Numerical Analysis
homework 04: Linear Iterative Methods

Due on Tuesday, March 28, 2017

102061149 Fu-En Wang

1 Introduction

To solve such linear system:

$$Ax = b \tag{1}$$

We had used **LU Decomposition** to get x in previous homework. In this project, we will solve it with the following iterative methods:

1. **Jacobi Method**
2. **Gauss-Seidel Method**
3. **Symmetric Gauss-Seidel Method**

To evaluate the performance of three method, we will use Question.4 in previous homework(20 resistors at each side).

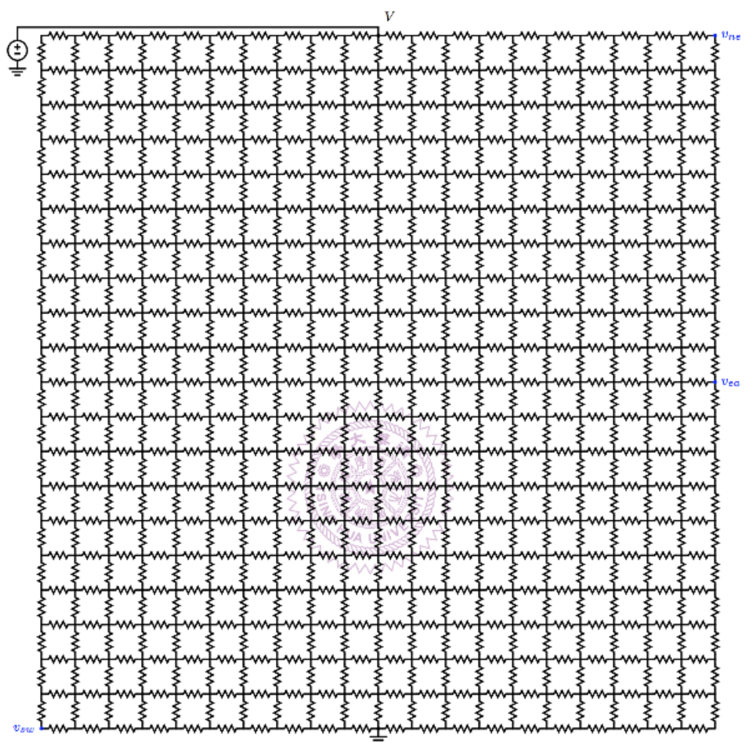

Figure 1: Simple resistor network

To calculate the error, we will use the following error formula:

1. $\|x\|_1 = \sum_{i=1}^{n} |x_i|$
2. $\|x\|_2 = (\sum_{i=1}^{n} x_i^2)^{\frac{1}{2}}$
3. $\|x\|_\infty = max_{i=1}^{n} |x_i|$

2 Implementation

2.1 Jacobi Method

Algorithm 1 Jacobi Method

```
for it ∈ {1, ..., maxIter} do
    lastX = X
    for i ∈ {1, ..., N} do
        sum = 0
        for j ∈ {1, ..., N} do
            if i ≠ j then
                sum += A[i][j] * lastX[j]
            end if
        end for
        x[i] = (1 / A[i][i]) * (b[i] -sum)
    end for
    if Error of (lastX - x) ≤ tol then
        break
    end if
end for
```

2.2 Gauss-Seidel Method

Algorithm 2 Gauss-Seidel Method

```
for it ∈ {1, ..., maxIter} do
    lastX = X
    for i ∈ {1, ..., N} do
        sum1 = 0
        sum2 = 0
        for j ∈ {1, ..., i-1} do
            sum1 += A[i][j] * x[j]
        end for
        for j ∈ {i+1, ..., N} do
            sum2 += A[i][j] * lastX[j]
        end for
        x[i] = (1 / A[i][i]) * (b[i] - sum1 - sum2)
    end for
    if Error of (lastX - x) ≤ tol then
        break
    end if
end for
```

2.3 Symmetric Gauss-Seidel Method

Algorithm 3 **Symmetric Gauss-Seidel Method**

```
for it ∈ {1, ..., maxIter} do
    lastX = X
    for i ∈ {1, ..., N} do
        sum1 = 0
        sum2 = 0
        for j ∈ {1, ..., i-1} do
            sum1 += A[i][j] * x[j]
        end for
        for j ∈ {i+1, ..., N} do
            sum2 += A[i][j] * lastX[j]
        end for
        x[i] = (1 / A[i][i]) * (b[i] - sum1 - sum2)
    end for
    for i ∈ {N, ..., 1} do
        sum1 = 0
        sum2 = 0
        for j ∈ {1, ..., i-1} do
            sum1 += A[i][j] * x[j]
        end for
        for j ∈ {i+1, ..., N} do
            sum2 += A[i][j] * x[j]
        end for
        x[i] = (1 / A[i][i]) * (b[i] - sum1 - sum2)
    end for
    if Error of (lastX - x) ≤ tol then
        break
    end if
end for
```

2.4 Complexity

In the three algorithm, we only use double for-loop for each time of iteration. As a result, the complexity is $O(n^2)$. When the number of iteration is small, iterative method should be faster than LU Decomposition.

3 Discussion

In this project, we will discuss the following topic:

1. How many iteration and tolerance to get accurate V_{ne}, V_{eq} and V_{sw}(error $< 10^{-7}$).
2. Which algorithm has the fastest convegence speed.
3. Is Symmetric Gauss-Seidel better than Gauss-Seidel?
4. Which error calculation method is the best?

3.1 Accuracy

To get accurate V_{ne}, V_{eq} and V_{sw}(error $< 10^{-7}$), we have to adjust tolerance to an appropriate number. In this section, I will use 5 numbers to indicate the accuracy of algorithm:

1. **iteration**(number of iteration)
2. **runtime**(runtime of algorithm)
3. **iter_avg**(average runtime of each iteration)
4. **tolerance**(threshold to stop iteration)
5. **error**(error calculated by three error method)

Table 1, 2, 3 show the experiment result for getting accurate three corner voltage:

Jacobi	Error_1	Error_2	Error_infinite
iteration	11658	11272	10892
runtime(s)	8.55661	8.75754	8.16E+00
iter_avg	0.000734	0.000776929	0.00074904
tolerance	1.95E-08	2.30E-09	3.00E-10
error	9.92E-08	9.92E-08	9.91E-08

Table 1: Jacobi result

Gauss-Seidel	Error_1	Error_2	Error_infinite
iteration	5822	5630	5441
runtime(s)	4.2267	4.20362	3.91E+00
iter_avg	0.00072599	0.00074665	0.00071941
tolerance	3.93E-08	3.27E-09	3.05E-10
error	9.98E-08	9.95E-08	9.91E-08

Table 2: Gauss-Seidel result

Symmetric Gauss-Seidel	Error_1	Error_2	Error_infinite
iteration	2969	2869	2773
runtime(s)	4.36364	4.52277	4.12E+00
iter_avg	0.001469734	0.001576427	0.00148608
tolerance	7.70E-08	6.50E-09	5.95E-10
error	9.96E-08	1.00E-07	9.96E-08

Table 3: Symmetric Gauss-Seidel result

3.2 Error Method

From the experiment in Table 1, 2, 3, we can plot the error with each iteration. Figure 2, 3, 4 show error vs iteration with log-scale:

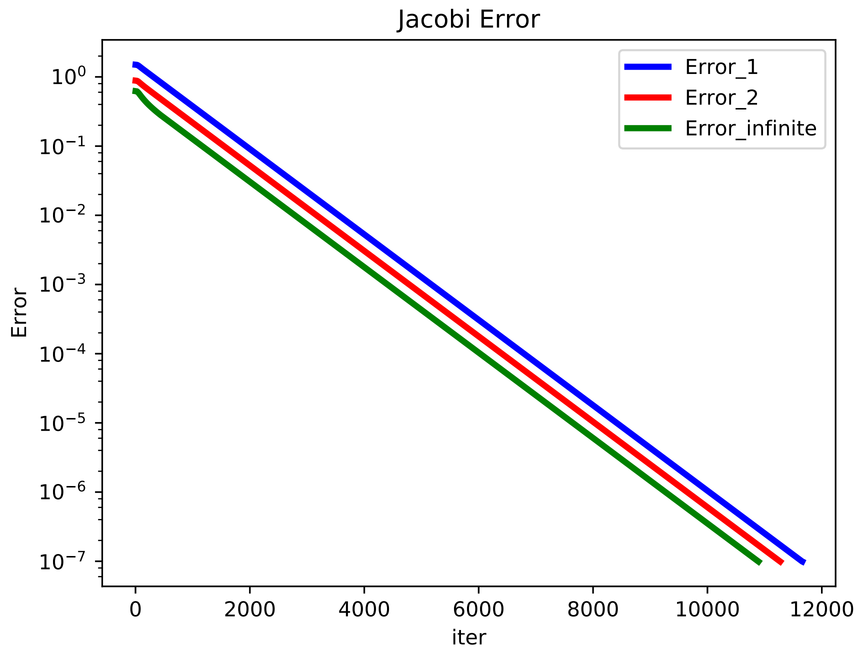

Figure 2: Error drop of Jacobi Method

Figure 3: Error drop of Gauss-Seidel Method

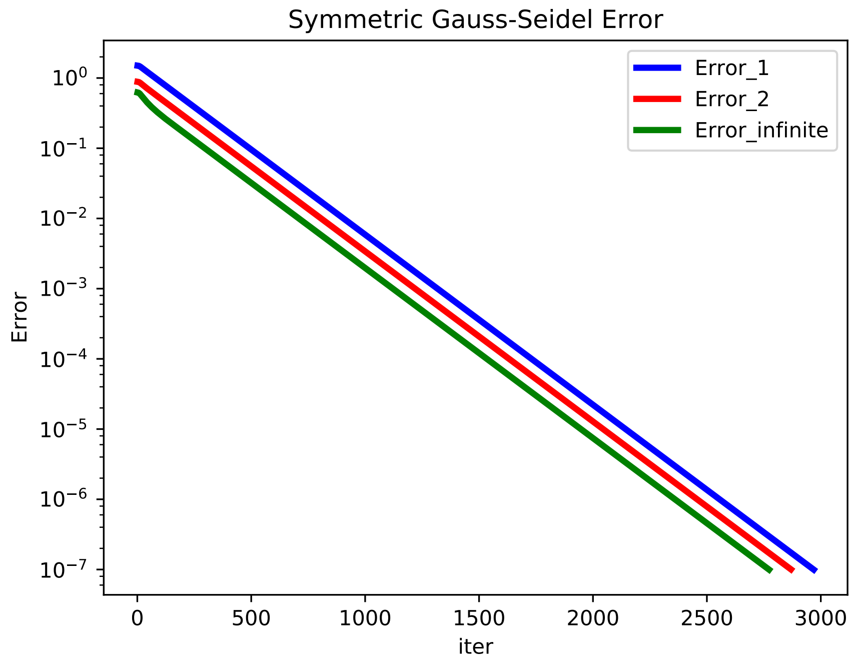

Figure 4: Error drop of Symmetric Gauss-Seidel Method

From Figure 2, 3, 4, we can find that **Error_infinite** has the fastest convergence rate.

3.3 Complexity

From Section 2.4, we know the complexity of each iteration should be $O(n^2)$. As a result, when we plot time vs N, the slope should be same as n^2. To verify this, I run several experiment with different N. Table 4 show detail result:

	N	9	25	121	441	1681
Jacobi	iteration	62	310	2404	10892	47614
	runtime(s)	0.000158	0.00118	0.146127	8.12884	532.855
	iter_avg(s)	2.54839E-06	3.80645E-06	6.07849E-05	0.000746313	0.01119114
Gauss-Seidel	iteration	32	155	1202	5441	23779
	runtime(s)	0.000133	0.000586	0.075157	4.06437	251.868
	iter_avg(s)	4.15625E-06	3.78065E-06	6.25266E-05	0.00074699	0.01059203
Symmetric Gauss-Seidel	iteration	23	90	628	2773	12007
	runtime(s)	0.000163	0.000691	0.079197	4.103	259.473
	iter_avg(s)	7.08696E-06	7.67778E-06	0.00012611	0.001479625	0.02161014

Table 4: Result of different N

From Table 4, Figure 5 show iter_avg vs N with log-scale:

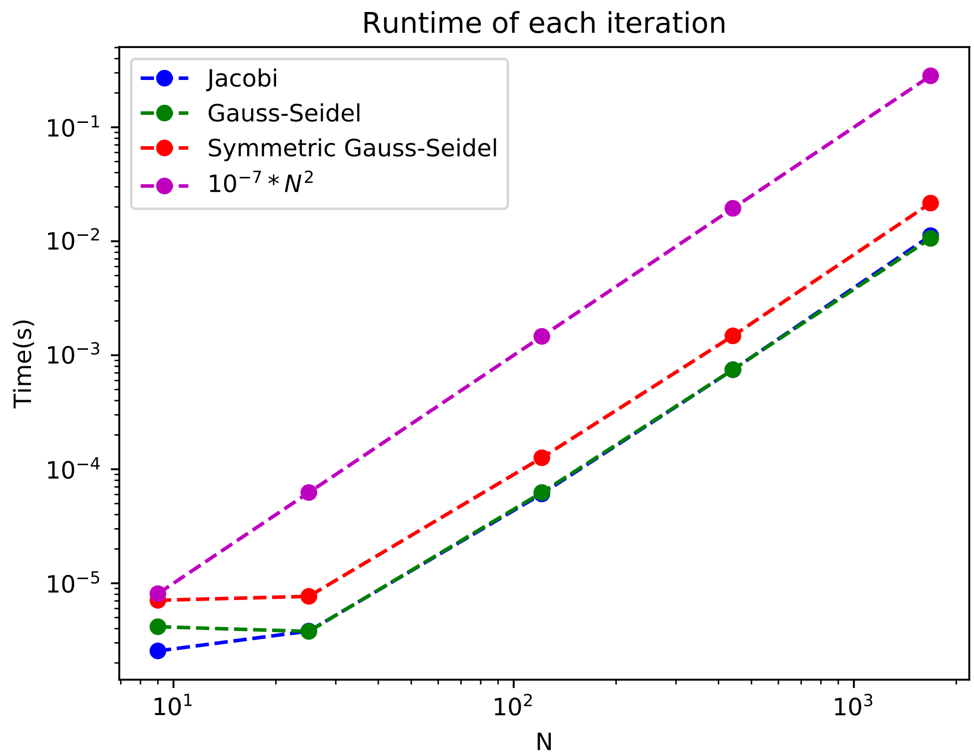

Figure 5: Iteration runtime(each)

For more clear visualization, I plot $10^{-7} * N^2$ vs N instead of N^2 vs N. In Figure 5, we can clearly see that the slope of three iteration methods are same as N^2, which mean their complexity is exactly $O(n^2)$ and satisfy my analysis in Section 2.4.

3.4 Gauss-Seidel vs Symmetric Gauss-Seidel

From Table 4, we can found that the number of iteration of Gauss-Seidel is almost double as Symmetric Gauss-Seidel as shown in Figure 6:

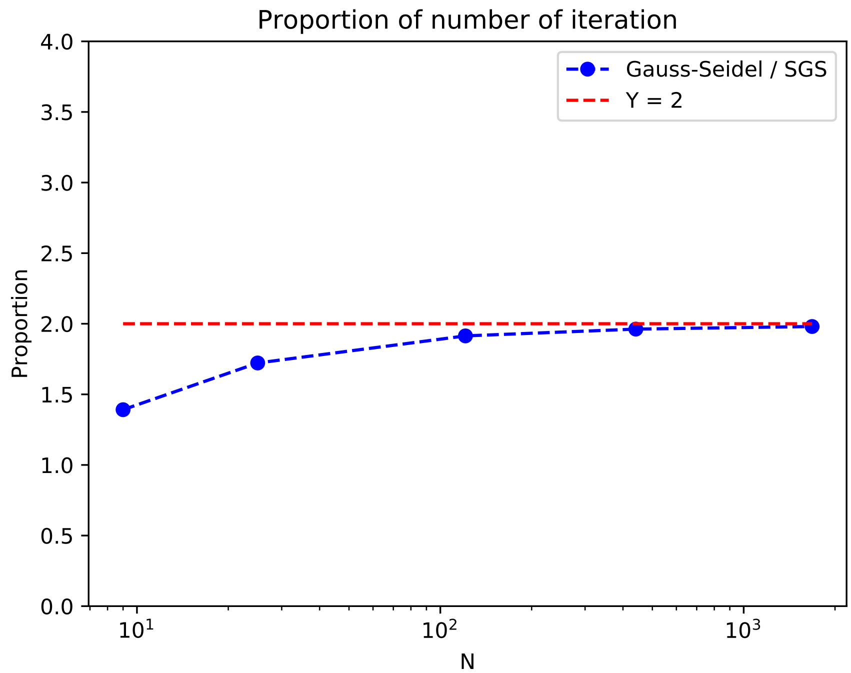

Figure 6: Iteration number proportion

Figure 6 exactly shows that Symmetric Gauss-Seidel only use $\frac{1}{2}$ iterations as Gauss-Seidel. However, when we plot proportion of iter_avg vs N, we will find that the runtime of each iteration of Symmetric Gauss-Seidel is double as that of Gauss-Seide, as shown in Figure 7:

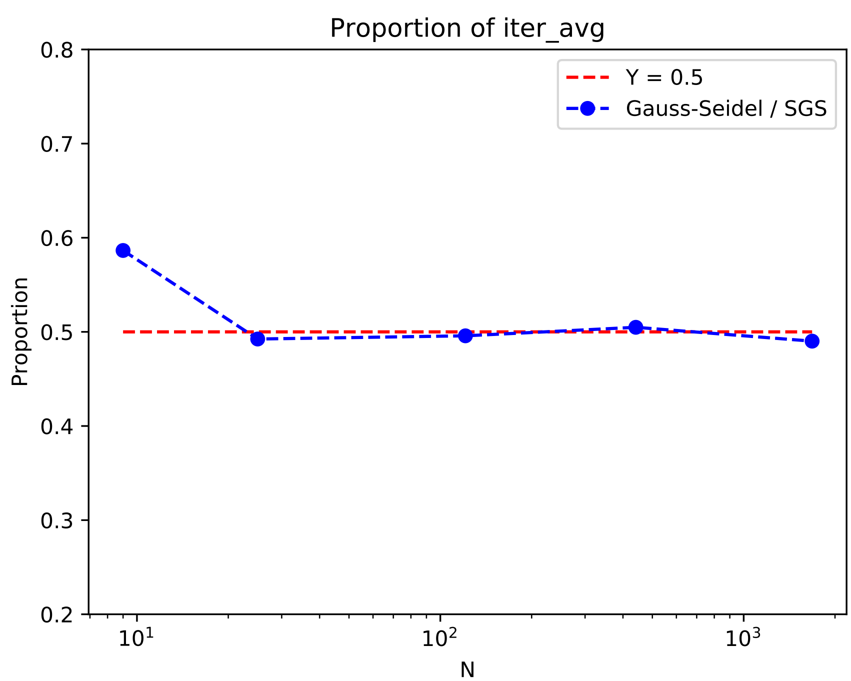

Figure 7: Iteration runtime proportion(each)

As a result, the efficiency of Symmetric Gauss-Seidel is almost same as Gauss-Seidel. When we plot the proportion of total runtime vs N, we will see it converges to 1, as shown in Figure 8:

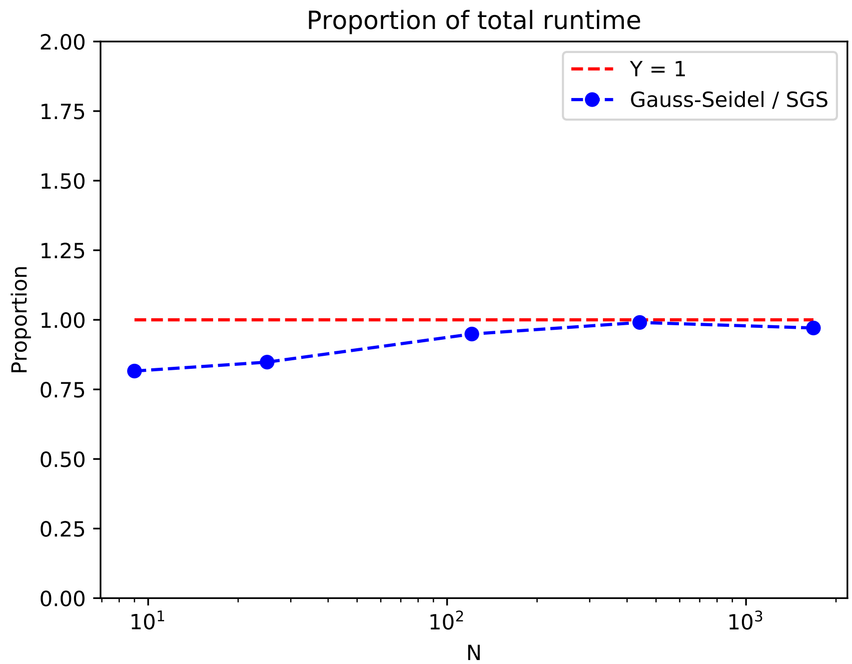

Figure 8: Iteration runtime proportion(total)

From Figure 8, we can find that Symmetric Gauss-Seidel is actually a little slower than Gauss-Seidel. As a result, Gauss-Seidel Method is better than Symmetric Gauss-Seidel.

3.5 Comparison

From Table 4, iter_avg of Jacobi and Gauss-Seidel are almost same, but iteration number of jacobi is larger than Gauss-Seidel. And from Section 3.4, we know Gauss-Seidel is better than Symmetric Gauss-Seidel. As a result, Gauss-Seidel is the best algorithm in this homework.

4 Conclusion

As we discuss in Section 3.2 and Section 3.5, Infinite Error and Gauss-Seidel should be the best method in this project.