# Project 2: Solving Linear Equations

CS 3010 - Numerical Methods

Summer 2020

Instructor: Dr. Amar Raheja

Computer Science Department

California State Polytechnic University, Pomona

Submitted On: 25-JUL-2020

Submitted By:

Damian Ugalde

**Instructions:**

Write a program that asks the user for the number of linear equations to solve (let's say n <=10) using the following three methods. Ask the user to first enter the number of equations and then give them the choice to enter the coefficients from the command line (by asking for each row that includes the b value) or have them enter a file name which has the augmented coefficient matrix (including the b values) in a simple text file format as seen below for an example of 3 equations :

1) the scaled partial pivoting method for Gaussian elimination.

2) Jacobi iterative method (this method will require an additional input the user has to enter, which is the desired error). You can allow user to enter the starting guess x vector values or randomly choose some value in the code

3) Gauss-Seidel method (again, the user must enter the desired error that will be used for stopping condition). You can allow user to enter the starting guess x vector values or randomly choose some value in the code

If the iterative methods don't achieve the desired error in 50 iterations, end the iterative method and print out the x column vector for the final iteration.

E.g. the contents of a file for 3 linear equations 2x-y = 1, -x+3y-z=8, -y+2z=-5 will be:

2 -1 0 1

-1 3 -1 8

0 -1 2 -5

The final output of your program should be the solution in the following format:

x=2

y=3

z=-1

Also, please print the intermediate matrices for the Gaussian Elimination method (1) and the values of the column vector x after each iteration for the other two methods. Show snapshots of all the 3 methods solving a set of equations in your report. Comment on the convergence of each method for the set of equations you used for testing it.

For the iterative methods, please ask the user to enter a value for error below which you will stop the iterative method. Calculate L2 norm for error after each iteration and when this value is less than the error the user entered, then stop the iterations.

Upload your code, executable and report

Extra credit: Test your program with large values of n, say like 20, 50, 100 etc. Generate the coefficients randomly for such large equations. Comments on the time taken by your program as the number of equations increase. Do you run into any other problems? Write a one page report on this extra credit part. Show a graph of how the time increases as the number of equations increase (3 curves on this graph, one for each method)

Doing the extra credit will help you bump your grade up by one grade level. So, a B will go to B+ or an A- to an A and so on and so forth.

**Test Matrix:**

The test matrix used is provided in the testMatrix.txt file:

$$\begin{aligned} 2x - 1y + 0z &= 1 \\ -1x + 3y - 1z &= 8 \\ 0x - 1y + 2z &= -5 \end{aligned}$$

With a solution of:

$$\begin{aligned} x &= 2 \\ y &= 3 \\ z &= -1 \end{aligned}$$

**User Interface:**

The program lets the user type a matrix:

```
Would you like to enter your own input, random or use a file?
Type 'input' to type a matrix.
Type 'random' to get a random matrix.
Type 'file' to use a file.
Your choice: input
Enter the number of equations/variables: 3
Please enter equation #1, separating each term with a space.
2 -1 0 1
Please enter equation #2, separating each term with a space.
-1 3 -1 8
Please enter equation #3, separating each term with a space.
0 -1 2 -5
The Linear System is:
2.0 -1.0 0.0 1.0
-1.0 3.0 -1.0 8.0
0.0 -1.0 2.0 -5.0

What method would you like to use?
Type 'pivoting' for scaled partial pivoting for Gaussian elimination method.
Type 'jacobi' for Jacobi iterative method.
Type 'gauss' for the Gauss-Seidel method.
Your choice:
```

It also lets the user specify a file. The file looks like so:

testMatrix.txt

```
2 -1 0 1

-1 3 -1 8

0 -1 2 -5
```

With the input:

```
Would you like to enter your own input, random or use a file?
Type 'input' to type a matrix.
Type 'random' to get a random matrix.
Type 'file' to use a file.
Your choice: file
What is the name of the file you want to use? Don't forget to type the file extension!
Your file name: testMatrix.txt
Opening testMatrix.txt
File opened, input found as:
2.0 -1.0 0.0 1.0
-1.0 3.0 -1.0 8.0
0.0 -1.0 2.0 -5.0
What method would you like to use?
Type 'pivoting' for scaled partial pivoting for Gaussian elimination method.
Type 'jacobi' for Jacobi iterative method.
Type 'gauss' for the Gauss-Seidel method.
Your choice:
```

Or the user can select a random matrix:

```
Would you like to enter your own input, random or use a file?
Type 'input' to type a matrix.
Type 'random' to get a random matrix.
Type 'file' to use a file.
Your choice: random
Enter the matrix size that you want to use as an integer
3
The Linear System is:
9.657476176669343 -0.1310638506692392 -2.1864423892123988 9.521278100306118
-3.5007485658723443 -2.4529425243445218 7.69210464796787 1.2704928921943743
4.43193061274806 -3.037162685714023 7.220048649530874 -1.5923645624805278

What method would you like to use?
Type 'pivoting' for scaled partial pivoting for Gaussian elimination method.
Type 'jacobi' for Jacobi iterative method.
Type 'gauss' for the Gauss-Seidel method.
Your choice:
```

For the iterative methods, the user can also select a starting value of x and it works as follows:

```
What would you like to do for the starting value of x?
Type '0' to start with zeroes.
Type 'random' to start with random values.
Type 'input' to provide the starting values.
0
Starting with initial guess at 0.
```

```
random
Starting with random initial guess.
Enter a minimum value: -5
enter a maximum value: 5
```

```
input
Starting with user guess.
Enter the initial guess matrix with size 3, separating each term with a space.
0 1 2
```

**Scaled Partial Pivoting Method for Gaussian Elimination:**

This method is the most precise, since it gets an exact solution (Except for that of round-off error caused by floating-point representation). However, this precision comes at a time cost. The general outline is as follows:

```
    private static int[] getIndexArray(LinearSystem system) {
        ...

        for(i = 0; i < system.n; i++){
            ...
            for(j = 0; j < system.n; j++){...}
            ...
        }

        for(k = 0; k < system.n-1; k++){
            ...
            for(i = k; i < system.n; i++){
                ...
            }
            ...

            for(i = k+1; i < system.n; i++){
                ...
```

```
                for(j = k+1; j < system.n; j++){...}
            }

            ...
        }

        ...
    }

    private static void forwardElimination(LinearSystem system, int[] l){
        for(int k = 0; k < system.n-1; k++){
            for(int i = k+1; i < system.n; i++){...}
        }
    }

    private static double[] backSubstitution(LinearSystem system, int[] l){
        ...

        for(k = 0; k < system.n-1; k++){
            for(i = k + 1; i < system.n; i++){...}
        }

       ...

        for(i = system.n-2; i >=0; i--){
                    ...
            for(int j = i + 1; j < system.n; j++){...}
                    ...
        }

        ...

    }
}
```

The getIndexArray() method uses two nested $n^2$ loops and an $n$-size for loop with an $n^2$ and an $n$-sized for loop inside, for a total of $O(n^2 + n(n + n^2) = O(n^2 + n^2 + n^3) = O(n^3 + 2n^2)$. The forwardElimination() method uses two nested $n^2$ loops for a total of $O(n^2)$. The backSubstitution() method uses two sets of nested for loops, each with $n^2$ time complexity, for a total of $O(2n^2)$. This equals a total time complexity $O(n^3 + 2n^2 + n^2 + 2n^2) = O(n^3 + 5n^2)$ . Since this is an $O(n^3)$ method, it is an extremely inefficient algorithm because there are multiple $n^2$ loops and an $n^3$ loop inside of this method. For applications that have to be executed many times and can jeopardize minimal precision for exponential time, iterative methods are better.

Example solution:

```
Your choice: pivoting
Solving with Scaled Partial Pivoting method for Gaussian Elimination.
2.0 -1.0 0.0 1.0
-0.5 2.5 -1.0 8.0
0.0 -1.0 2.0 -5.0

2.0 -1.0 0.0 1.0
-0.5 2.5 -1.0 8.0
0.0 -0.4 1.6 -5.0



The Solution is:
2.55
4.1
1.2500000000000002
```

**Jacobi Iterative Method:**

This iterative method has a general outline as follows:

```
public static double[] solve(LinearSystem system, double[] x, double error){
	...
	for(k = 0; k < kmax; k++){
		y = Arrays.copyOf(x, x.length);
		for(i = 0; i < n; i++){
			...
			for(j = 0; j < n; j++){...}
			...
		}

		if(vectorMagnitude(x) - vectorMagnitude(y) < error)
			...
	}
	...

}

public static double vectorMagnitude(double[] x){
	...
	for(double next : x){
		sum += next * next;
	}

	return Math.sqrt(sum);
}
```

Every part is encased by a loop with maximum iterations. In this case, 50. Inside, the array copy takes one $n$ loop. The nested for loops inside take up $O(n^2)$ complexity. The vector

magnitude takes a complexity of $O(n)$ for the addition and multiplication, and $O(\sqrt{n})$ to calculate the square root. Each vector magnitude takes up $O(n+\sqrt{n})$. Since it must be calculated twice, it uses $O(n+\sqrt{n})$. The total complexity is $O\left(50\left(n+n^2+2\left(n+\sqrt{n}\right)\right)\right) = O\left(50\left(n^2+3n+2\sqrt{n}\right)\right)$. This method is very efficient for larger sizes of $n$ compared to scaled partial pivoting, since it is $O(n^2)$, but the 50 dominates for small $n$.

Example solution:

```
Your choice: jacobi
Solving with Jacobi Iterative Method.
What is the maximum error?
0.01
What would you like to do for the starting value of x?
Type '0' to start with zeroes.
Type 'random' to start with random values.
Type 'input' to provide the starting values.
0
Starting with initial guess at 0.
0.5 0.0 0.0
0.5 2.6666666666666665 0.0
0.5 2.6666666666666665 -2.5
1.8333333333333333 2.6666666666666665 -2.5
1.8333333333333333 2.0 -2.5
1.8333333333333333 2.0 -1.1666666666666667


The Solution is:
1.8333333333333333
2.0
-1.1666666666666667
```

**Gauss-Seidel Method:**

This iterative method is very similar to Jacobi, but this one uses the updated value of x as soon as it is available. Therefore, it converges a lot quicker than Jacobi. The general outline follows:

```
public static double[] solve(LinearSystem system, double[] x, double error){
	...
	for(k = 0; k < kmax; k++){
		y = Arrays.copyOf(x, x.length);
		for(i = 0; i < n; i++){
			...
			for(j = 0; j < i; j++){...}
			for(j = i + 1; j < n; j++){...}
			...
		}
		if(vectorMagnitude(x) - vectorMagnitude(y) < error)
			...

	}
	...
}

public static double vectorMagnitude(double[] x){
	...
	for(double next : x){
		sum += next * next;
	}
	return Math.sqrt(sum);

}
```

Similar to Jacobi, every part is encased by a loop with maximum iterations. In this case, 50. Inside, the array copy takes one $n$ loop. The nested for loops inside take up $O(n^2)$ complexity. Because the two inner-most loops each only take $\frac{n}{2}$ iterations, since each covers only half the array. As discussed for Jacobi, the vector magnitude must be calculated twice with time $O(n+\sqrt{n})$. The total complexity is $O\left(50\left(n+n^2+2\left(n+\sqrt{n}\right)\right)\right) = O\left(50\left(n^2+3n+2\sqrt{n}\right)\right)$. This method is very efficient for larger sizes of $n$ compared to scaled partial pivoting, since it is $O(n^2)$, but the 50 dominates for small $n$. This method has the same time complexity as Jacobi, but it converges a lot quicker because it uses the newest value for each $x_i$ available instead of looking at the previous iteration.

Example solution:

```
Your choice: gauss
Solving with Gauss-Seidl Method.
What is the maximum error?
0.01
What would you like to do for the starting value of x?
Type '0' to start with zeroes.
Type 'random' to start with random values.
Type 'input' to provide the starting values.
0
Starting with initial guess at 0.
0.5 2.8333333333333335 -1.0833333333333333
1.9166666666666667 2.944444444444444 -1.027777777777778
1.972222222222222 2.981481481481481 -1.0092592592592595
1.9907407407407405 2.993827160493827 -1.0030864197530864


The Solution is:
1.9969135802469136
2.997942386831275
-1.0010288065843624

C:\Users\Pilot\Desktop\Solving-Linear-Equations>
```