 500
Tolerance (tol): 1e-08

Figure 34: Problem Characteristics



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Residual	Total Work
Richardson	False	1000	9.55e+03	2.03e+01	1.75e+02	2501002
Steepest Descent	False	1000	9.55e+03	1.55e+01	2.15e+01	4500500
Conjugate Gradient	False	1000	9.55e+03	6.63e+05	2.58e+08	5501999

Figure 35: Results Table

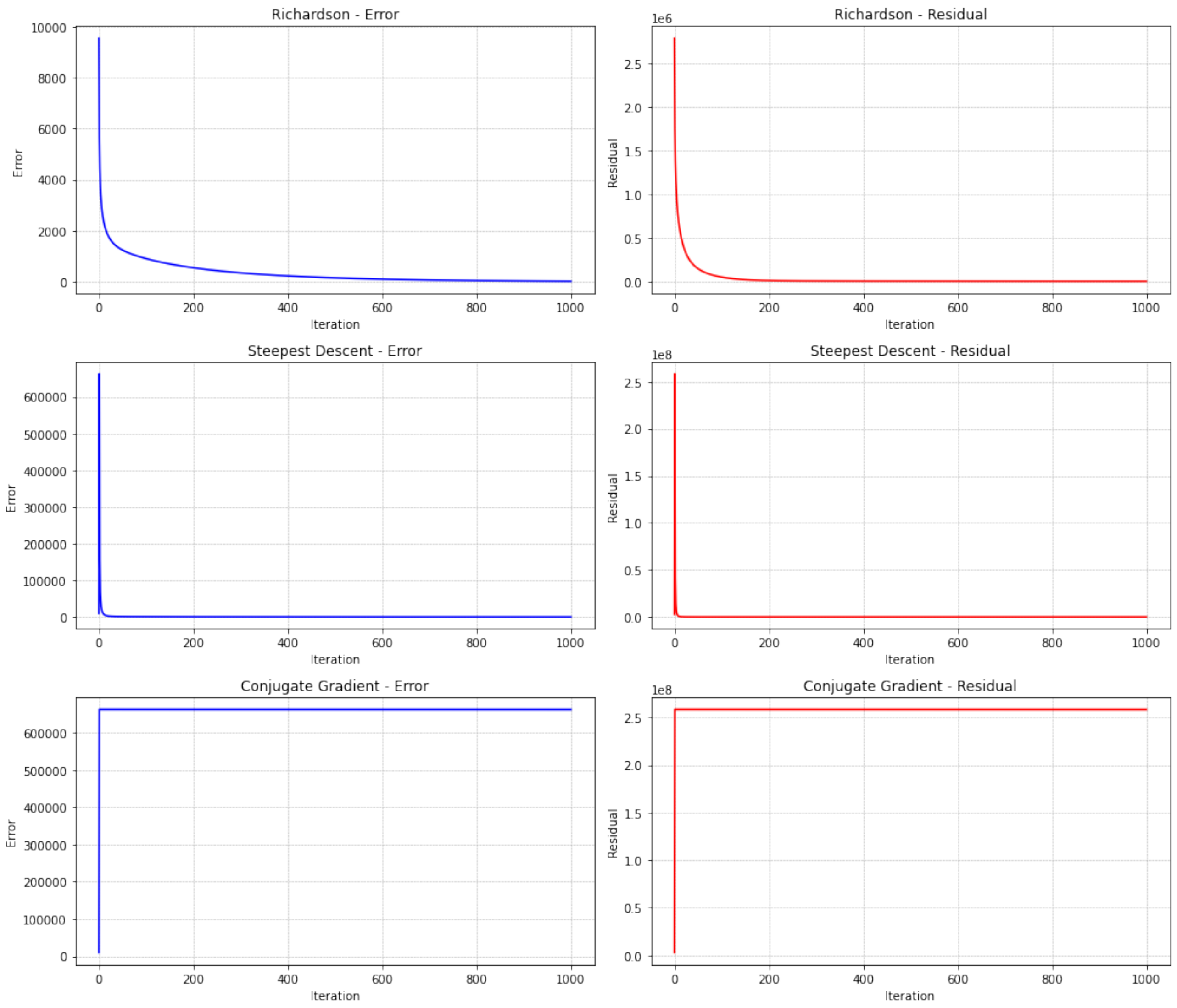


Figure 36: Error and Residual Convergence

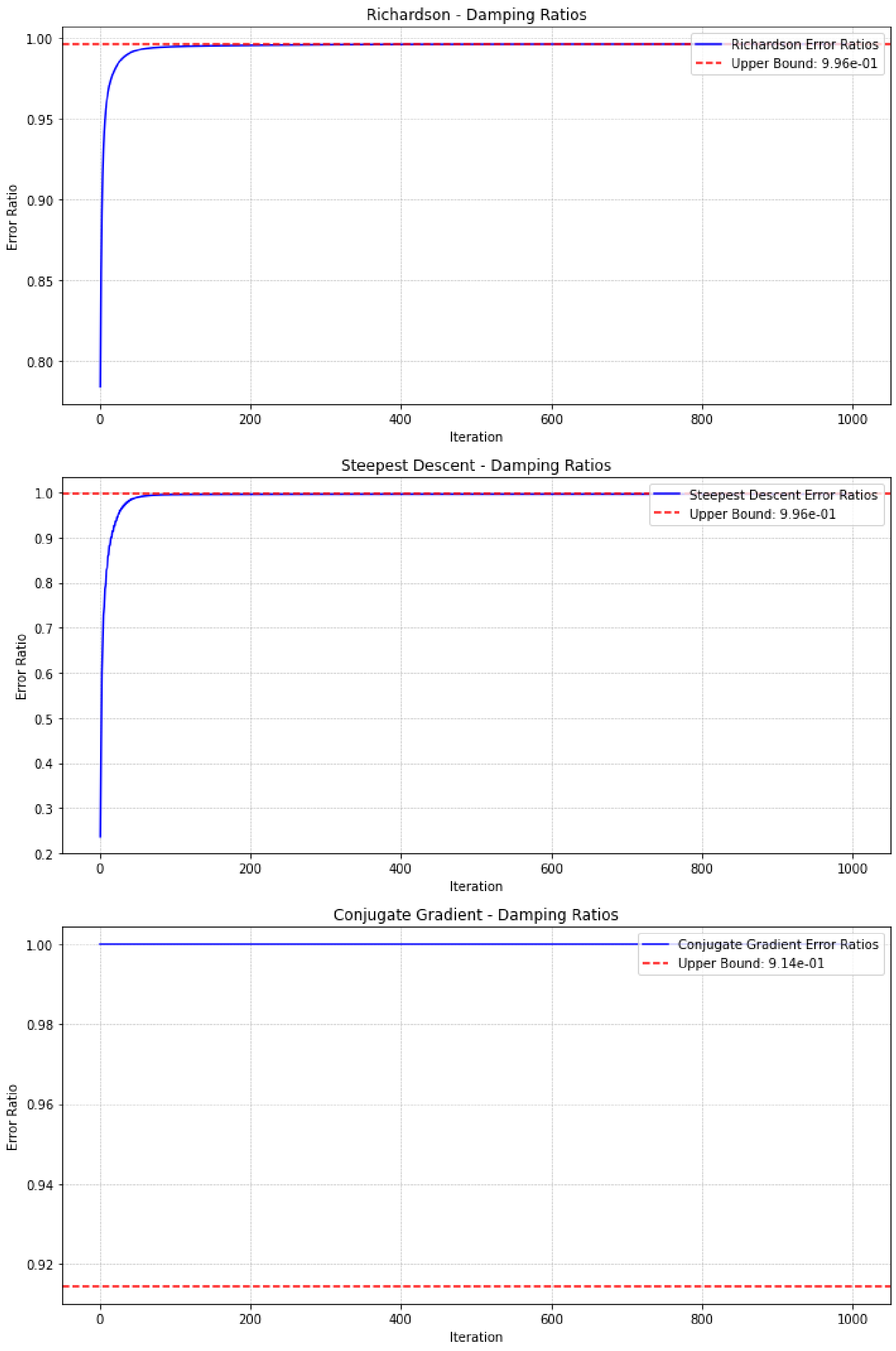
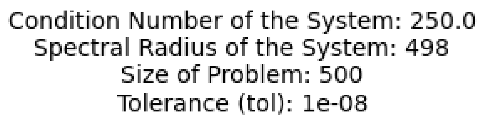


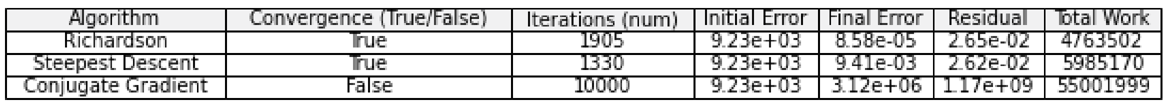
Figure 37: Damping Factors

In this particular experiment, due to randomization, we also had a large error for our initial guess. Smoother convergence is observed for RF and SD. The initial shock in the error curve of the SD hides the overall convergence rate, but with more a more careful look at the error plot (e.g. by slicing the shock out) we can observe that is looks identical to RF. CG diverges in this case, which we expect for such large values. It is also clear that it will be useful in this case to allow more iterations to happen. We get the expected results:



Condition Number of the System: 250.0
Spectral Radius of the System: 498
Size of Problem: 500
Tolerance (tol): 1e-08

Figure 38: Problem Characteristics



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Residual	Total Work
Richardson	True	1905	9.23e+03	8.58e-05	2.65e-02	4763502
Steepest Descent	True	1330	9.23e+03	9.41e-03	2.62e-02	5985170
Conjugate Gradient	False	10000	9.23e+03	3.12e+06	1.17e+09	55001999

Figure 39: Results Table

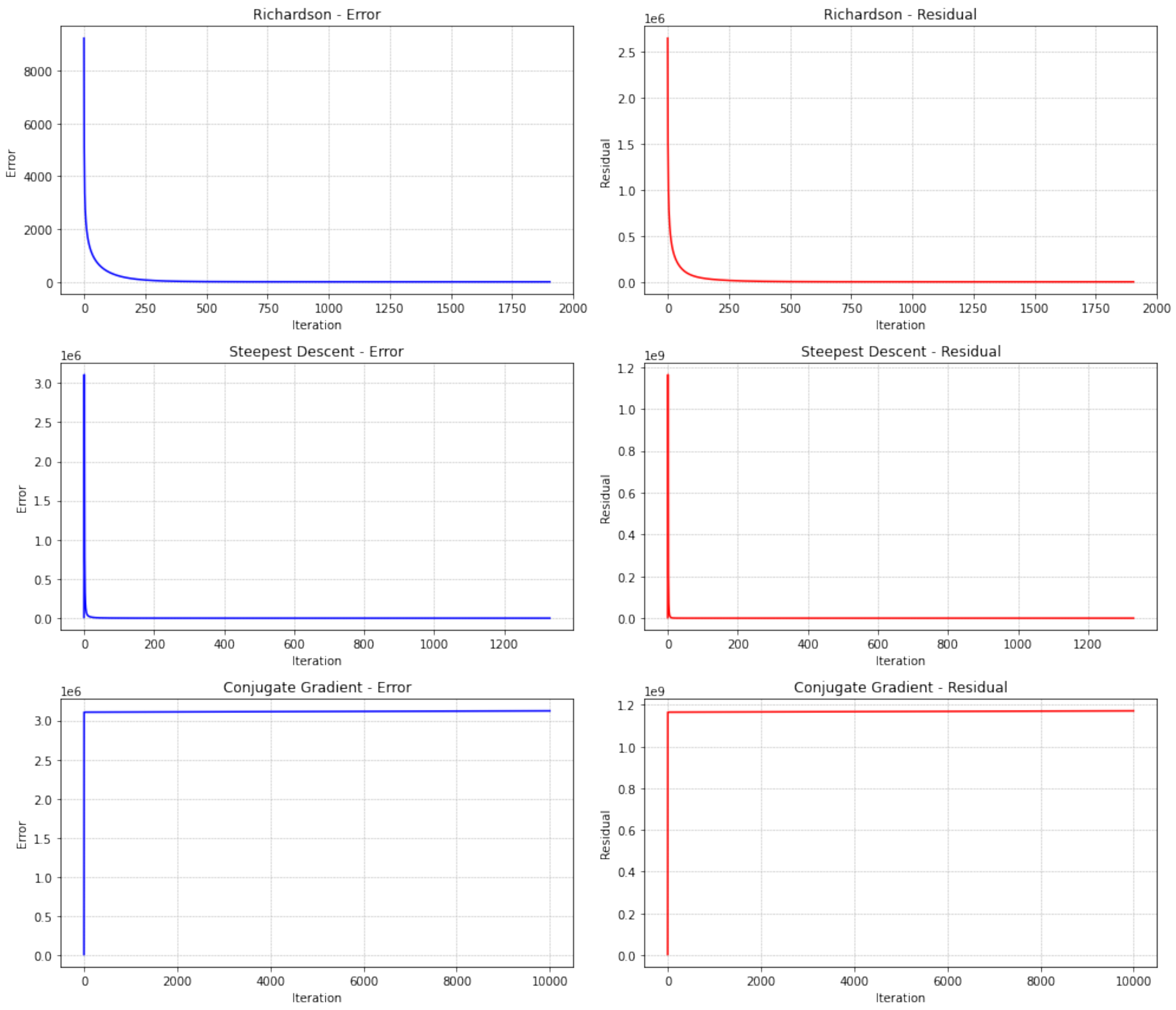


Figure 40: Error and Residual Convergence

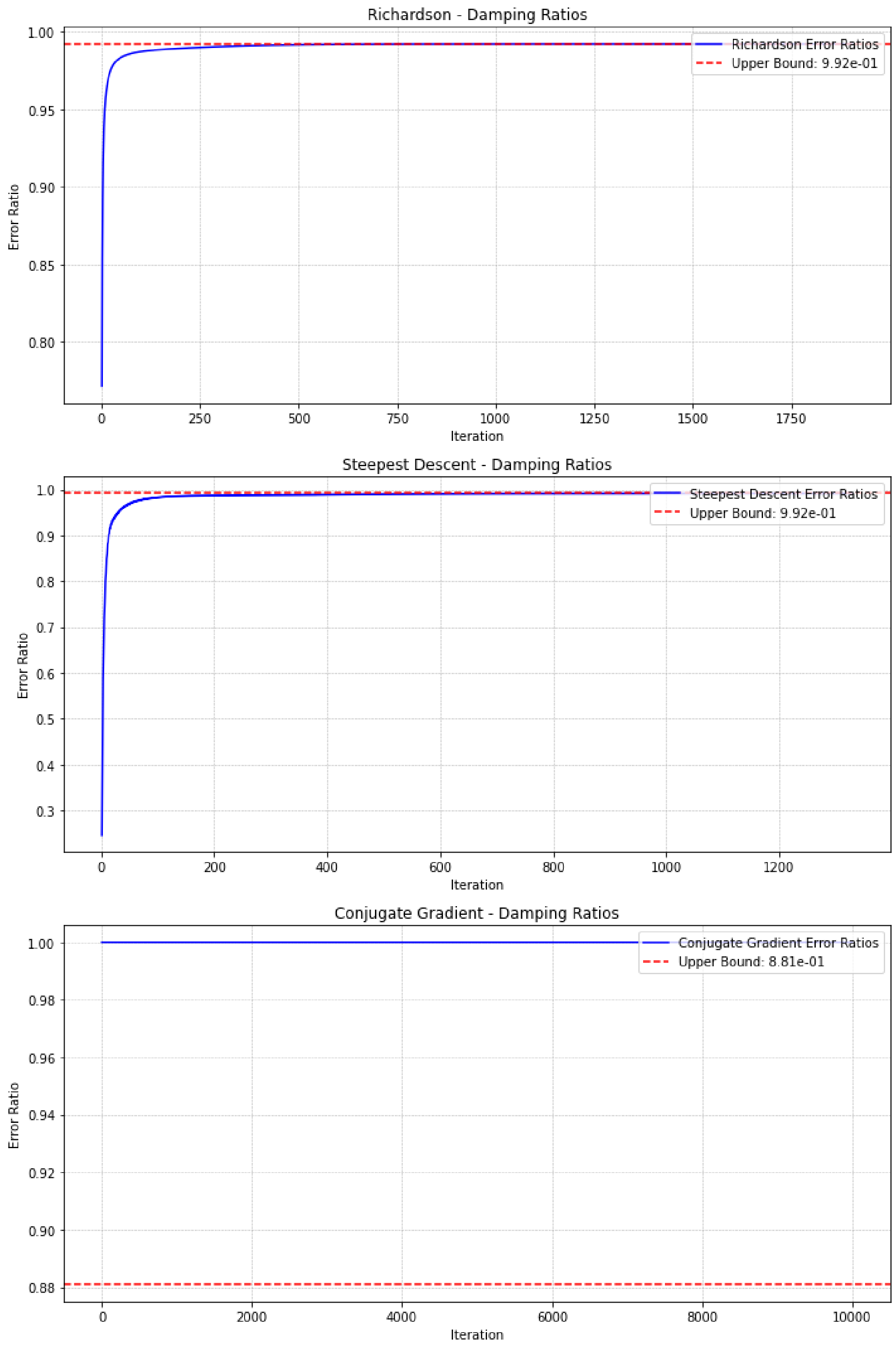


Figure 41: Damping Factors

Thus we can also conclude that the maximum iterations have to be increased for the same tolerance level if we want convergence to the same level of accuracy. This can be probed further, but all of these properties are common to iterative algorithms of this kind in general, thus not particularly important to our experiments. We can assume the expected behavior for tolerance and iterations.

Now that we have done initial testing and seen how these algorithms behave, we can conduct larger experiments and present statistical evidence for our hypotheses. We will also conduct structured experiments for the spectrum of the matrices using the examples provided in the assignment.

Spectral Analysis

We have demonstrated that the dimension of the problem has a very minor effect in analysis of convergence. Thus we can run our experiments for a fairly large dimension value, and appeal to the heuristic that for smaller dimension that results will tend to agree with prediction to a higher degree (for our eigen-basis, this is even more true due to the induced simplicity). Also note that the analysis with varying the spectra of the system is equivalent in our case to changing the size of the values of the system in the eigen-basis. Then for this section, we will set dimension = 100, MAXITER = 1000, and tol = $1^{-08} = 10^{-9}$. Since this will more or less accommodate the extremes of our generated problems. We have already seen the effect of changing the spectrum (i.e. the values) on our algorithms. In these section, we will impose extra structure on the spectrum and evaluate the algorithms similarly for these set of problems.

- all n eigenvalues the same;
- k distinct eigenvalues with the multiplicities of each chosen deterministically or randomly;
- k distinct eigenvalues that are used as the mean of a normally distributed selection of a “cloud” of eigenvalues around each distinct eigenvalue (the number in each must also be chosen as the multiplicities in the previous item);
- from a uniform distribution between parameters the two extreme eigenvalues included in the problem, $\lambda_{\min}$ and $\lambda_{\max}$, that you choose in order to set κ;
- from a normal distribution between parameters the two extreme eigenvalues included in the problem, $\lambda_{\min}$ and $\lambda_{\max}$, that you choose in order to set κ with a selected mean and variance.

For each one, we first look at a representative example (to be able to see plots) so we can see what results we can expect from the larger set of parameterized problems for many initial guesses.

The **first problem** we can check immediately using the randomizer by fixing min and max in the random function for A to be the same constant value. This will set each eigenvalue to this constant. We expect immediate convergence in this case since the error at the first iteration is forced to be 0 by error theorem in the first section. Indeed, this is what we observe:

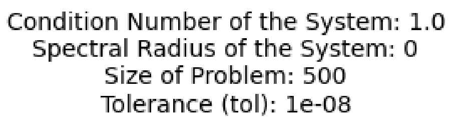
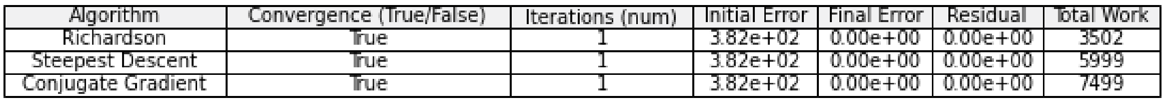

```
Condition Number of the System: 1.0
Spectral Radius of the System: 0
Size of Problem: 500
Tolerance (tol): 1e-08
```

Figure 42: Problem Characteristics



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Residual	Total Work
Richardson	True	1	3.82e+02	0.00e+00	0.00e+00	3502
Steepest Descent	True	1	3.82e+02	0.00e+00	0.00e+00	5999
Conjugate Gradient	True	1	3.82e+02	0.00e+00	0.00e+00	7499

Figure 43: Results Table

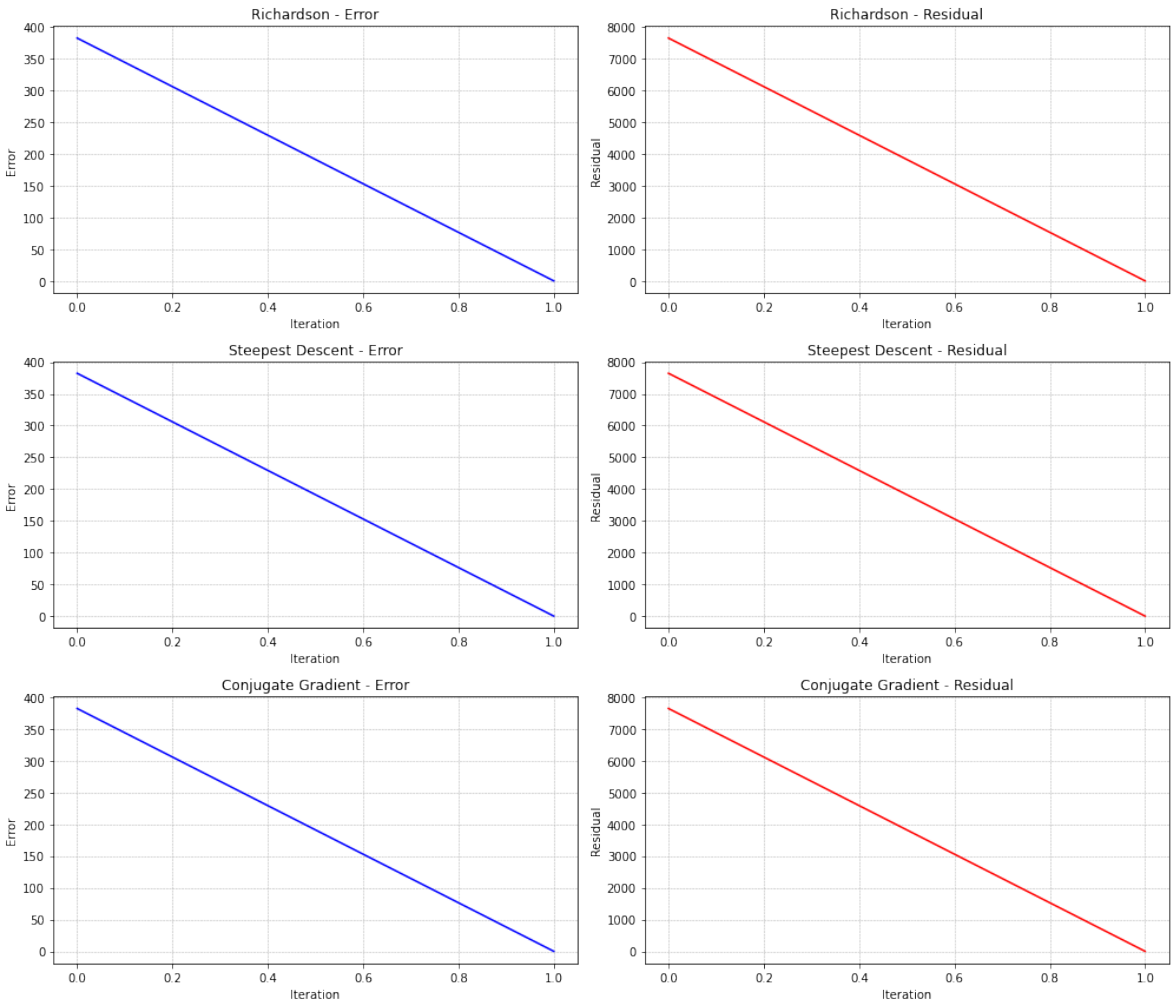


Figure 44: Error and Residual Convergence

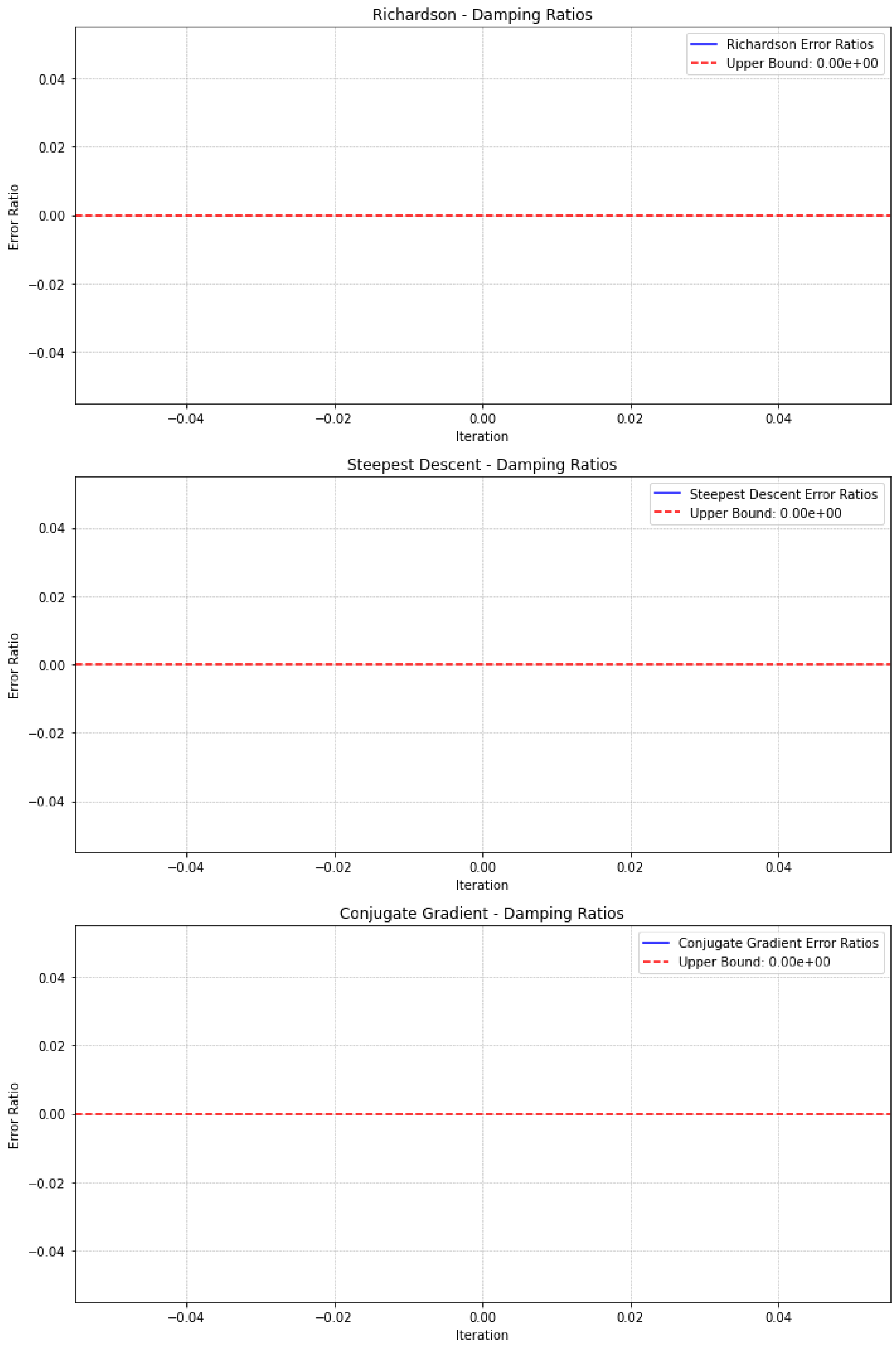
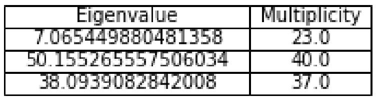


Figure 45: Damping Factors

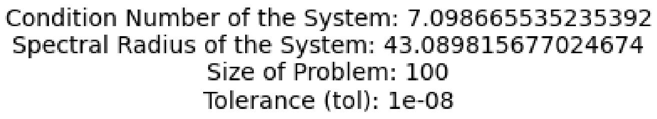
We used a smaller eigenvalue constant in this case (20). Setting larger values for this, combined with larger initial guess causes more iterations for RF and SD sometimes, but only as far as 2, or 3 for very numerically unstable problems. In these cases, CG is also at risk of diverging by the previous discussion.

For the **second problem**, we can write a python function that returns such a vector with the required parameters. We can set the range of possible eigenvalues, and how many distinct eigenvalues we require (this is also randomized, but can be made deterministic). The multiplicities are chosen randomly. We will present 3 of these experiments with plots, but the blocks of code for these tests can be run individually in the python file with any specifications. The first example has the following randomly generated specifications:



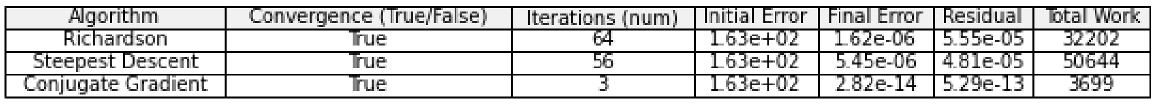
Eigenvalue	Multiplicity
7.065449880481358	23.0
50.155265557506034	40.0
38.0939082842008	37.0

Figure 46: Eigen spectrum



```
Condition Number of the System: 7.098665535235392
Spectral Radius of the System: 43.089815677024674
Size of Problem: 100
Tolerance (tol): 1e-08
```

Figure 47: Problem Characteristics



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Residual	Total Work
Richardson	True	64	1.63e+02	1.62e-06	5.55e-05	32202
Steepest Descent	True	56	1.63e+02	5.45e-06	4.81e-05	50644
Conjugate Gradient	True	3	1.63e+02	2.82e-14	5.29e-13	3699

Figure 48: Results Table

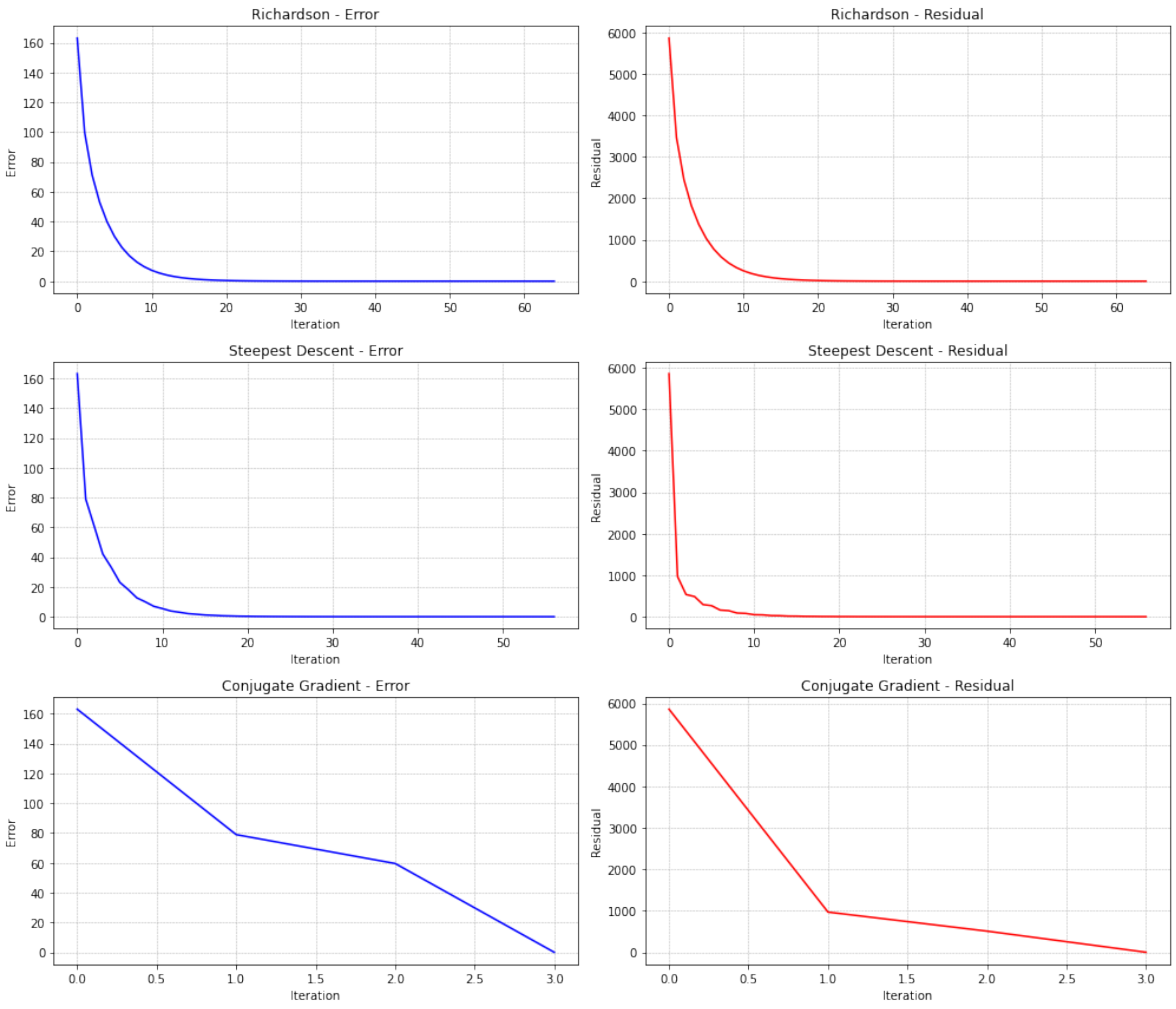


Figure 49: Error and Residual Convergence

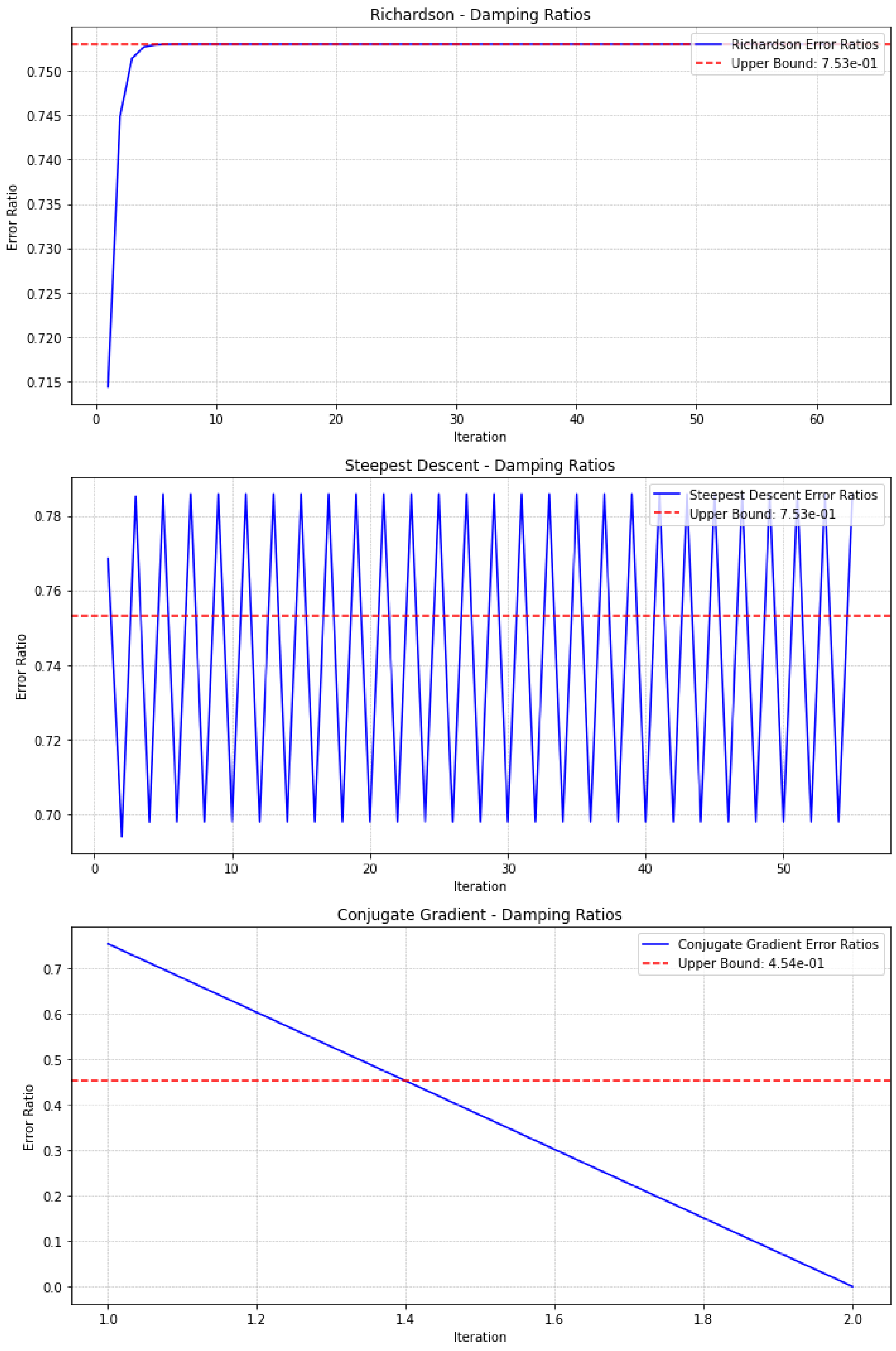
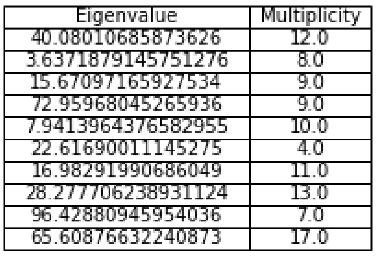


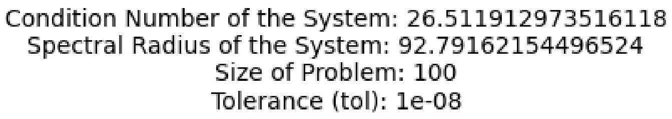
Figure 50: Damping Factors

We can first notice that the extra structure causes our algorithms to behave as expected. CG in particular performs very well, and much faster. We can run a series of these experiments for more insight:



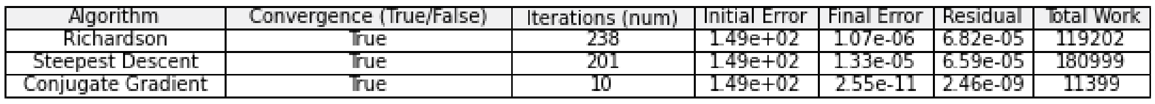
Eigenvalue	Multiplicity
40.08010685873626	12.0
3.6371879145751276	8.0
15.67097165927534	9.0
72.95968045265936	9.0
7.9413964376582955	10.0
22.61690011145275	4.0
16.98291990686049	11.0
28.277706238931124	13.0
96.42880945954036	7.0
65.60876632240873	17.0

Figure 51: Eigen spectrum



Condition Number of the System: 26.511912973516118
Spectral Radius of the System: 92.79162154496524
Size of Problem: 100
Tolerance (tol): 1e-08

Figure 52: Problem Characteristics



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Residual	Total Work
Richardson	True	238	1.49e+02	1.07e-06	6.82e-05	119202
Steepest Descent	True	201	1.49e+02	1.33e-05	6.59e-05	180999
Conjugate Gradient	True	10	1.49e+02	2.55e-11	2.46e-09	11399

Figure 53: Results Table

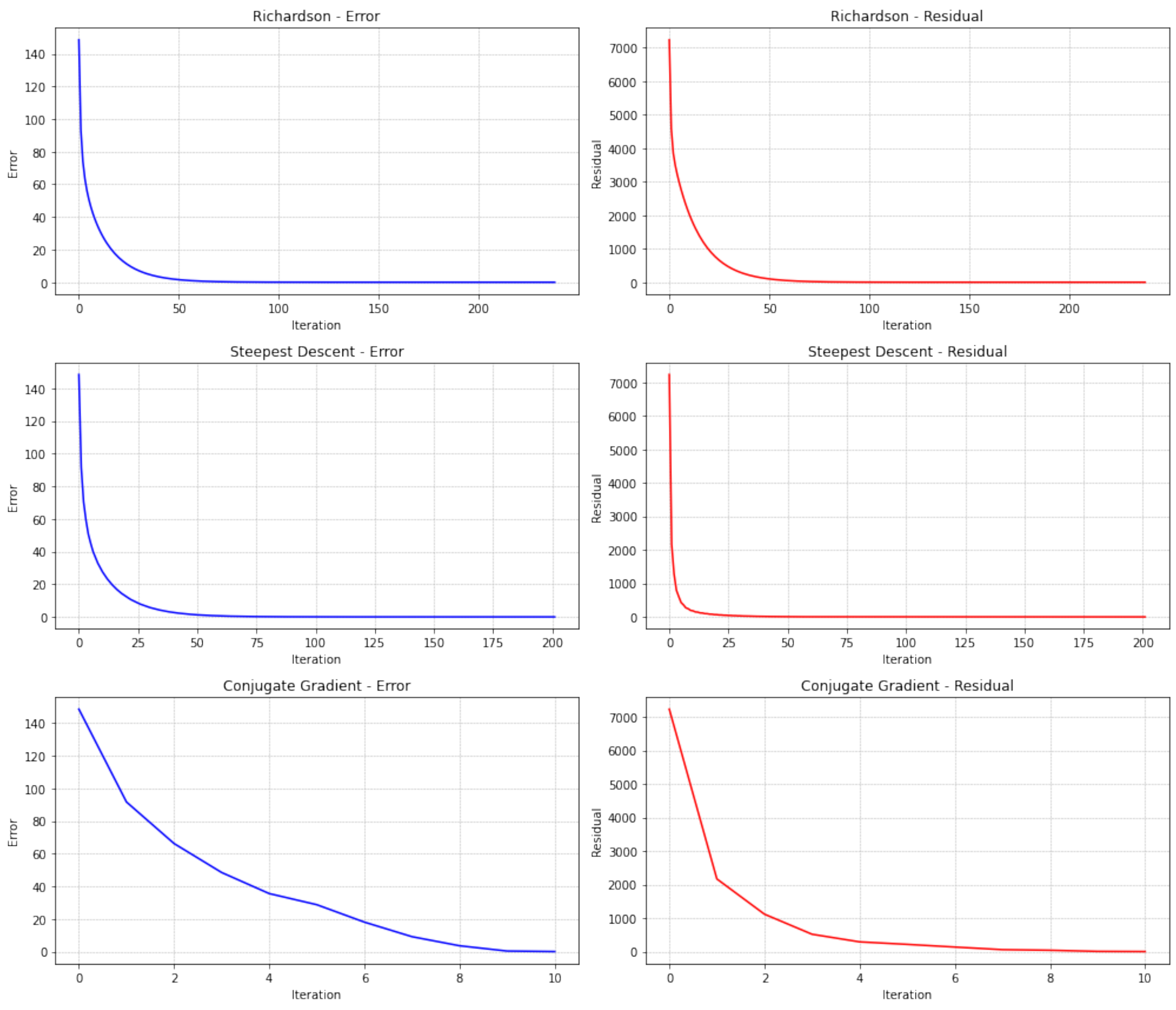


Figure 54: Error and Residual Convergence

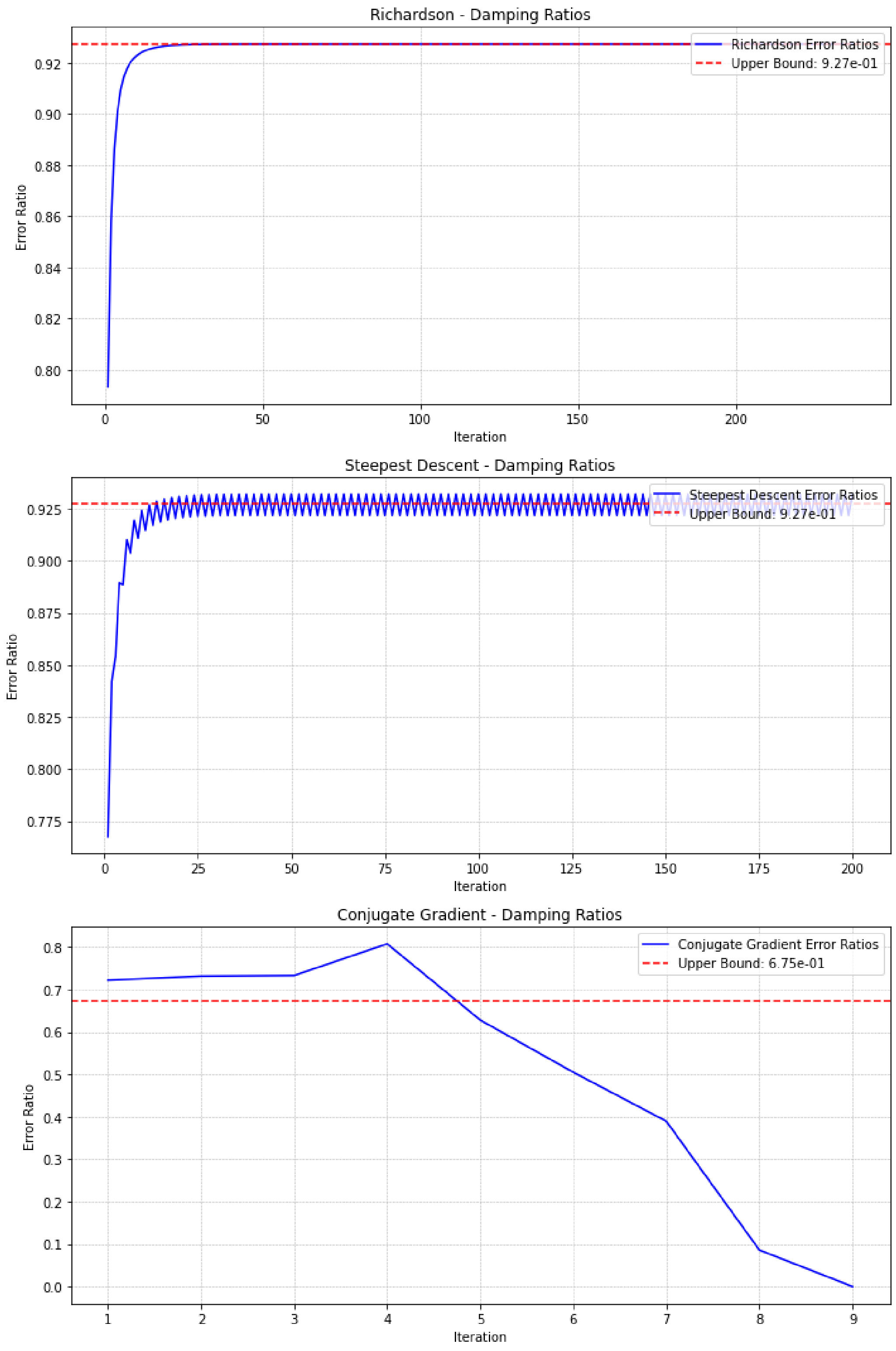
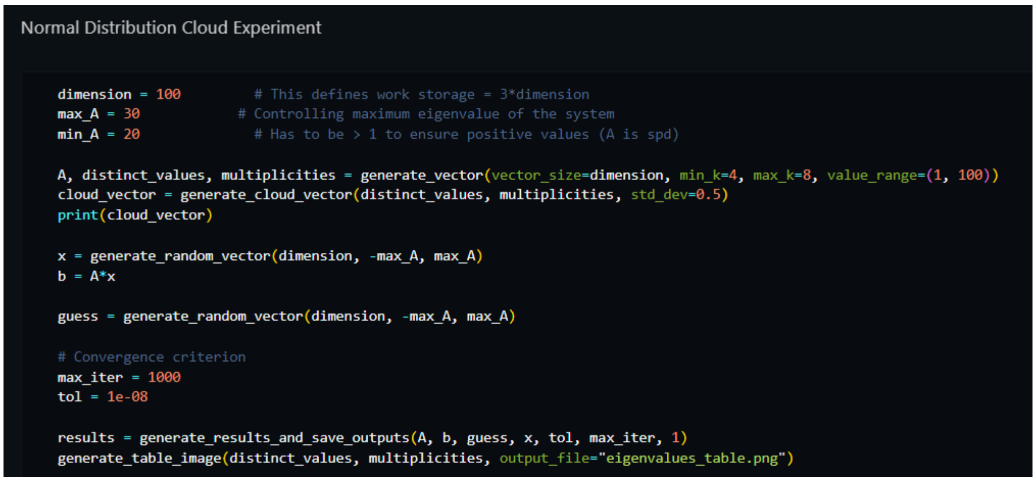


Figure 55: Damping Factors

Under numerically stable conditions, we get very similar results for each problem in this set! However, the the larger the range of eigenvalues produced with varying degrees of multiplicities, the slower RF and SD become. However, the more concentrated the system, the more this effect is damped. This is consistent with our predictions, since the concentrations of the eigenvalues (the spread) controls the condition number.

For **problems 3 and 5**, we in fact get very similar results as above, but the performance of the algorithms is related to the range of the eigenvalues and also the maximum ratio of the means of the normal distributions used for the 'clouds' (for 3) and related to the spread of the normal distribution for problem 5. Regardless, we make separate blocks for these experiments in the Jupyter notebook, which can be run manually to conduct more experiments. For example, for Problem 3, we have the following block:

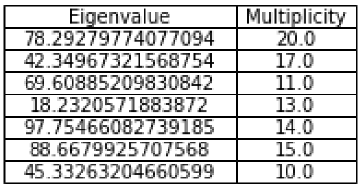


Normal Distribution Cloud Experiment

```
dimension = 100         # This defines work storage = 3*dimension
max_A = 30            # Controlling maximum eigenvalue of the system
min_A = 20              # Has to be > 1 to ensure positive values (A is spd)

A, distinct_values, multiplicities = generate_vector(vector_size=dimension, min_k=4, max_k=8, value_range=(1, 100))
cloud_vector = generate_cloud_vector(distinct_values, multiplicities, std_dev=0.5)
print(cloud_vector)

x = generate_random_vector(dimension, -max_A, max_A)
b = A*x

guess = generate_random_vector(dimension, -max_A, max_A)

# Convergence criterion
max_iter = 1000
tol = 1e-08

results = generate_results_and_save_outputs(A, b, guess, x, tol, max_iter, 1)
generate_table_image(distinct_values, multiplicities, output_file="eigenvalues_table.png")
```

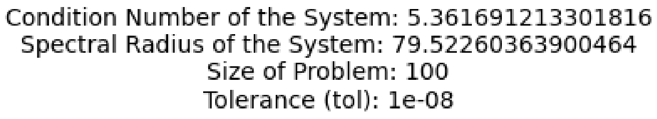
Figure 56: Problem Set 3: Normal Clouds

We can display results for one such experiment for number 3. For each cloud, we set a the variance to be 0.5:



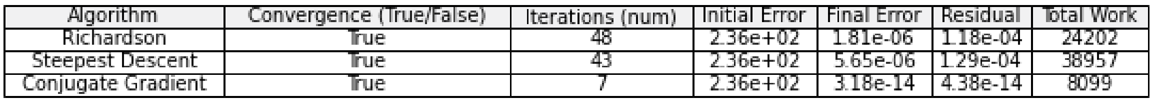
Eigenvalue	Multiplicity
78.29279774077094	20.0
42.34967321568754	17.0
69.60885209830842	11.0
18.2320571883872	13.0
97.75466082739185	14.0
88.6679925707568	15.0
45.33263204660599	10.0

Figure 57: Eigen spectrum



Condition Number of the System: 5.361691213301816
Spectral Radius of the System: 79.52260363900464
Size of Problem: 100
Tolerance (tol): 1e-08

Figure 58: Problem Characteristics



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Residual	Total Work
Richardson	True	48	2.36e+02	1.81e-06	1.18e-04	24202
Steepest Descent	True	43	2.36e+02	5.65e-06	1.29e-04	38957
Conjugate Gradient	True	7	2.36e+02	3.18e-14	4.38e-14	8099

Figure 59: Results Table

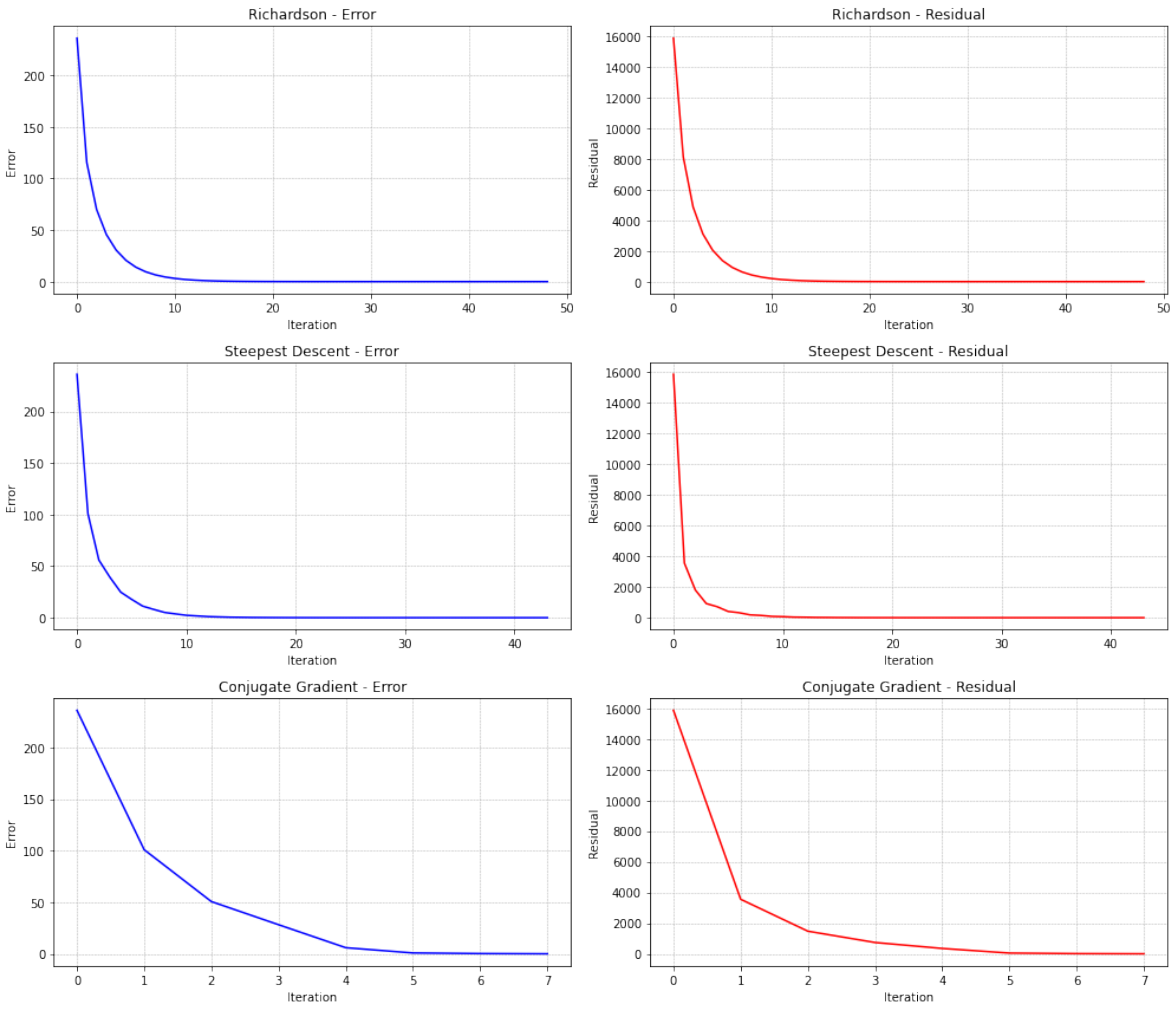


Figure 60: Error and Residual Convergence

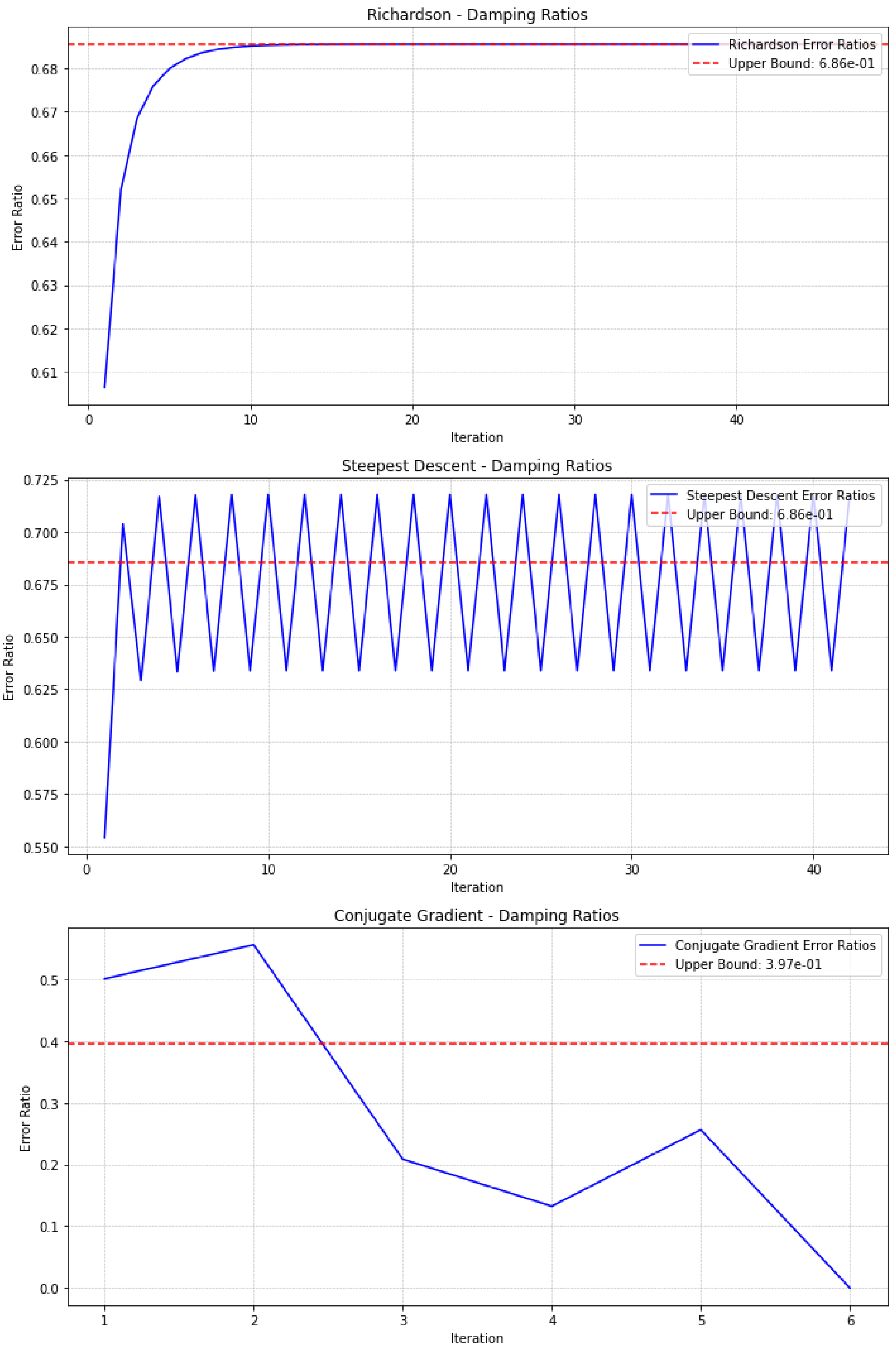


Figure 61: Damping Factors

Again, the most important observation is that CG performs extremely efficiently on the normally structured problems in particular. There is also less of a difference between RF and SD for these problems, but RF is still more efficient. We do not need to show results for the **fourth problem** since this is exactly how the original problem generator was designed (i.e. as a uniformly generated random positive vector). We tested this extensively in the first part of the experimental design.

Now that we have collected preliminary evidence and visualization for our hypothesis, we can run complete experiments.

Large Parameterized Experiments

In this section we run a larger set of experiments and test many of our hypothesis directly. There are five main results that we will try to demonstrate in the experiments.

- Convergence reliability of each algorithm
- Speed of each algorithm with different initial guesses
- Efficiency of each algorithm with different initial guesses
- Speed of each algorithm with varying spectra
- Efficiency of each algorithm with varying spectra

For the design, I fix the dimension, tolerance and maximum allowed iterations uniformly. I use counts of convergence as a test for convergence reliability and I use number of iterations as a proxy for speed, total work done as a proxy for efficiency, and the condition number as representing a particular spectrum. Please consult the script file for complete details of the design and implementation. We run these experiments (50 at first, and then an increasing number) and present the results:

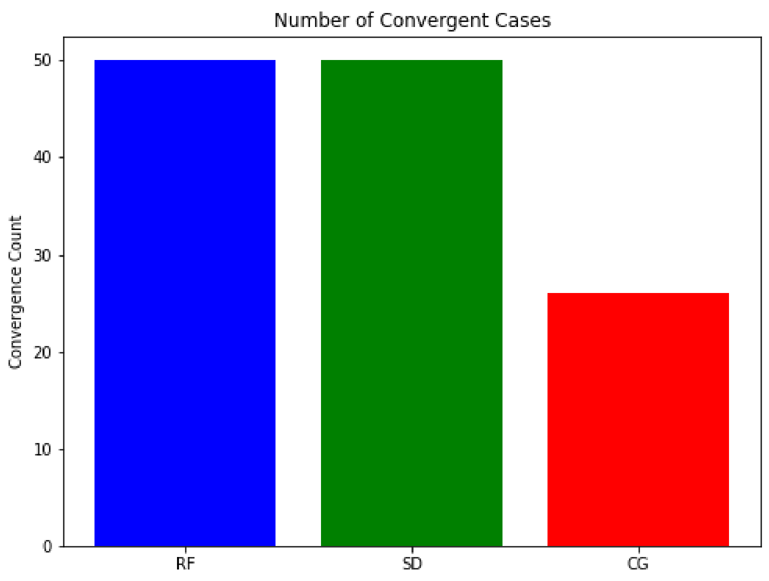


Figure 62: Convergence Reliability

RF and SD converged in every single case for the first run of experiments, and CG only converged for about half the values. Clearly, CG is much less reliable than both RF and SD. This becomes even more apparent in the subsequent result:

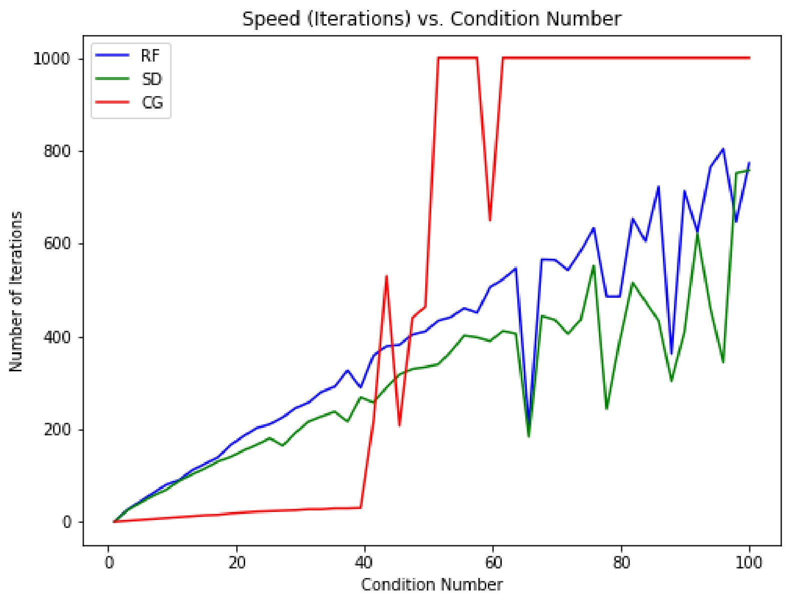


Figure 63: Speed against Spectra

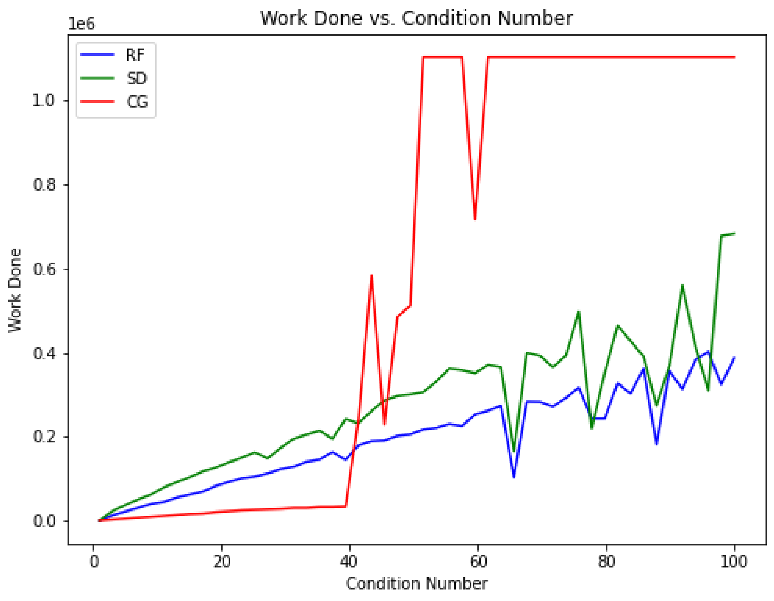


Figure 64: Total Work against Spectra

These are extremely insightful plots for the speed and total work of each algorithm. Generally, the speed tends to slow down (higher number of iterations) for a higher condition number. RF was almost always faster than SD, but they followed a closer pattern, and the difference was not as significant. CG tends to becomes numerically unstable for values of condition number larger than 40. Below 40, it is much faster than both RF and SD, but it becomes unstable very quickly and subsequently starts diverging for values of $\kappa \geq 50$. This is consistent with results we observed manually. For certain values the algorithms become uncharacteristically faster (and CG even converges). I suspect that this is due to numerical pitfalls (and perhaps luck). The total work done also follows a similar pattern, except that RF usually takes less work than SD, so it is more efficient. This is also what we observed before!

The initial error was less consequential to each algorithm:

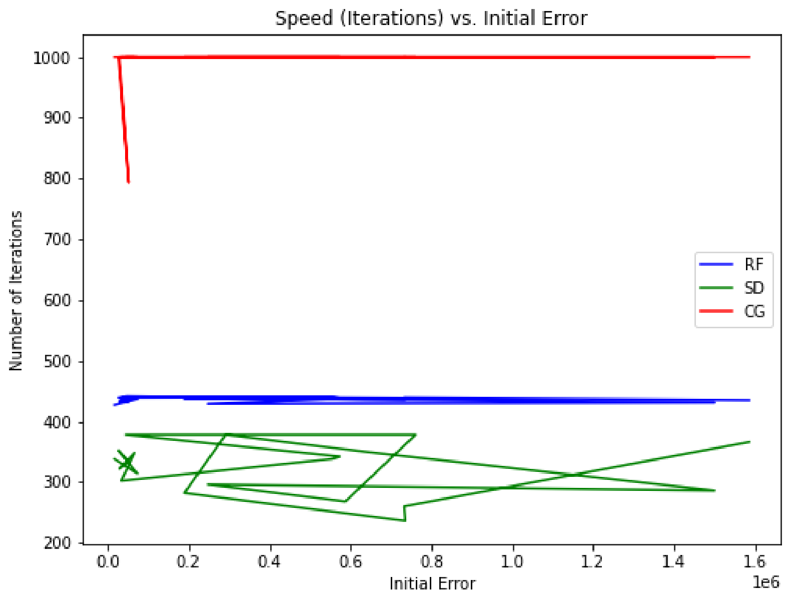


Figure 65: Speed against Initial Guess

We are only able to establish the same hierarchy of algorithms as before since this presents even more evidence of the order of speed and efficiency. We can also observe CG to be very sensitive to perturbations in initial guesses.

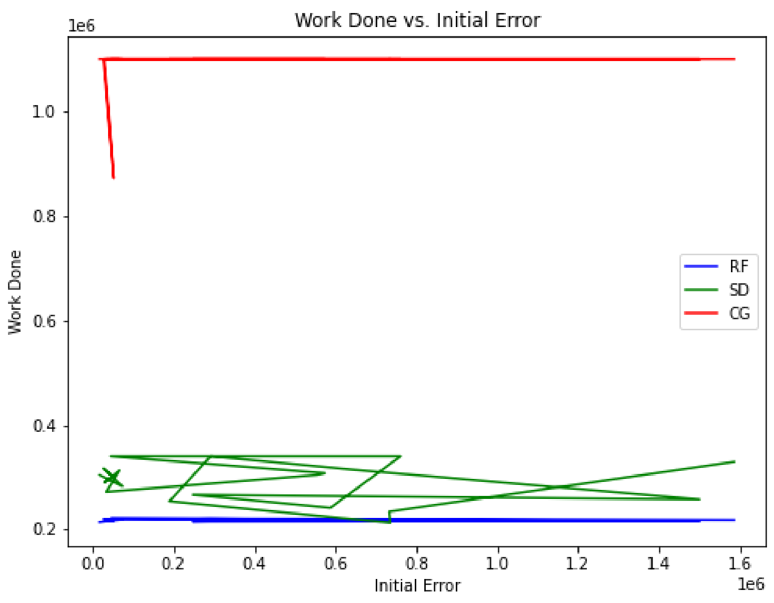


Figure 66: Total Work against Initial Guess

If we increase the number of experiments to a 100, similar plots are observed again:

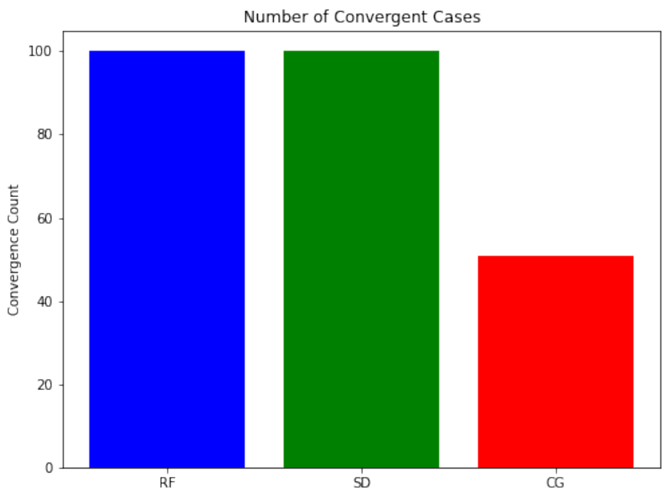


Figure 67: Convergence Reliability

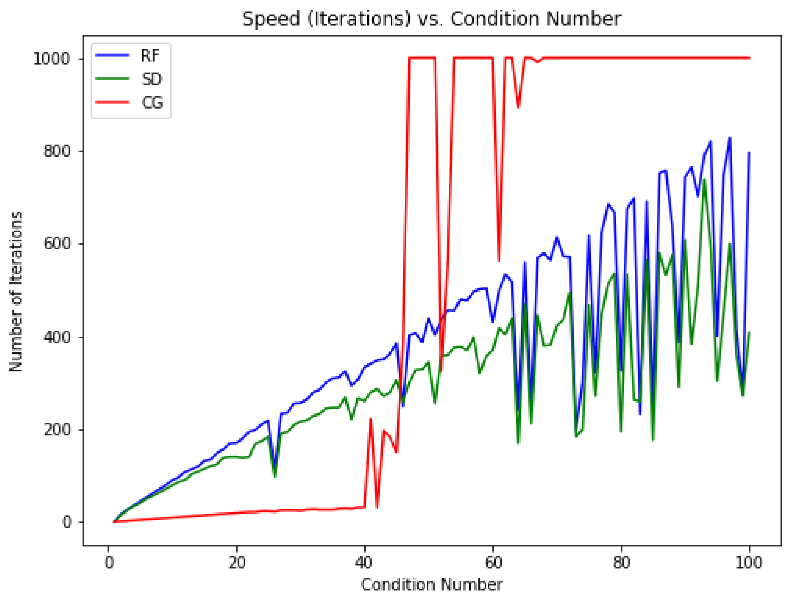


Figure 68: Speed against Spectra

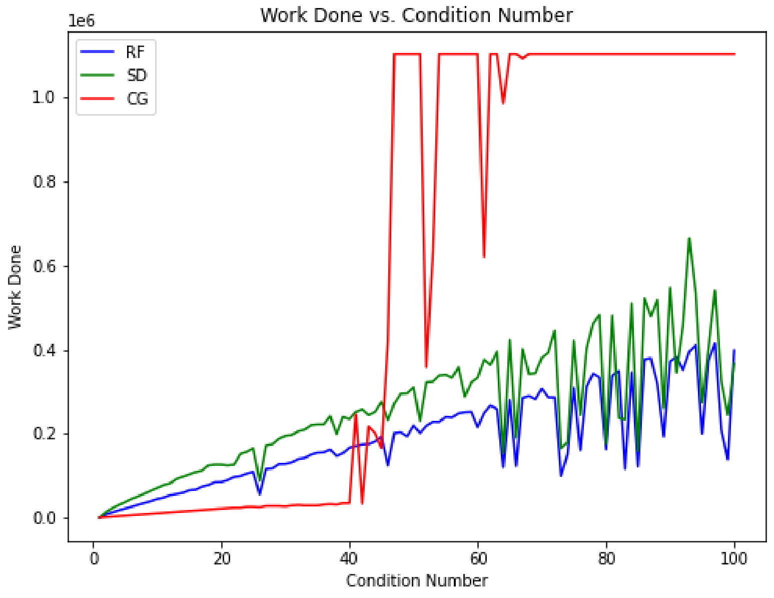


Figure 69: Total Work against Spectra

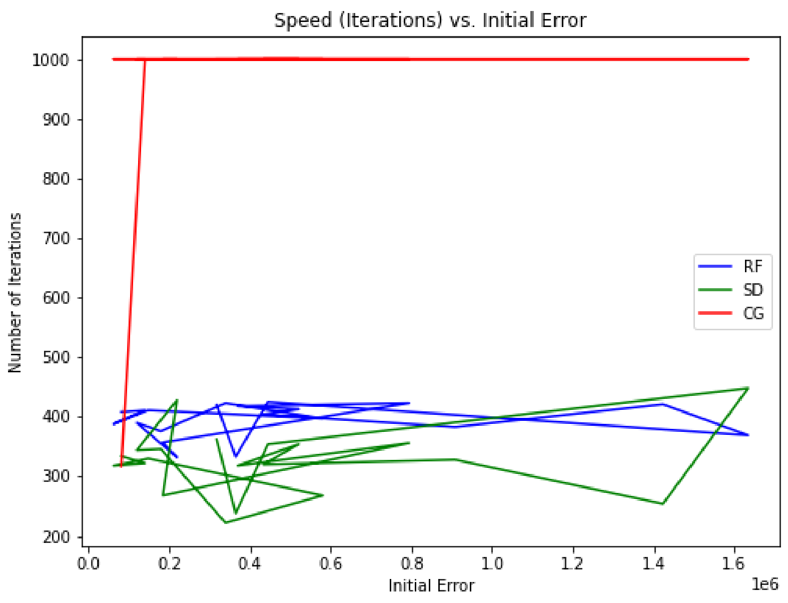


Figure 70: Speed against Initial Guess

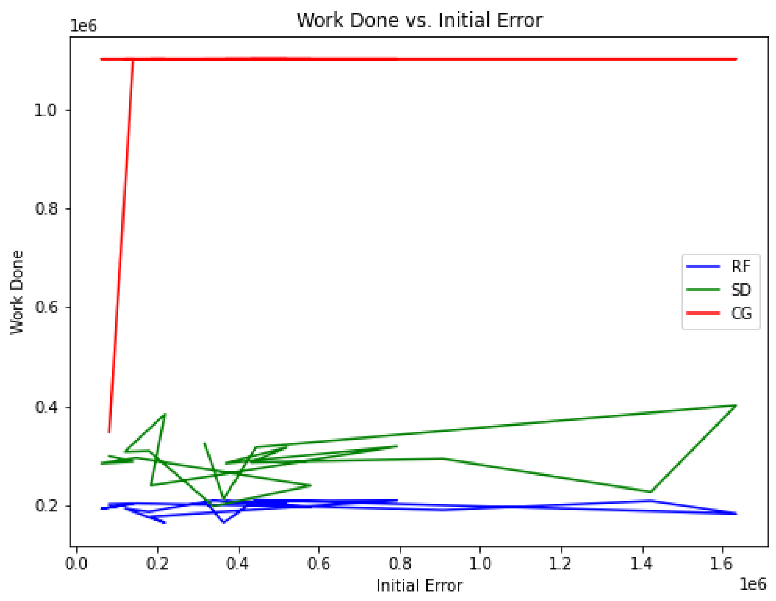


Figure 71: Total Work against Initial Guess

Using the same driver, more experiments may be run by varying different values for different types of plots. Overall, the same conclusions as above can be deduced from different experiments.

Task 2 and 3

For this task we now look at 3 particular stationary methods in the Richardson Family of Methods. These are

- Jacobi
- Gauss-Seidel
- Symmetric Gauss-Seidel

Each of these differ in the pre-conditioner that they use for the update step for x_k (See Appendix). For these methods, we cannot work in the eigen-basis; thus we have to generalize our implementation to matrices. The two functionalities that we will additionally need are matrix-vector products and lower and upper triangular solves. The experimental design, control flow and presentation will be similar that of Task 1. We implement a single function that takes the same inputs as before and in addition a flag indicating which pre-conditioner to use (hence, which method is employed). For this task, the dense solves and matrix-vector products are implemented.

We also have different convergence results for these stationary methods, given by the following theorem:
Theorem
Suppose $A \in \mathbb{R}^{n \times n}$ is symmetric positive definite and the preconditioner P is used.
If $(P + P^T) - A$ is symmetric positive definite, then P^{-1} exists and the iterative method

$$x_{k+1} = x_k + P^{-1} r_k$$

has

- $\rho(G) \leq \|G\|_A < 1 \quad \text{(convergent)}$
- $\|e^{(k+1)}\|_A < \|e^{(k)}\|_A \quad \text{(monotonic)}$

where G is the contractive form of the iterative matrices.

We restrict our analysis to the 8 matrices provided in the assignment, as representatives of certain types of problems. For each matrix, we will then have to compute G for error analysis. The matrices are as follows:

$$A_0 = \begin{bmatrix} 3 & 7 & -1 \\ 7 & 4 & 1 \\ -1 & 1 & 2 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} 3 & 0 & 4 \\ 7 & 4 & 2 \\ -1 & -1 & 2 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} -3 & 3 & -6 \\ -4 & 7 & -8 \\ 5 & 7 & -9 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 4 & 1 & 1 \\ 2 & -9 & 0 \\ 0 & -8 & -6 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} 7 & 6 & 9 \\ 4 & 5 & -4 \\ -7 & -3 & 8 \end{bmatrix}$$

$$A_5 = \begin{bmatrix} 6 & -2 & 0 \\ -1 & 2 & -1 \\ 0 & -\frac{6}{5} & 1 \end{bmatrix}$$

$$A_6 = \begin{bmatrix} 5 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -\frac{3}{2} & 1 \end{bmatrix}$$

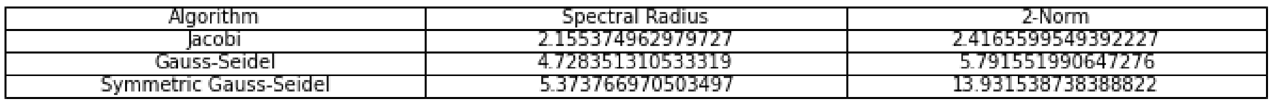
$$A_7 = \begin{bmatrix} 4 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 4 \end{bmatrix}$$

$$A_8 = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

We first conduct manual experiments on each of the matrices to formulate hypotheses and observe behavior through plots. For further insight, note that our theorem ensures convergence for SPD matrices given that the contractive form has spectral radius < 1. These algorithms also converge for systems that are not SPD. This can only be observed numerically in our problem. We then check which matrices are indeed SPD:

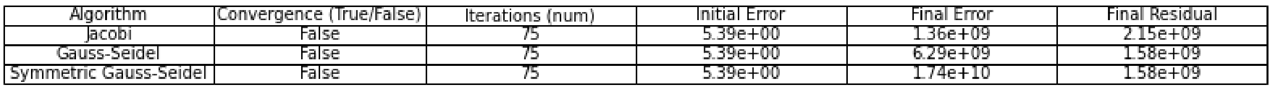
Matrix	Eigenvalues	SPD
A_0	$[10.52, -3.86, 2.34]$	No
A_1	$[(4.16 + 3j), (4.16 - 3j), (0.68 + 0j)]$	No
A_2	$[(-3.87 + 6.88j), (-3.87 - 6.88j), (2.75 + 0j)]$	No
A_3	$[4.03, -9.49, -5.54]$	No
A_4	$[(7.97 + 5.45j), (7.97 - 5.45j), (4.07 + 0j)]$	No
A_5	$[6.47, 2.34, 0.18]$	Yes
A_6	$[5.33, 2.56, 0.11]$	Yes
A_7	$[2.15, 2.59, 3.23, 4.0, 5.85, 5.41, 4.77]$	Yes
A_8	$[3.85, 3.41, 2.77, 2.0, 0.15, 0.59, 1.23]$	Yes

Now we conduct experiments for each matrix (with the solution vector of all 1's and $x_0 = 0$) and present the results.
0.



Algorithm	Spectral Radius	2-Norm
Jacobi	2.155374962979727	2.4165599549392227
Gauss-Seidel	4.728351310533319	5.791551990647276
Symmetric Gauss-Seidel	5.373766970503497	13.931538738388822

Figure 72: Iterative Matrices



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Final Residual
Jacobi	False	75	5.39e+00	1.36e+09	2.15e+09
Gauss-Seidel	False	75	5.39e+00	6.29e+09	1.58e+09
Symmetric Gauss-Seidel	False	75	5.39e+00	1.74e+10	1.58e+09

Figure 73: Results Table

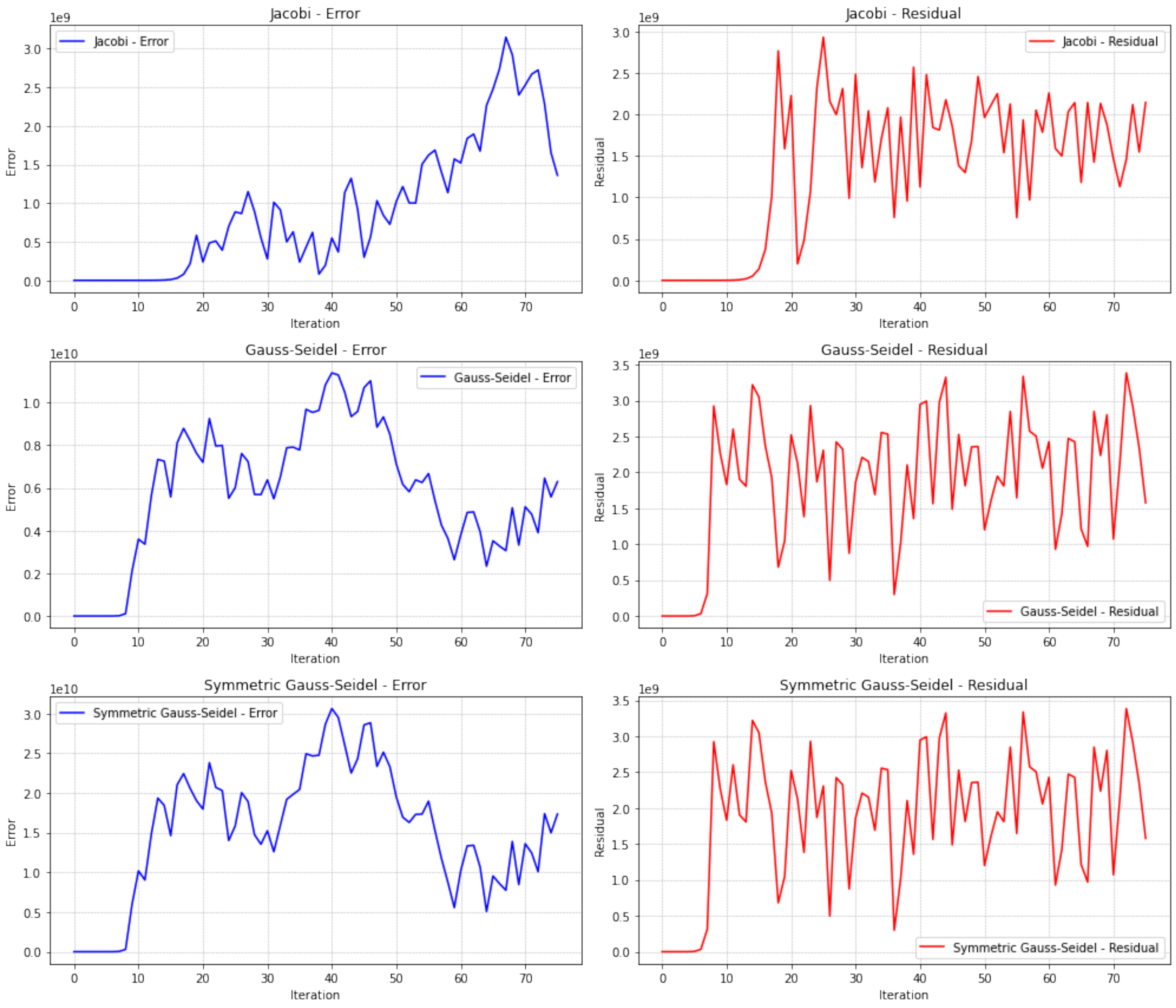
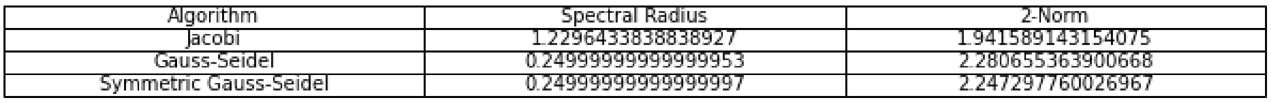


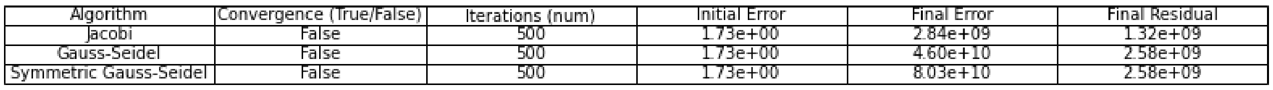
Figure 74: Error and Residual

1.



Algorithm	Spectral Radius	2-Norm
Jacobi	1.2296433838838927	1.941589143154075
Gauss-Seidel	0.24999999999999953	2.280655363900668
Symmetric Gauss-Seidel	0.24999999999999997	2.247297760026967

Figure 75: Iterative Matrices



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Final Residual
Jacobi	False	500	1.73e+00	2.84e+09	1.32e+09
Gauss-Seidel	False	500	1.73e+00	4.60e+10	2.58e+09
Symmetric Gauss-Seidel	False	500	1.73e+00	8.03e+10	2.58e+09

Figure 76: Results Table

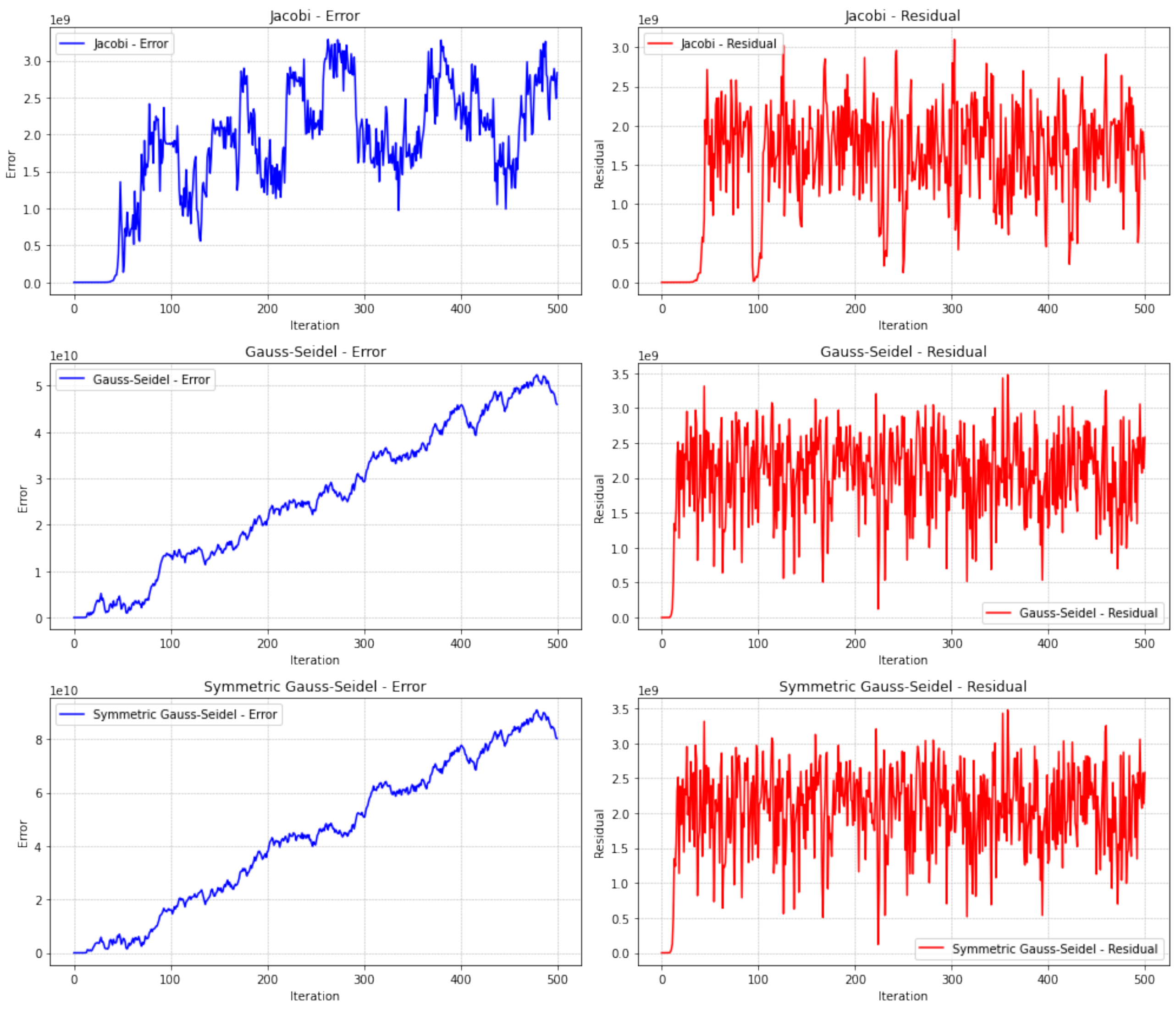
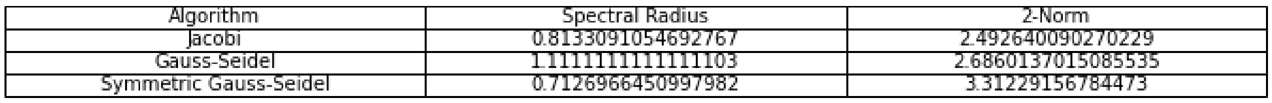


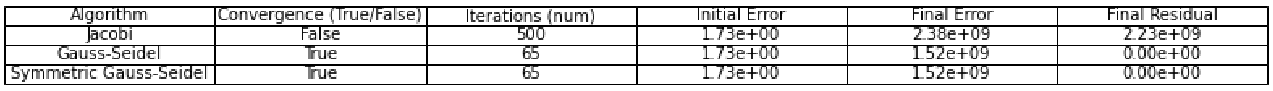
Figure 77: Error and Residual

2.



Algorithm	Spectral Radius	2-Norm
Jacobi	0.8133091054692767	2.492640090270229
Gauss-Seidel	1.1111111111111103	2.6860137015085535
Symmetric Gauss-Seidel	0.7126966450997982	3.31229156784473

Figure 78: Iterative Matrices



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Final Residual
Jacobi	False	500	1.73e+00	2.38e+09	2.23e+09
Gauss-Seidel	True	65	1.73e+00	1.52e+09	0.00e+00
Symmetric Gauss-Seidel	True	65	1.73e+00	1.52e+09	0.00e+00

Figure 79: Results Table

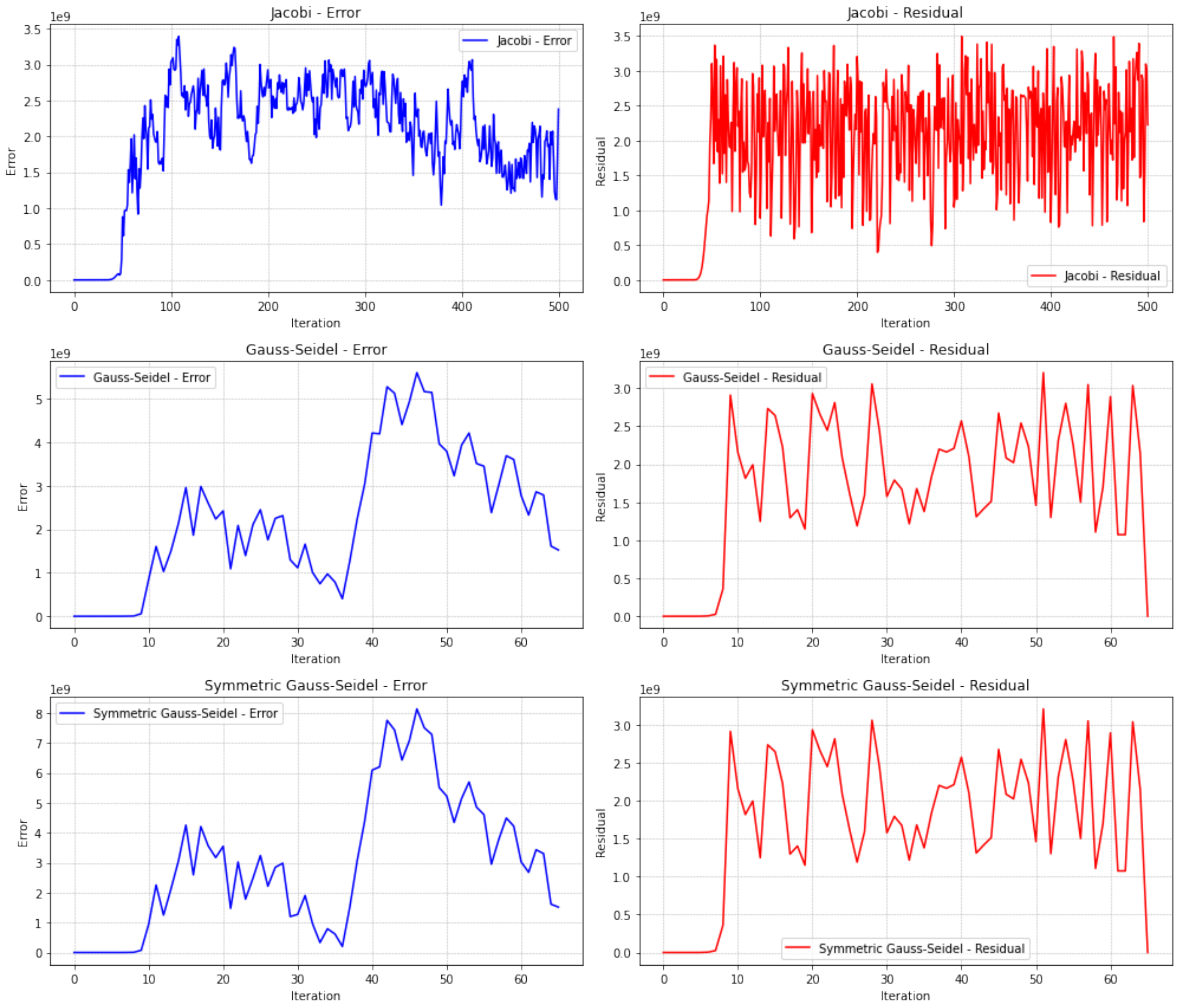
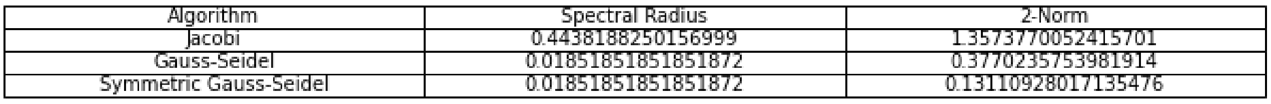


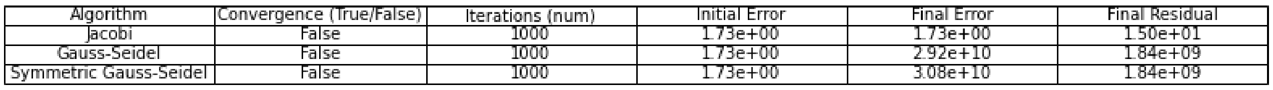
Figure 80: Error and Residual

3.



Algorithm	Spectral Radius	2-Norm
Jacobi	0.4438188250156999	1.3573770052415701
Gauss-Seidel	0.01851851851851872	0.3770235753981914
Symmetric Gauss-Seidel	0.01851851851851872	0.13110928017135476

Figure 81: Iterative Matrices



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Final Residual
Jacobi	False	1000	1.73e+00	1.73e+00	1.50e+01
Gauss-Seidel	False	1000	1.73e+00	2.92e+10	1.84e+09
Symmetric Gauss-Seidel	False	1000	1.73e+00	3.08e+10	1.84e+09

Figure 82: Results Table

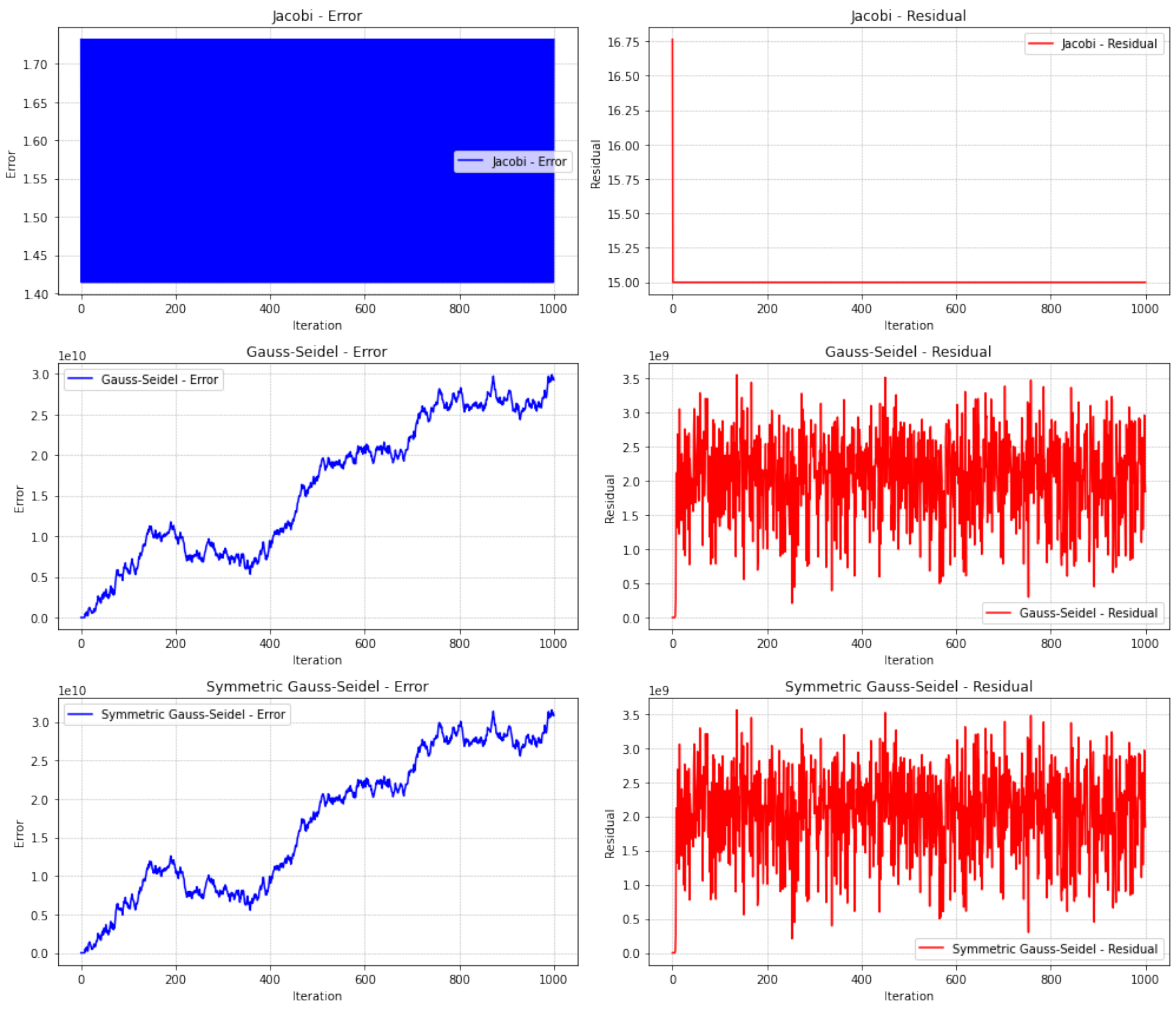
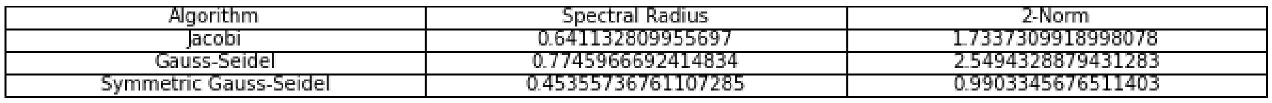


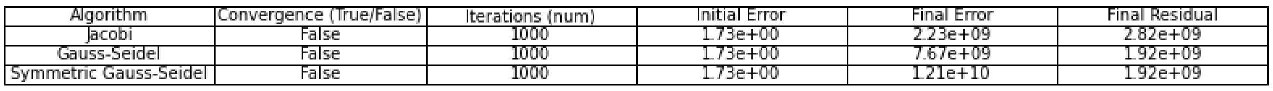
Figure 83: Error and Residual

4.



Algorithm	Spectral Radius	2-Norm
Jacobi	0.641132809955697	1.7337309918998078
Gauss-Seidel	0.7745966692414834	2.5494328879431283
Symmetric Gauss-Seidel	0.45355736761107285	0.9903345676511403

Figure 84: Iterative Matrices



Algorithm	Convergence (True/False)	Iterations (num)	Initial Error	Final Error	Final Residual
Jacobi	False	1000	1.73e+00	2.23e+09	2.82e+09
Gauss-Seidel	False	1000	1.73e+00	7.67e+09	1.92e+09
Symmetric Gauss-Seidel	False	1000	1.73e+00	1.21e+10	1.92e+09

Figure 85: Results Table

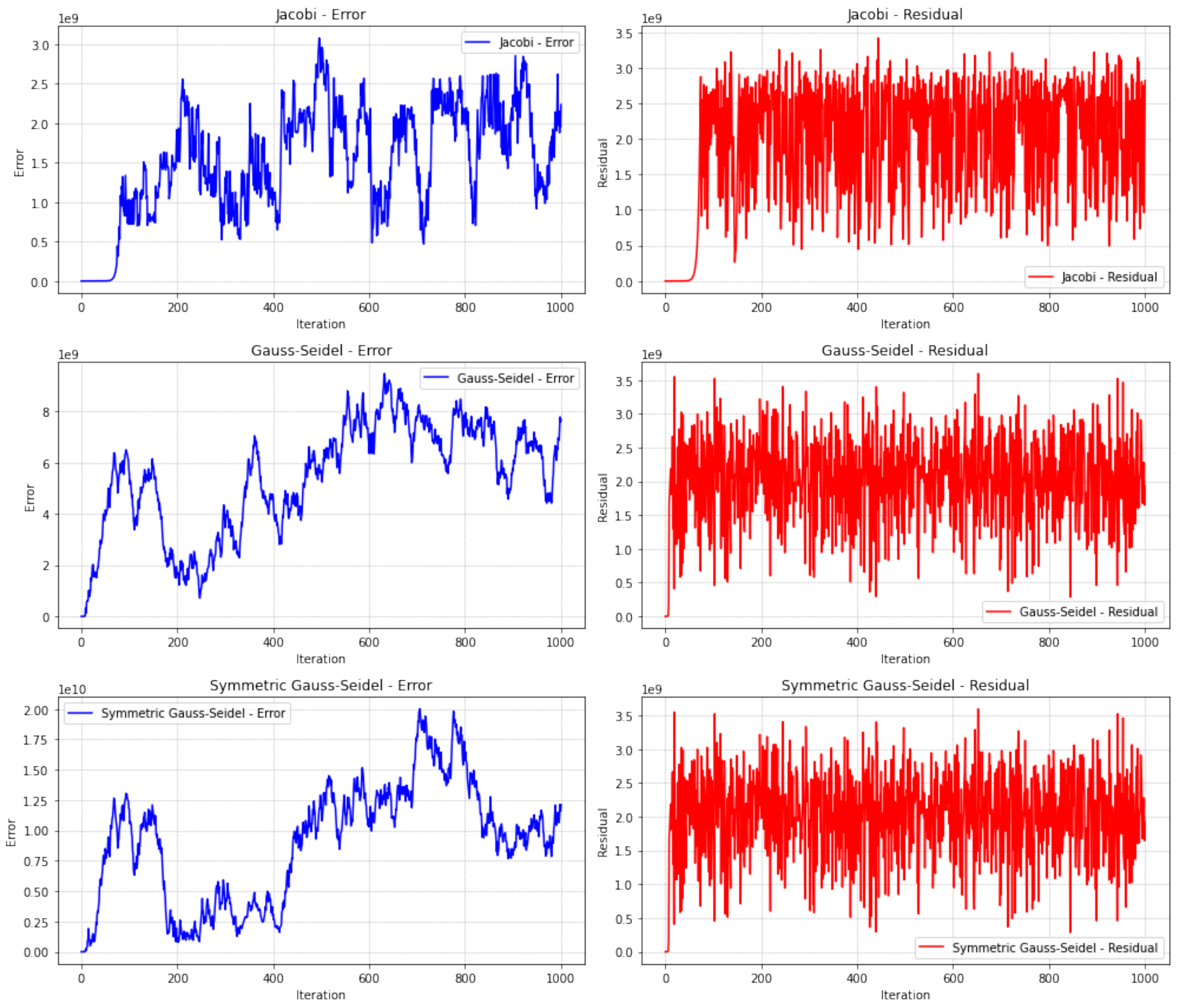


Figure 86: Error and Residual

For the rest of the matrices, we get convergence. For the larger set of experiments, we get the same results for the divergent matrices. For some of the experiments, Jacobi seems to get stuck immediately. There's also much more numerical sensitivity for these matrices due to the solves. The code for the last few problems had to be modulated to ensure that there were no runtime errors. For the experiments we observe the same qualitative effects.
We use the same function as before and use the scipy sparse library for added functionality. The implementation of the sparse methods requires us to change matmul to sparse matrix-vector products and also re-implement sparse upper and lower solves. Since the diagonal elements are assumed to be dense and positive, these methods remain unchanged. We can make the following conclusions about large sparse matrices:

- The jacobi preconditioning step remains unchanged for sparse systems.
- Each successive method is faster in terms of number of iterations i.e GS is faster than Jacobi and Symmetric GS is faster than GS.
- The complexity of each algorithm also increased.
- In terms of numerical reliability, the algorithms usually converge and have very similar convergence profiles (in most experiments, all of them converge).

Another observations we made from the experiments is that the diagonal elements control the speed of the the algorithms for each method. For sparsity, the higher the sparsity, the less total work is done with each algorithm.

Conclusions

We have then conducted thorough investigation of all of our hypotheses and presented evidence for each observation and result that we observed. We were able to substantiate all of our theoretical results and prediction, and explain whatever deviation we did observe in terms of finite-precision or other numerical errors. or practical considerations. We presented numerical evidence as visual plots, important metrics and statistical evidence. Each individual problem was handled separately in implementation, and the results were consolidated for this report.

Appendix

Keep in mind that RF is presented in a completely general form and the form of the preconditioner matrix P can be changed to produce the following four algorithms in the Richardson's Family of Methods:

1. Without Preconditioning: $P = I$
2. Jacobi: $P = D$
3. Gauss Seidel: $P = (D - L)$
4. Symmetric Gauss Seidel (Forward and Backward): $P = (D - L)D^{-1}(D - U)$

Algorithm 1 Richardson's Method (RF)

```
1: $A$ is symmetric positive definite
2: Choose $x_0$ arbitrarily
3: Compute $r_0 = b - Ax_0$
4: Set $\alpha$ to ensure convergence
5: **while** not converged **do**
6:     Solve $Pz_k = r_k$
7:     $x_{k+1} \leftarrow x_k + z_k\alpha$
8:     $r_{k+1} \leftarrow r_k - Ar_k\alpha$
9: **end while**
```

Algorithm 2 Steepest Descent (SD) Method

```
1: $A$ is symmetric positive definite
2: Choose $x_0$ arbitrarily
3: Compute $r_0 = b - Ax_0$, $v_0 = Ar_0$
4: while not converged do
5:     $\alpha_k \leftarrow \frac{r_k^T r_k}{r_k^T v_k}$
6:     $x_{k+1} \leftarrow x_k + r_k \alpha_k$
7:     $r_{k+1} \leftarrow r_k - v_k \alpha_k$
8:     $v_{k+1} \leftarrow Ar_{k+1}$
9: end while
```

Algorithm 3 Conjugate Gradient (CG) Method

```
1: $x_0$ arbitrary; $r_0 = b - Ax_0$; $d_0 = r_0$; $\sigma_0 = r_0^T r_0$
2: for $k = 0, 1, \ldots$ do
3:     $v_k \leftarrow Ad_k$
4:     $\mu_k \leftarrow d_k^T v_k$
5:     $\alpha_k \leftarrow \sigma_k / \mu_k$
6:     $x_{k+1} \leftarrow x_k + \alpha_k d_k$
7:     $r_{k+1} \leftarrow r_k - \alpha_k v_k$
8:     $\sigma_{k+1} \leftarrow r_{k+1}^T r_{k+1}$
9:     $\beta_k \leftarrow \sigma_{k+1} / \sigma_k$
10:    $d_{k+1} \leftarrow r_{k+1} + \beta_k d_k$
11: end for
```