An Analysis on Direct and Iterative Techniques to Solve Linear Systems

Ratislav Krylov Caleb Lewis

April 17, 2018

1 Introduction

The need to solve the problem $Ax = b$ comes up in many real-world scenarios. As a result, there have been many solutions of varying efficiency that have been employed to solve it. In this paper, we look at 9 of these methods and compare their complexity and accuracy.

2 Methods

2.1 Gaussian Elimination

Gaussian Elimination is one of the oldest methods used to solve systems of linear equations. It utilizes elementary row operations to convert the augmented system into an upper triangular matrix and then solve for each unknown x through backwards substitution. In theory, Gaussian Elimination finds exact values for the system of linear equations, but due to physical limitations, its not really used in practice. There is always rounding in numerical computations and Gaussian Elimination is quite susceptible to them. Additionally its computational of $O(n^3)$ is quite high and makes impractical to use for large matrices.

While this algorithm's complexity cannot be reduced, there exist techniques to deal with the effect that rounding has on Gaussian Elimination.

2.1.1 Partial Pivoting

The round off errors accumulate the most when pivot is a lot smaller than other that it eliminates within its column. A simple way to deal with this is to find the largest value within the column before performing elimination and do a row swap if its greater than the current pivot. This way when eliminations are performed the rounding effect will not be as dramatic because the elimination values are smaller.

2.1.2 Scaled Partial Pivoting

Partial Pivoting may not always work when the rows are not scaled properly to each other. The values of one row can be larger than those of another and the partial pivoting will not deal with rounding error properly. This can be easily fixed through dividing all the rows by their largest element.

While Partial Pivoting and Scaling are nice and easy techniques to deal with rounding errors, they add more comparisons to an already computationally costly algorithm. But since the main

reason why anyone would use Gaussian Elimination is to have as precise solutions as possible, these additional comparisons can be worth it.

2.2 Iterative Refinement

Iterative Refinement is an algorithm used to take advantage of all computational digits available and to improve the accuracy of an acquired solution. Given x_{approx} and y the approximate solution to the system $A\mathbf{y} = \mathbf{r}$ where r is the residual that can be calculated through x_{approx}, we can use the fact that $y \approx x - x_{approx}$ and calculate a better estimate $x_{approx} + y$ for x. It is often used together with Gaussian Elimination since it can theoretically calculate the exact solution if not for rounding errors. Combined with Iterative Refinement it actually allows us to compute the exact answer for the system of linear equations upto the rounding digits.

2.3 Jacobi Iterative

The first of the iterative methods that we consider is the Jacobi Iterative method. For a diagnolly dominant matrix A,

$$\text{let} \quad x^{(0)} \in \mathbb{R}^n, \quad D = diag(A), \quad R = A - D \tag{1}$$

$$x^{k+1} = D^{-1}(b - Rx^{(k)}) \tag{2}$$

We repeat the above equation for $k = 0, 1, ...$ until convergence. The stopping critia is:

$$\frac{\left\|x^{(k)} - x^{(k-1)}\right\|}{x^{(k)}} \leq \epsilon \tag{3}$$

For our experiments, we set $\epsilon = .001$ as we found it was enough to show the differences in accuracies and behaviors between this and other methods. The following is a table of the results after running the method on the examples for a various number of iterations:

Example 2	10 0.52058755	20 0.17910029	30 0.20600456	40 0.02831351
Example 3	10 0.46132609	20 0.04399548	30 .00419573634	40 .000400136598

Notice that examples 1 and 4 have been left out - those are not diagonally dominant, therefore the Jacobi method is not able to converge.

2.4 Gauss-Seidel

Next is the Gauss-Seidel method. This method has similar limitations as the Jacobi method, but performs better in practice. First we define L to be the lower triangle part of A and:

$$letx^{(0)} \in \mathbb{R}^n, \quad U = A - L \tag{4}$$

the method is the iteration

$$x^{k+1} = L^{-1}(b - Ux^{(k)}) \tag{5}$$

for $k = 0, 1, ...$ until the convergence or stopping criteria. We will use the same criteria as we did in the Jacobi method.

Example 2	10 .1327548	17 .01452508	20 .00144870	30 .00292537
Example 3	10 .02302011	13 .00899223	20 .00033498	30 .00000190

Examples 1 and 4 have been left out due to the same reasons as the Jacobi method - this method does not converge on matrices that are not diagnolly dominant. The method converges at 17 and 13 iterations for examples 2 and 3 respectively.

2.5 Successive Over-Relaxation

The Successive Over-Relaxation (SOR) technique is an improvement on the Gauss-Seidel method by introducing a **relaxation** parameter to over-correct for the error at each step. The iterative method is given by:

$$x^{k+1} = (D - \omega L)^{-1}[(1 - \omega)D + \omega U]x^{(k-2)} + \omega(D - \omega L)-1b \tag{6}$$

where $D, -L,$ and $-U$ are the diagnol, strict lower, and string upper triangular parts of A respectively. There is an optimal value of $0 < \omega < 2$ where this method converges the fastest. SOR is only stable where $\omega > 2$ and is under-relaxed where $0 < \omega < 1$.

Example 2	10 .04903008	14 .00560935	20 .00294247	30 .00307403485
Example 3	8 .00101802	10 .00005610	20 .000000005	30 .00000000

Note that SOR converges at 14 & 8 iterations for examples 2 and 3 respectively. It does indeed converge faster than the Gauss Seidel method as it converges at 17 and 13 iterations and the errors between each iteration are quite small.

2.6 (Preconditioned) Conjugate Gradient Method (PCG)

The Preconditioned Conjugate Gradient (PCG) Method is the last of the methods we will explore. Given a preconditioner $C, x^{(0)}, r^{(0)} = b - Ax^{(0)}, w^{(0)} = C^{-1}r^{(0)}, v^{(1)} = C^{-T}w^{(0)}$, it is described by

the following:

$$t_k = \frac{\langle w^{(k-1)}, w^{(k-1)} \rangle}{\langle v^{(k)}, Av^{(k)} \rangle}$$

$$x^{(k)} = x^{(k-1)} + t_k v^{(k)}$$

$$r^{(k)} = r^{(k-1)} - t_k A v^{(k)}$$

$$w^{(k)} = C^{-1} r^{(k)}$$

$$s_k = \frac{\langle w^{(k)}, w^{(k)} \rangle}{\langle w^{(k-1)}, w^{(k-1)} \rangle}$$

$$v^{(k+1)} = C^{-T} w^{(k)} + s_k v^{(k)}$$

PCG is a modified version of gradient descent that corrects for the direction and magnitude problem by using the fact that the residual of the matrix is also the steepest descent direction and its orthogonality with off-diagnol vectors. With this, we can solve for scalars t_k, s_k to maximize direction and magnitude for each iteration. This is further optimized by preconditioner C.

Example 2	10 0303791568]	21 .00298286092
Example 3	4 .000000000	10 .000000000
Example 4	1 .0	10 .0

3 Numerical experiments

3.1 Example 1

3.2 Example 2

3.3 Example 3

3.4 Example 4

4 Discussion

This is a major part for this project. It should constitute your findings and thoughts. Based on the tests you have, you want to comment on the performance of these methods and how would you suggest to use in practice. Have extensive discussions with your team members and give detailed reasonings for your claims. You can cite books, papers, or other