CSE6643 Numerical Linear Algebra HW6

December 11, 2016

1

(Saad, p. 147, #6) Consider the linear system $Ax = b$, where A is an SPD matrix. We define a projection method that uses a two-dimensional space at each step. At a given step, take $\mathcal{L} = \mathcal{K} = \text{span}\{r, Ar\}$, where $r = b - Ax$ is the current residual.

- (a) For a basis of $\mathcal{K}$ use the vector r and the vector p obtained by orthogonalizing Ar against r with respect to the A inner product. Give the formula for computing p(no need to normalize the resulting vector).
- (b) Write the algorithm for performing the projection method described above.
- (c) Will the algorithm converge for any initial guess x_0? Justify the answer. [Hint: Exploit the convergence results for one-dimensional projection techniques.]

(a)Solution:
Since $\mathcal{K} = span\{r, Ar\}$, and p is a basis of $\mathcal{K}$, thus p can be represented by basis Ar and r. Besides, since p is orthogonal to Ar which is $p^T Ar = 0$, therefore basis of $\mathcal{K}$, p can be defined as $p = Ar - cr$. (c is constant) In order to compute p, we can obtain by:

$$p^T Ar = (Ar - cr)^T Ar = (Ar)^T Ar - (cr)^T Ar = r^T A^2 r - cr^T Ar$$
$$\Rightarrow c = \frac{r^T A^2 r}{r^T Ar}$$
$$\Rightarrow p = Ar - \frac{r^T A^2 r}{r^T Ar} r$$

(b)Solution:

According to the problem description, r and p form a basis of $\mathcal{K}$ and besides, since $\mathcal{L} = \mathcal{K}$, then we can let $V = W = [r, p]$. According to (Saad, p.135) If the approximate solution is written as

$$\tilde{x} = x_0 + Vy$$

then the orthogonality condition leads immediately to the following system of equations for the vector y:

$$W^T AVy = W^T r_0$$

Since matrix A is an SPD matrix, so matrix $W^T AV$ is non-singular, the following expression for the approximate solution $\tilde{x}$ results,

$$\tilde{x} = x_0 + V(W^T AV)^{-1} W^T r_0 \tag{1}$$

Since:

$$W^TAV = \begin{bmatrix} r^T \\ p^T \end{bmatrix} A \begin{bmatrix} r & p \end{bmatrix} = \begin{bmatrix} r^TAr & 0 \\ 0 & p^TAp \end{bmatrix} \tag{2}$$

Substitute equation (2) into equation (1):

$$\begin{aligned} \tilde{x} &= x_0 + [r,p] \begin{bmatrix} r^TAr & 0 \\ 0 & p^TAp \end{bmatrix}^{-1} \begin{bmatrix} r^T \\ p^T \end{bmatrix} r \\ &= x_0 + \frac{p^Tr}{p^TAp}p + \frac{r^Tr}{r^TAr} \end{aligned}$$

Therefore, we can construct a iterated two-dimensional projection process which is similar to one-dimensional projection processes.(Saad, p.142)
Let:

$$\alpha = \frac{r^Tr}{r^TAr} \quad \beta = \frac{p^Tr}{p^TAp}$$

We need to update α and β in each iterated step, and compute the x_{k+1}.

Algorithm 1: PROBLEM-1

```
input  : $A, b$ and initial guess $x_0$
output: Final converged result $\tilde{x}$
1 Function Two-dimensional Projection $(A, b, x_0)$
2     $maxIt = m$; //Maximum iterations
3     $r_0 = b - Ax_0$;
4     for $k = 1$ to maxIt do
5         $\alpha = \frac{r_k^T r_k}{r_k^T A r_k}$;
6         $\beta = \frac{p_k^T r_k}{p_k^T A p_k}$;
7         $x_{k+1} = x_k + \alpha r_k + \beta p_k$;
8         $r_{k+1} = r_k - \alpha(Ar_k) - \beta(Ap_k)$;
9     end
```

(c)Answer: The algorithm will converge for any initial guess x_0.

Justify the answer:
Suppse that x_* is the exact solution of the original problem, according to the Theorem 5.9 (Saad, p.144), since A is a SPD matrix, then if the A-norms of the error vectors $d_k = x_* - x_k$ satisfy the relation:

$$||d_{k+1}||_A \leq \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}} ||d_k||_A$$

then algorithm will converges for any initial guess x_0. Start by expanding the square of the A-norm of $d_{k+1} = d_k - \alpha_k r_k - \beta_k r_k$ as

$$||d_{k+1}||_A^2 = (d_{k+1}, d_k - \alpha r_k - \beta r_k)_A = (d_{k+1}, d_k)_A - \alpha_k (d_{k+1}, r_k)_A - \beta_k (d_{k+1}, r_k)_A$$

Since A is SPD, it defines an inner product which is usually denoted by $(.,.)_A$ and $(A(d_0 - \delta), \omega) \rightarrow (d_0 - \delta, \omega)_A = 0$. Then,

$$||d_{k+1}||_A^2 = ||x_* - x_{k+1}||_A^2 = (A(x_* - x_{k+1}))^T (x_* - x_{k+1}) = (x_* - x_{k+1})^T A (x_* - x_{k+1}) \tag{3}$$

Since

$$b = Ax_* \quad r_k = b - Ax_k \quad r_{k+1} = b - Ax_{k+1} \tag{4}$$
$$\Rightarrow r_k = A(x_* - x_k) \quad r_{k+1} = A(x_* - x_{k+1}) \tag{5}$$

Substitute into the previous equation (3)

$$\begin{aligned} ||d_{k+1}||_A^2 = ||x_* - x_{k+1}||_A^2 = (x_* - x_{k+1})^T A(x_* - x_{k+1}) &= (x_* - x_k - \alpha_k r_k - \beta_k p_k)^T r_{k+1} \\ &= (x_* - x_k)^T A(x_* - x_k) - \alpha_k r_k^T r_k - \beta_k r_k^T p_k \\ &= ||x_* - x_k||_A^2 - \alpha_k r_k^T r_k - \beta_k r_k^T p_k \end{aligned}$$

Since

$$\alpha_k = \frac{(r_k, r_k)}{(r_k, Ar_k)} \quad \beta_k = \frac{(r_k, p_k)}{(p_k, Ap_k)}$$

Therefore

$$\begin{aligned} &\Rightarrow ||d_k||_A^2 (1 - \frac{(r_k, r_k)}{(r_k, Ar_k)} \times \frac{(r_k, r_k)}{(r_k, A^{-1} r_k)} - \frac{(r_k, p_k)}{(p_k, Ap_k)} \times \frac{(r_k, p_k)}{(r_k, A^{-1} r_k)}) \\ &\leq ||d_k||_A^2 (1 - \frac{(r_k, r_k)}{(r_k, Ar_k)} \times \frac{(r_k, r_k)}{(r_k, A^{-1} r_k)}) \end{aligned}$$

According to **Lemma 5.8 (Kantorvich inequality) Saad, p.143**, then

$$\begin{aligned} &\Rightarrow 1 - \frac{(r_k, r_k)}{(r_k, Ar_k)} \times \frac{(r_k, r_k)}{(r_k, A^{-1} r_k)} \leq 1 - \frac{4\lambda_{max}\lambda_{min}}{(\lambda_{max} + \lambda_{min})^2} = \frac{(\lambda_{max} - \lambda_{min})^2}{(\lambda_{max} + \lambda_{min})^2} \\ &\Rightarrow ||d_{k+1}||_A \leq ||d_k||_A \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}} \end{aligned}$$

Let

$$\mu = \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}} = 1 - \frac{2}{\lambda_{max}/\lambda_{min} + 1}$$

Thus

$$0 \leq \mu < 1$$

Therefore, the algorithm will converge for any initial guess x_0.

2

Recall that a matrix $A = (A_{ij})$ is Toeplitz if each descending diagonal from left to right is constant, i.e.,

$$A_{i,j} = a_{i-j} \quad for \quad all \quad i, j.$$

In this project we are interested in solving the linear system $Ax = b$ where A is a symmetric Toeplitz matrix, using preconditioned conjugate gradient methods.

Consider the following A (with size of A being $100 \times 100, 500 \times 500, and 1000 \times 1000$):

- (1) $A_{ii} = \alpha, A_{i+1,i} = A_{i,i+1} = \beta, A_{i,j} = A_{j,i} = 0$, if $|i-j| > 1$ with $-\frac{\alpha}{2\beta} = 1+\delta, \delta \geq 0$. (Vary $\delta = 0.1, 1$, and 2.)
- (2) Let $A_{i,j} = \frac{1}{\sqrt{|i-j|+1}}, 1 \leq i,j \leq n$.
- (3) Let $A_{i,j} = \frac{1}{(|i-j|+1)^2}, 1 \leq i,j \leq n$.

Choose a non-zero b and tolerance level around single machine pre- cision. Discuss convergent rate, number of iterations, when different preconditioning vis un-preconditioned CG methods applied. You are encouraged to research and experiment with different preconditioning matrices. You do not have to run your algorithms on all three matrices above, and depending on the level of investigation, you may want to do just one or two from above.

Solution:

- I chose the first matrix above to run my algorithm. In order to use preconditioned conjugate gradient methods, A has to be positive definite matrix and since $-\frac{\alpha}{2\beta} = 1+\delta, \delta \geq 0$. Then $\beta < 0$.
- For parameter β and vector b, I set them randomly and tolerance level is 10^{-9}.
- To implement and compare the difference between PCG and CG, I have tried several different preconditioner. E.g. Jacobi preconditioner, SSOR preconditioner and Gauss-Seidel preconditioner.

The following figures are the comparisons:

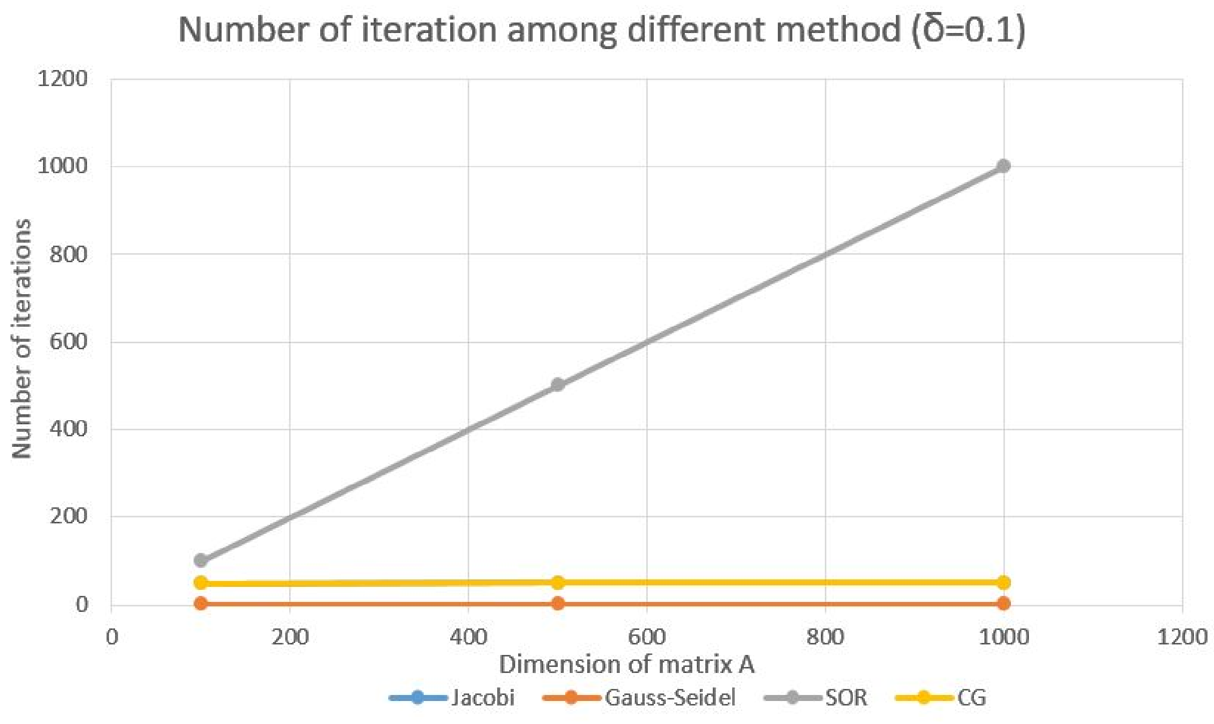

Figure 1: Number of iteration among different methods ($\delta = 0.1$)

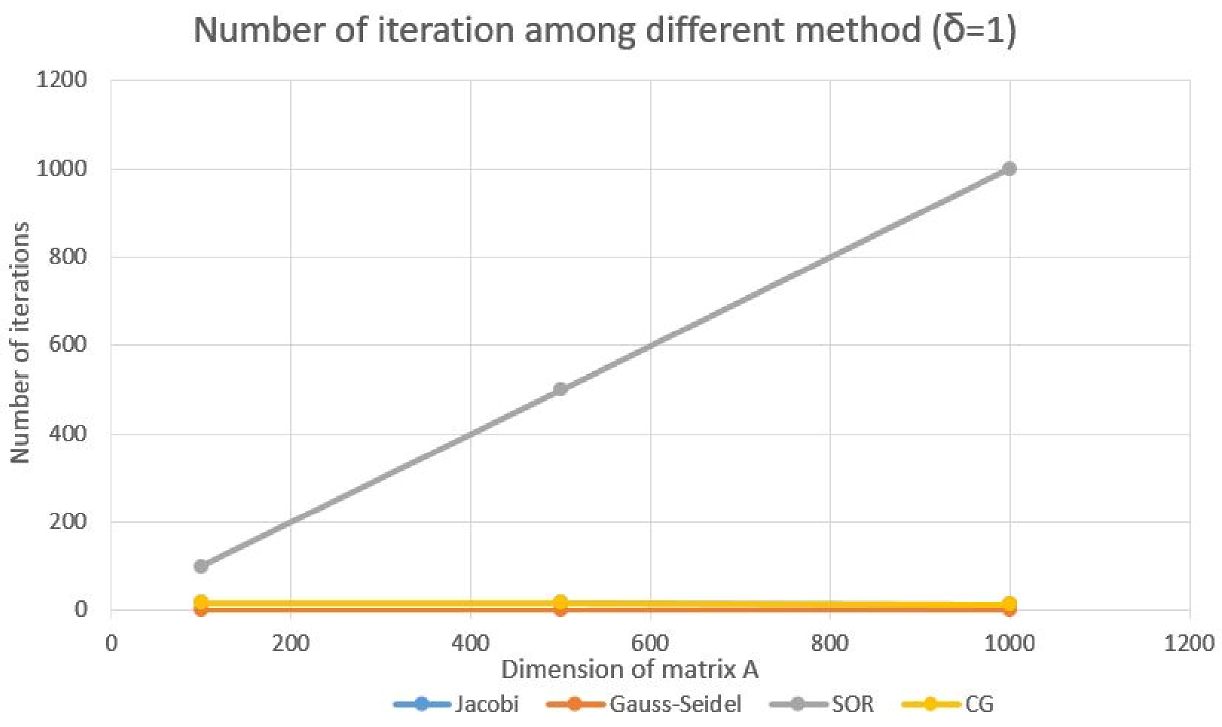

Figure 2: Number of iteration among different methods ($\delta = 1$)

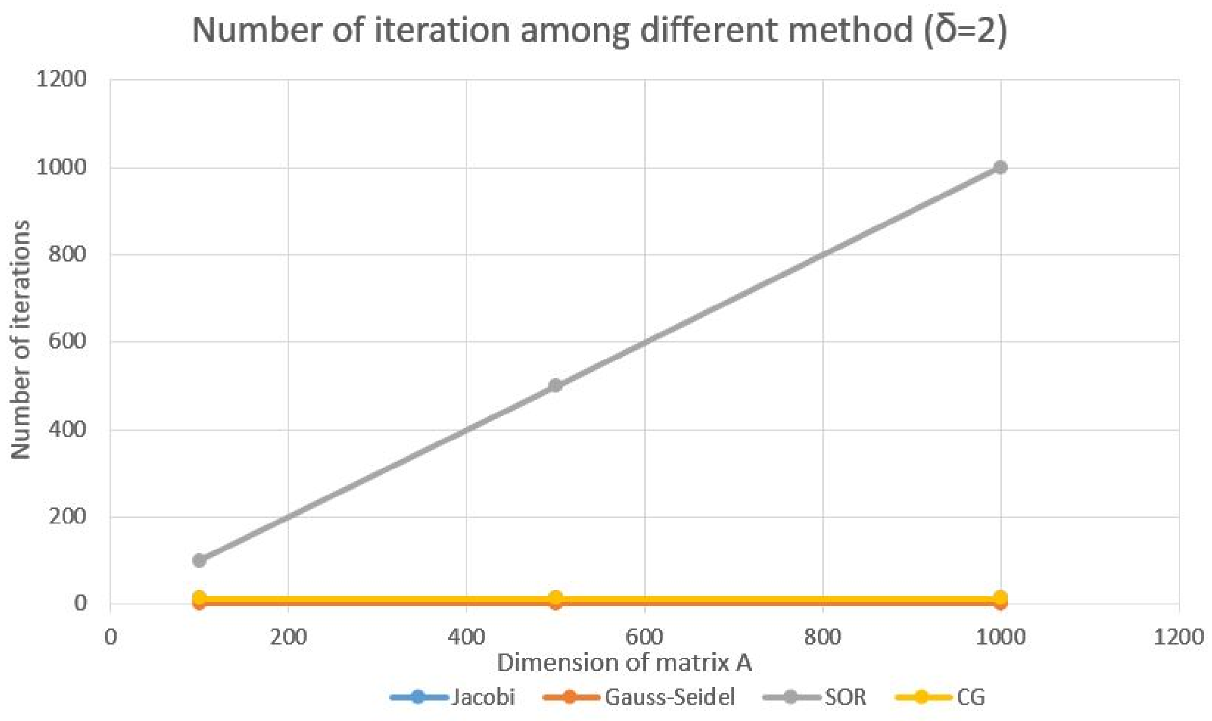

Figure 3: Number of iteration among different methods ($\delta = 2$)

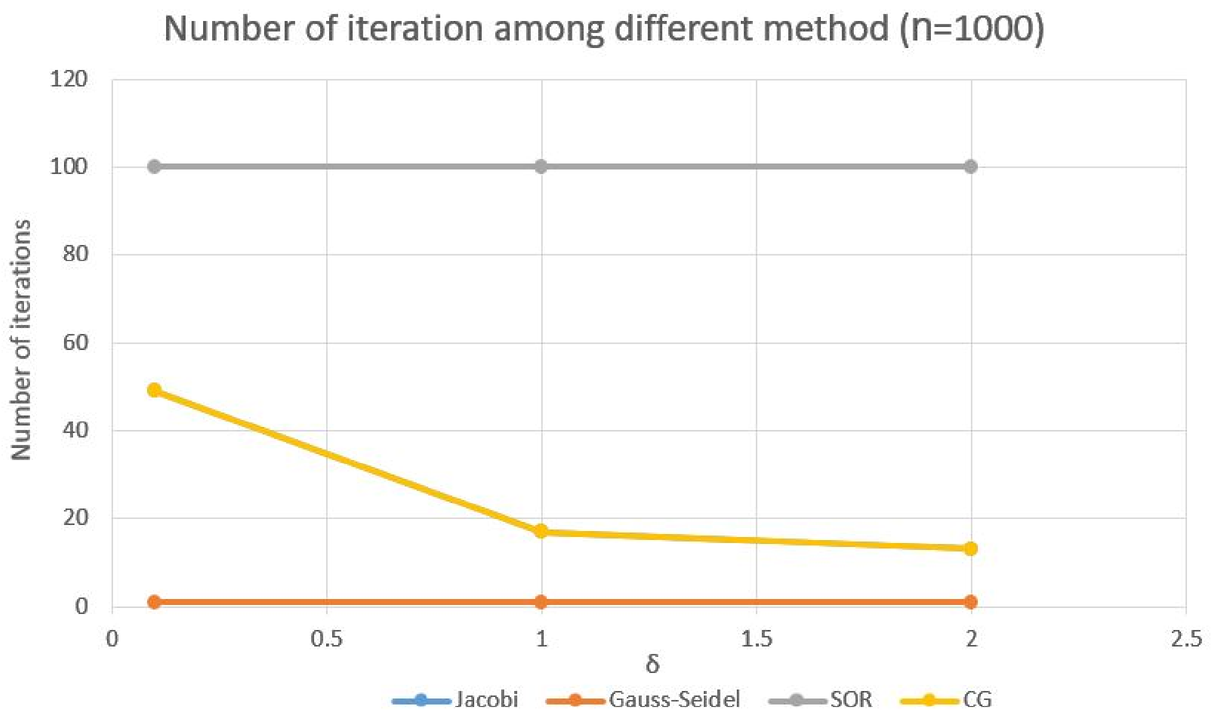

Figure 4: Number of iteration among different methods ($n = 100$)

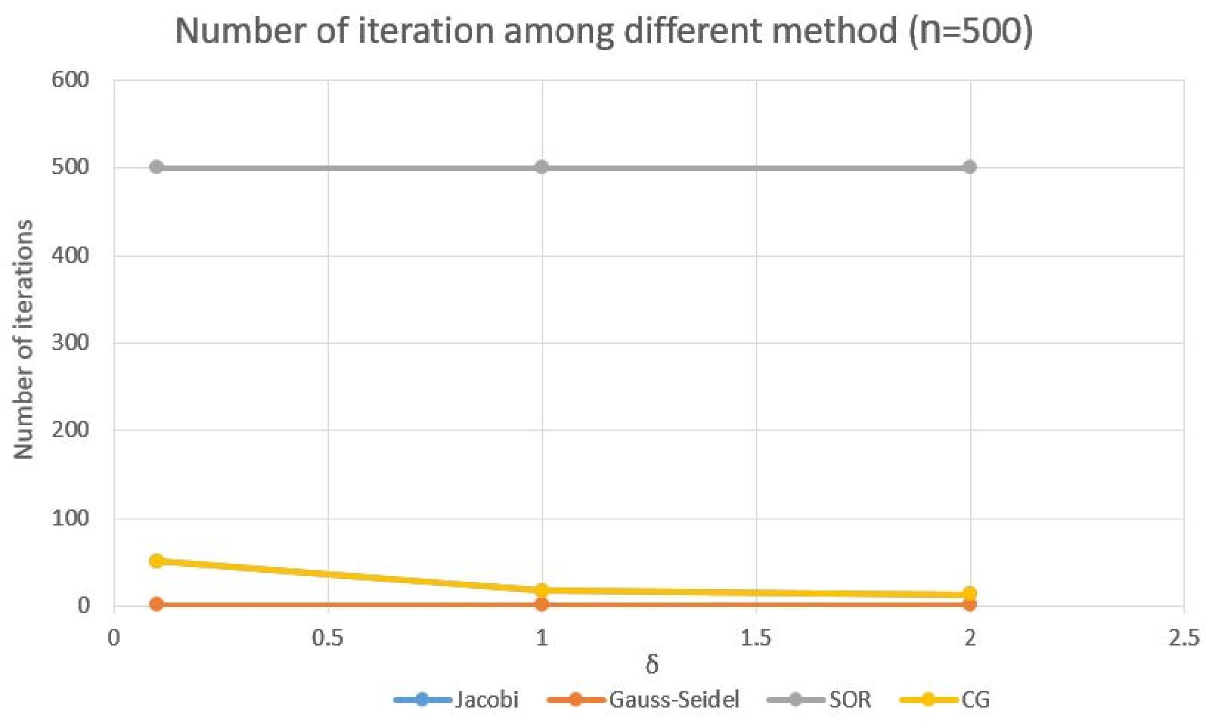

Figure 5: Number of iteration among different methods ($n = 500$)

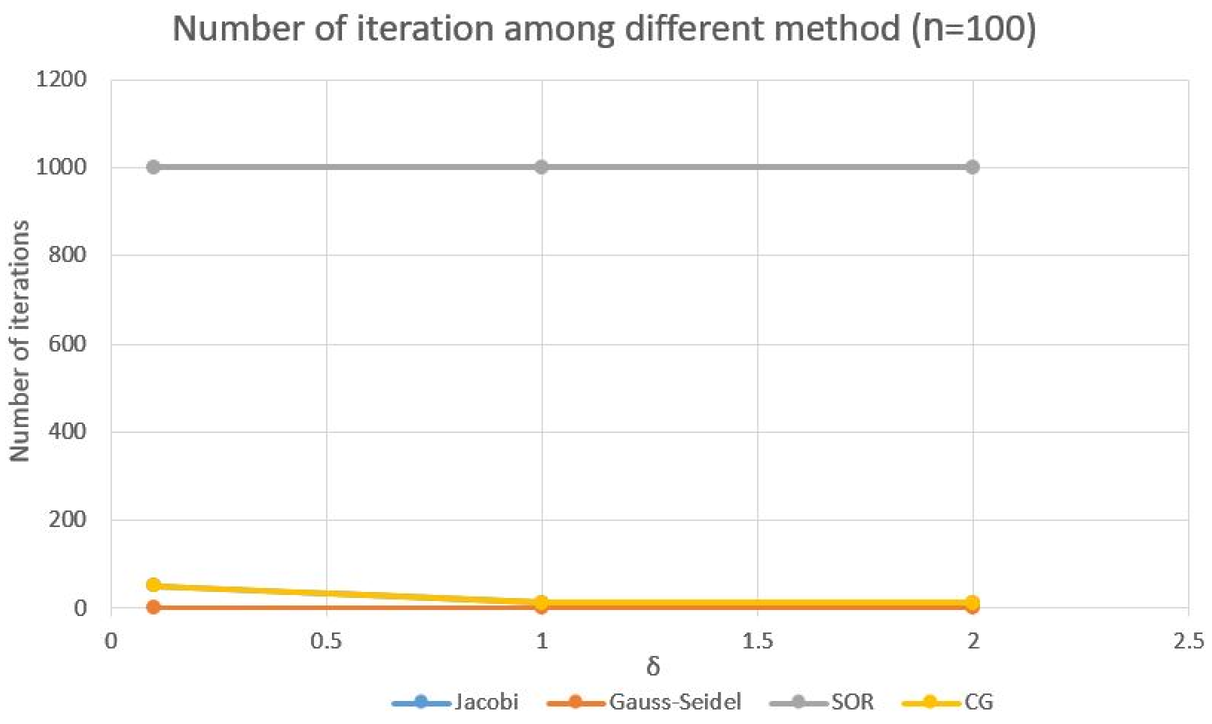

Figure 6: Number of iteration among different methods ($n = 1000$)

Conclusion:

- According to the modification of M, we know that there are two extreme cases $M = I$ and $M = A$. If $M = I$, then it is the same as conjugate gradient method. If $M = A$, it is actually just as complicated as the original one without any improvement.

- By investigate some researches, a proper matrix M should include the following properties:
  (1) M should be symmetric and positive definite.
  (2) M should be such that $M\bar{r_n} = r_n$ can be solved efficiently.
  (3) M should be close to A^-1 in the sense that $||I - M^{-1}A|| < 1$.

- By following some researches and experiments, I chose:($A = L + D + L^T$)
  (1) $M = D$: Jacobi preconditioning;
  (2) $M = L + D$: Gauss-Seidel preconditioning;
  (3) $M = \frac{1}{\omega}(D + \omega L)$ SOR preconditioning;

- **Number of iterations:** According to figures, SOR preconditioner is not a good choice, since by using it, the algorithm won't converge. Besides, by using Jacobi preconditioner, the PCG algorithm performed as good as the CG algorithm. The Gauss-Seidel preconditioner is the best among those three, for $n = 100, 500, 1000$, it just needs one iteration to converge. Therefore, Gaussian-Seidel needs the less number of iterations than other methods.

- According to the figures, with greater δ, number of iterations is decreasing.

- It is obvious that if we decrease the tolerance (for example: 10^{-20}) then the number of iterations will increase too.

- After carefully observe, I find that the precision of SOR preconditioner is limited. If we implemented PCG with SOR preconditioner, the precision level will be 10^{-4}, which will never meet the tolerance I set. And that's the reason why it won't stop until the maximum iterations.

- **Convergent rate:** To compute rate of convergence, we need to compute the following:(Suppse that the sequence x_k converges L)

$$\lim_{k\to\infty} \frac{|x_{k+1} - L|}{|x_k - L|} = \mu$$

  For Jacobi precondition, $\lim_{k\to\infty} \frac{|x_{k+1}-L|}{|x_k-L|} = \mu, 0 < \mu < 1$, then it converges linearly to L.

  For SOR precondition, $\mu = \mu_k$ varies from step to step with $\mu_k \to 1$ for $k \to \infty$, then it converges sublinearly.

  For Gauss-Seidel precondition, $\mu = \mu_k$ varies from step to step with $\mu_k \to 1$ for $k \to \infty$, then it converges superlinearly.

  For CG, μ is the same as Jacobi precondition, thus, it converges linearly to L.

Preconditioning Conjugate Gradient MATLAB Code:

```
function [x,it] = PCG(A,b,x,M)
    r=b-A*x;
    z=M\r;
    p=z;
    tol=1e-9;
    it=1;
    maxIt=1000;
    while it<maxIt
        Ap=A*p;
        alpha=(r'*z)/(p'*Ap);
        x=x+alpha*p;
        rNew=r-alpha*Ap;
        if sqrt(rNew'*rNew)<tol
            break;
        end
        zNew=M\rNew;
        beta=(zNew'*rNew)/(z'*r);
        pNew=zNew+beta*p;
        p=pNew;
        r=rNew;
        z=zNew;
        it=it+1;
    end
end
```

Conjugate Gradient MATLAB Code:

```
function [xNew,it] = CG(A,b,x)
    r=b-A*x;
    p=r;
    tol=1e-9;
    it=1;
    maxIt=1000;
    while it<maxIt
        Ap=A*p;
        alpha=(r'*r)/(p'*Ap);
        xNew=x+alpha*p;
        rNew=r-alpha*Ap;
        if sqrt(rNew'*rNew)<tol
            break;
        end
        beta=(rNew'*rNew)/(r'*r);
        pNew=rNew+beta*p;
        p=pNew;
        r=rNew;
        x=xNew;
        it=it+1;
    end
end
```