暨南大学本科实验报告专用纸

课程名称 数值计算实验 成绩评定

实验项目名称 Computing problem 指导教师 Liangda Fang

实验项目编号 01 实验项目类型 综合性 实验地点 N116

学生姓名 甄洛生、吴凯一 学号 2018054625、2018050821

学院 信息科学技术学院 系 计算机系 专业 计算机科学技术专业

实验时间 2020 年 9 月 30 日下午 10:30~ 12:10

I.Problems

Let A be the 1000 × 1000 matrix with entries A(i, i) = i, A(i, i + 1) = A(i+1,i) = 1/2,A(i, i+2) = A(i+2, i) = 1/2 for all i that fit with in the matrix.

• Solve the system with Ax = [1, 1, · · · , 1]⊤ by the following methods in 15 steps:

1. The Jacobi Method;
2. The Gauss-Seidel Method;
3. SOR with ω = 1.1;
4. The Conjugate Gradient Method;
5. The Conjugate Gradient Method with Jacobi preconditioner.

• Report the errors of every step for each method.

II.Algorithm summary

● The Jacobi method

what is the Jacobi method? First of all, it is similar to the fixed point iterative method mentioned in the first chapter, we need to rewrite the original linear equations and iteratively solve by the initial solution system x0. The rewriting steps are as follows:

$$Ax = b$$
$$(D + L + U)x = b$$
$$Dx = b - (L + U)x$$
$$x = D^{-1}(b - (L + U)x) \qquad (1-1)$$

It can be observed that we decompose the original matrix A into three independent matrices:

1. principal diagonal matrix D
2. lower triangular (elements below principal diagonal) matrix L
3. upper triangular (elements above principal diagonal) matrix U

And now, we can solve it iteratively as follows:

$$x_0 = initial\ solution$$
$$x_{k+1} = D^{-1}(b - (L + U)x_k), k = 0,1,2, \ldots \qquad (1-2)$$

However, the Jacobi method has an important premise that if the matrix A is not a strictly diagonally dominant matrix, then the iterative solution may diverge. And now, what is the strictly diagonally dominant matrix? Its definition as follows:

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, i = 1,2,\dots,n \qquad (1-3)$$

That is why it is called dominant matrix. Cause diagonal element of each row is bigger than the sum of element of this row(exclude diagonal element)!

In our experiment, A is not a strictly diagonally dominant matrix as its first row doesn't match the above formula. But I solve it by 15 iterations and it can be converged. So strictly diagonally dominant matrix is only a sufficient condition.

● The Gauss-Seidel method

The Gauss Seidel method is similar to the Jacobi method, except that Jacobi uses the value of the old variable group to calculate the value of the new variable group, while the Gauss Seidel method uses the new variable value to get the value of the new variable. The method describes as follows:

$$x_{k+1} = D^{-1}(b - Ux_k - Lx_{k+1}) \qquad (1-4)$$

The algorithm is as follows:

$$x_k = \left(x_k^{(1)}, x_k^{(2)}, \dots, x_k^{(n)}\right), k = 0,1,2,\dots$$
$$b = \left(b^{(1)}, b^{(2)}, \dots, b^{(n)}\right)$$
$$x_k^{(i)} = {D^{-1}}_{(i,i)} * \left(b^{(i)} - (L+U)_{(i,:)} * x_k\right), i = 1,2,\dots,n$$
$$when\ all\ i\ computed, x_{k+1} = x_k$$

In the case of using the same steps, the numerical solution of the Gauss Seidel method will be more accurate, and the convergence rate is usually faster than that of the Jacobi method (if convergent), and as long as the matrix An is a strictly diagonal matrix, the Gauss Seidel method converges.

● SOR with $\omega = 1.1$

SOR is the abbreviation of successive overrelaxation iterative method, its birth is based on the Gauss Seidel method(G-S), the relaxation factor ω is introduced in the Gauss Seidel method, the new solution x_{k+1} obtained by G-S is multiplied by this ω, and the old solution x_k is multiplied by $1 - \omega$, and the new solution of SOR is obtained by adding the two. Reasonable selection of ω can greatly accelerate the speed of iterative convergence. Here gives its method procedure:

$$Ax = b$$
$$\omega Ax = \omega b$$
$$(\omega D + \omega L + \omega U)x = \omega b$$

$$(D+\omega L)x=\omega b-\omega Ux+(1-\omega)Dx \qquad (1-5)$$

And that is to say…

$$(D+\omega L)x_{k+1}=\omega b-\omega Ux_k+(1-\omega)Dx_k$$
$$x_{k+1}=(1-\omega)x_k+\omega D^{-1}(b-Ux_k-Lx_{k+1}) \qquad (1-6)$$

For programming, it should solve each variable in order like G-S:

$$x_k=\left(x_k^{(1)},x_k^{(2)},\dots,x_k^{(n)}\right),k=0,1,2,\dots$$

$$x_k^{(i)}=(1-\omega)x_k^{(i)}+\omega D_{(i,i)}^{-1}(b^{(i)}-(L+U)_{(i,:)}*x_k),i=1,2,\dots,n$$

$$when\ all\ i\ computed, x_{k+1}=x_k$$

Noted that if SOR converged, ω must be in $(0,2)$. Now we are concerned about what the best value of ω is. Now introducing Jacobi iterative matrix C_{jac}:

$$C_{jac}=I-D^{-1}A \qquad (1-7)$$

And find its biggest eigenvalue of absolute value μ:

$$u=\max(|eig(C_{jac})|)$$

So optimal ω is…

$$\omega_{opt}=1+\left(\frac{\mu}{1+\sqrt{1-\mu^2}}\right)^2 \qquad (1-8)$$

In this experiment, for matrix A, $\omega_{opt}=1.1317$. So, we use $\omega=1.1\ to\ solve\ it.$

● The Conjugate Gradient Method

The conjugate gradient method is different from the previous three methods in that it transforms the problem of $Ax=b$ into the problem to find the minimum value of the function:

$$\phi(x)=\frac{1}{2}x^TAx-b^Tx \qquad (1-9)$$

If we take the derivative of $\phi(x)$, get:

$$\phi'(x)=\frac{1}{2}A^Tx+\frac{1}{2}Ax-b \qquad (1-10)$$

$\phi(x)$ is a strictly convex quadratic programming, that is to say, if we find the min value of $\phi(x_0)$, $\phi'(x_0)$ must be zero. And that is we exactly want ($Ax_0-b=0$)! But how to find it? Now we introduce the conjugate gradient method.

Matrix A, it must be a symmetric positive definite matrix. Its definition as follows:

$$A'=A$$
$$\forall x\neq 0, x^TAx>0$$

And now, $\phi'(x) = Ax - b$. Cause matrix A is a symmetric matrix.

However, we want to find the min value for $\phi(x)$. We might as well give an initial point x_0, and now we need to determine the direction of the descent p and the step size α. And the new point will be:

$$x_1 = x_0 + \alpha d$$

And little more formal:

$$x_{k+1} = x_k + \alpha_k d_k \qquad (1-11)$$

So, now, how to determine α and d? Cause this method is called by the conjugate gradient method, of course we have to use the concept of conjugation:

$$P = \{p_1, p_2, \dots, p_n\}$$
$$if\ i \in \{1,2,\dots,n\}, p_i \neq 0\ and\ \forall i \neq j, p_i^T A p_j = 0$$
$$then\ P\ is\ a\ the\ conjugate\ basis\ vector\ set$$

Any two vectors in the conjugate basis vector set are linearly independent in matrix A, assume x^* is the solution of $Ax = b$. So here is:

$$x^* = \sum_{i=1}^{n} \gamma_i p_i, \gamma_i\ is\ constant \qquad (1-12)$$

Now times A both size:

$$Ax^* = \sum_{i=1}^{n} \gamma_i A p_i \qquad (1-13)$$

And then times any conjugate vector p_k:

$$p_k^T A x^* = \sum_{i=1}^{n} \gamma_i p_k^T A p_i$$
$$\because \forall i \neq j, p_i^T A p_j = 0$$
$$\therefore p_k^T b = \gamma_k p_k^T A p_k$$
$$\therefore \gamma_k = \frac{p_k^T b}{p_k^T A p_k} \qquad (1-14)$$

And now problem becomes:

$$x^* = \sum_{i=1}^{n} \frac{p_i^T b}{p_i^T A p_i} p_i$$

Therefore, how to determine P becoming our primary problem! Inspired by the gradient descent method, we might as well consider that the first vector of the vector set P is the gradient direction of the initial point x_0:

$$p_1 = -\phi'(x_0) = b - Ax_0$$

p_1 also called by residual which is difference between Ax and b. Now we should give other p_i iteratively, and make them conjugated each other:

$$p_k = r_k - \sum_{1}^{k-1} \lambda_i p_i$$

λ is a unknown variable, we can solve by timing $Ap_j, 1 \leq j \leq k-1$:

$$0 = p_j^T A p_k = p_j^T A r_k - \sum_{i=1}^{k-1} \lambda_i p_j^T A p_i = p_j^T A r_k - \lambda_j p_j^T A p_j$$

$$\therefore \lambda_i = \frac{p_i^T A r_k}{p_i^T A p_i} \qquad (1-15)$$

And now:

$$p_k = r_k - \sum_{1}^{k-1} \frac{p_i^T A r_k}{p_i^T A p_i} p_i$$

However, $r_i = b - Ax_i$. $x_{k+1} = x_k + \alpha_k p_k$. Now we know how to generate $P = \{p_1, p_2, \dots, p_n\}$. What about step size α? For minimal $f(x_{k+1}) = Ax_{k+1} - b$:

$$\because \frac{\partial f(x_{k+1})}{\partial \alpha_k} = 0 \; and \; x_{k+1} = x_k + \alpha_k p_k$$

$$\therefore \alpha_k = \frac{b - Ax_k}{Ap_k} \qquad (1-16)$$

Well done! Now we just calculate by this order:

$$x_1 \to r_1 \to p_1 \to x_2 \to \cdots \to x_n$$

Conjugate gradient method promises that $Ax = b$ can be solved with no more than n steps. n is A dimension. Whatmore, the conjugate gradient method is very suitable for solving sparse matrix.

● The Conjugate Gradient Method with Jacobi preconditioner

The use of preconditioners in other equations can speed up the convergence. The so-called pre-conditioner is just a matrix. The Jacobi preconditioner is the diagonal matrix D of matrix A. We only need to multiply the inverse of the preconditioned submatrix between the left and right sides of $Ax = b$:

$$M = D$$

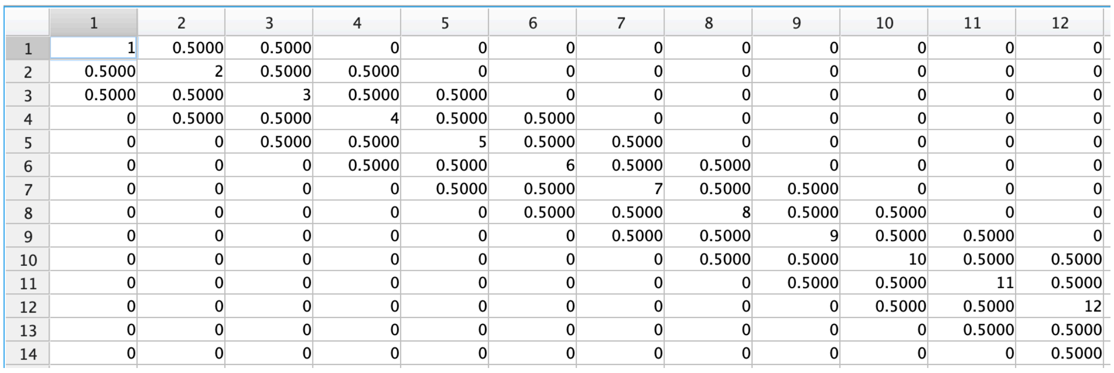
$$M^{-1}Ax = M^{-1}b$$

$$\hat{A}\hat{x} = \hat{b} \qquad (1-17)$$

Now the number of conditions of $\hat{A}$ is smaller than that of original A. And just solve it following above the conjugate gradient method.

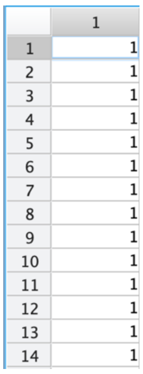
III.Result analysis

Matrix A(Only few rows for simplity):



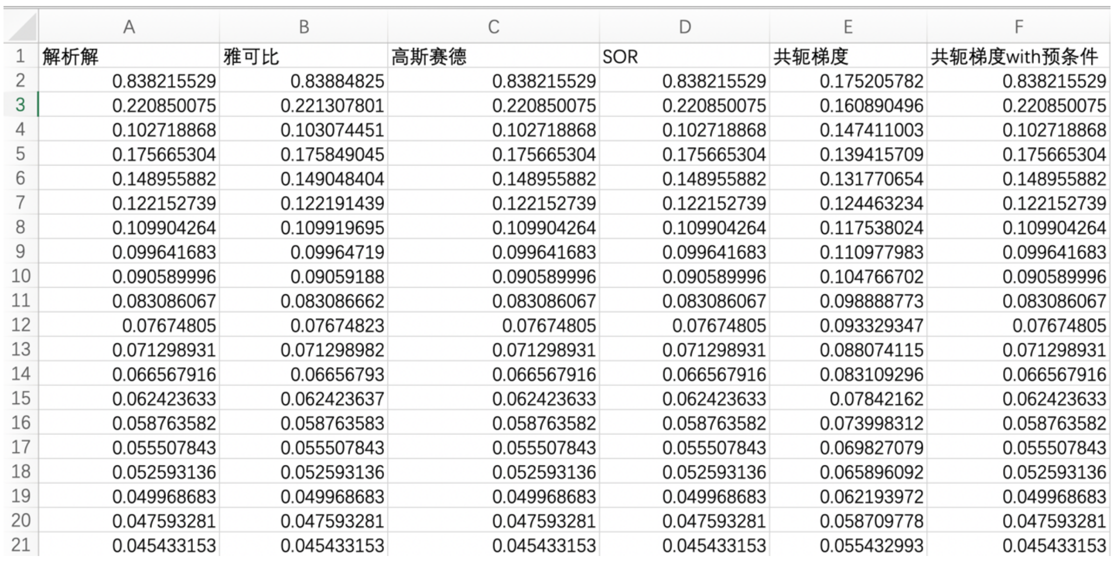
	1	2	3	4	5	6	7	8	9	10	11	12
1	1	0.5000	0.5000	0	0	0	0	0	0	0	0	0
2	0.5000	2	0.5000	0.5000	0	0	0	0	0	0	0	0
3	0.5000	0.5000	3	0.5000	0.5000	0	0	0	0	0	0	0
4	0	0.5000	0.5000	4	0.5000	0.5000	0	0	0	0	0	0
5	0	0	0.5000	0.5000	5	0.5000	0.5000	0	0	0	0	0
6	0	0	0	0.5000	0.5000	6	0.5000	0.5000	0	0	0	0
7	0	0	0	0	0.5000	0.5000	7	0.5000	0.5000	0	0	0
8	0	0	0	0	0	0.5000	0.5000	8	0.5000	0.5000	0	0
9	0	0	0	0	0	0	0.5000	0.5000	9	0.5000	0.5000	0
10	0	0	0	0	0	0	0	0.5000	0.5000	10	0.5000	0.5000
11	0	0	0	0	0	0	0	0	0.5000	0.5000	11	0.5000
12	0	0	0	0	0	0	0	0	0	0.5000	0.5000	12
13	0	0	0	0	0	0	0	0	0	0	0.5000	0.5000
14	0	0	0	0	0	0	0	0	0	0	0	0.5000

Vector b(Only few rows for simplity):



	1
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
11	1
12	1
13	1
14	1

Solution performance:

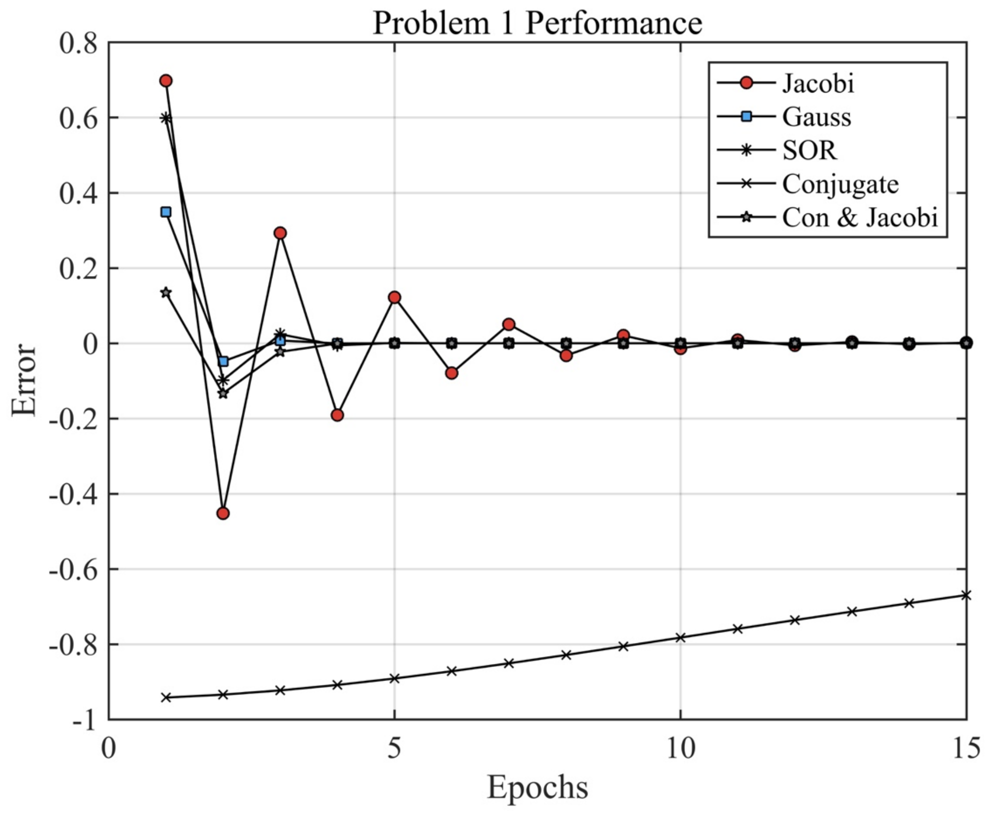


	A	B	C	D	E	F
1	解析解	雅可比	高斯赛德	SOR	共轭梯度	共轭梯度with预条件
2	0.838215529	0.83884825	0.838215529	0.838215529	0.175205782	0.838215529
3	0.220850075	0.221307801	0.220850075	0.220850075	0.160890496	0.220850075
4	0.102718868	0.103074451	0.102718868	0.102718868	0.147411003	0.102718868
5	0.175665304	0.175849045	0.175665304	0.175665304	0.139415709	0.175665304
6	0.148955882	0.149048404	0.148955882	0.148955882	0.131770654	0.148955882
7	0.122152739	0.122191439	0.122152739	0.122152739	0.124463234	0.122152739
8	0.109904264	0.109919695	0.109904264	0.109904264	0.117538024	0.109904264
9	0.099641683	0.09964719	0.099641683	0.099641683	0.110977983	0.099641683
10	0.090589996	0.09059188	0.090589996	0.090589996	0.104766702	0.090589996
11	0.083086067	0.083086662	0.083086067	0.083086067	0.098888773	0.083086067
12	0.07674805	0.07674823	0.07674805	0.07674805	0.093329347	0.07674805
13	0.071298931	0.071298982	0.071298931	0.071298931	0.088074115	0.071298931
14	0.066567916	0.06656793	0.066567916	0.066567916	0.083109296	0.066567916
15	0.062423633	0.062423637	0.062423633	0.062423633	0.07842162	0.062423633
16	0.058763582	0.058763583	0.058763582	0.058763582	0.073998312	0.058763582
17	0.055507843	0.055507843	0.055507843	0.055507843	0.069827079	0.055507843
18	0.052593136	0.052593136	0.052593136	0.052593136	0.065896092	0.052593136
19	0.049968683	0.049968683	0.049968683	0.049968683	0.062193972	0.049968683
20	0.047593281	0.047593281	0.047593281	0.047593281	0.058709778	0.047593281
21	0.045433153	0.045433153	0.045433153	0.045433153	0.055432993	0.045433153

Convergence rate:

Error function definition as follows:

$$E(x, y) = x^2 - y^2$$



Analysis:

We can observe that the fastest convergence speed is the Gaussian Seidel method, followed by the SOR method, then the conjugate gradient method with Jacobi preconditioner, then the Jacobi method, and finally the simple conjugate gradient method. At the same time, we find that the conjugate gradient has a large error within 15 steps.

IV.Experimental summary

In this experiment, we use five different iterative methods to solve the system of linear equations $Ax = b$. And the convergence rates of the five methods are compared based on the same number of iterations. Let me know that sometimes we don't have to get an exact solution, we just want to get an approximate solution in a short time. It reflects the trade between time and quality.

Also noted that A is not a strictly diagonally dominant matrix but it still works on Jacobi, G-S and SOR.

Anyway, each of the five methods has its own advantages, and satisfactory results can only be obtained by using them according to the situation.

V.Appendix : Source code

```
clear;clc

%% 定义常量
A = diag(repmat(1/2,999,1),1) + diag(repmat(1/2,998,1),2);
A = A + diag(repmat(1/2,999,1),-1) + diag(repmat(1/2,998,1),-2);
A = diag(1:1000) + A;
```

```matlab
b = repmat(1,1000,1);

%% 判断是否为严格对角占优矩阵
if abs(diag(A)) > sum(abs(A-diag(diag(A))),2)
    disp('Aha, here it is.')
else
    disp('Oh shit! Not a strictly diagonally dominant.')
end

LOOP = 15;    % 迭代次数
% 求解析解
real_x = inv(A) * b;

%% Jacobi Method
disp('Jacobi...')
% 分解 A 为 L D U
D = diag(diag(A));  % 对角矩阵
L = tril(A,-1);      % 下三角矩阵
U = triu(A,1);      % 上三角矩阵
% 设定初始解 x 向量
old_x = zeros(length(b),1);
% 记录每一次迭代误差值
Error_Jacobi = zeros(1,LOOP);
for count = 1: LOOP
    Jacobi = inv(D) * (b - (L + U) * old_x);
    % calculate error
    Error_Jacobi(count) = ErrorFunc(Jacobi, real_x);
    % update
    old_x = Jacobi;
end
% 显示解
%disp('Jacobi method:')
%disp(new_x')


%% Gauss-seidel Method
disp('Gauss-seidel...')
% 设定初始解 x 向量
old_x = zeros(length(b),1);
Gauss_seidel = zeros(size(old_x));

% 记录每一次迭代误差值
Error_Gauss_seidel = zeros(1,LOOP);
LU = L + U;
```

```
invD = inv(D);
for count = 1: LOOP
    % 依次求每个变量新的值，每次求新的值时，有些变量已经是最新的值
    for index = 1 : length(old_x)
        old_x(index) = invD(index,index) * (b(index) - LU(index, :) * old_x);
        Gauss_seidel = old_x;
    end
    Error_Gauss_seidel(count) = ErrorFunc(Gauss_seidel, real_x);
end
% 显示解
%disp('Gauss-seidel method:')
%disp(new_x')


%% SOR with omega == 1.1
disp('SOR...')
omega = 1.1;
% 设定初始解 x 向量
old_x = zeros(length(b),1);
invD = inv(D);
LU = L + U;
Error_SOR = zeros(1,LOOP);
for count = 1: LOOP
    for index = 1 : length(old_x)
        old_x(index) = (1 - omega) * old_x(index) + invD(index,index) * omega * (b(index) - (LU(index, :) * old_x));
    end
    SOR = old_x;
    Error_SOR(count) = ErrorFunc(SOR, real_x);
end
% 显示解
%disp('SOR-omega1.1 method:')
%disp(new_x')


%% Conjugate Gradient Method
disp('Conjugate gradient...')
% 设定初始解 x 向量
x = zeros(length(b),1);
old_d = b;
old_r = b;
zero_r = zeros(size(old_r));
Error_Conjugate = zeros(1,LOOP);
```

```matlab
for count = 1: LOOP
    if old_r == zero_r
        break
    else
        alpha = (old_r' * old_r) / (old_d' * A * old_d);
        Conjugate = x + alpha * old_d;  % new x
        new_r = old_r - alpha * A * old_d;
        my_beta = (new_r' * new_r) / (old_r' * old_r);
        new_d = new_r + my_beta * old_d;
        % calculate error
        Error_Conjugate(count) = ErrorFunc(Conjugate, real_x);
        % update d and r
        old_d = new_d;
        old_r = new_r;
        x = Conjugate;
    end
end
% 显示解
%disp('Conjugate Gradient method:')
%disp(x')


%% Conjugate Gradient Method with Jacobi preconditioner
disp('Conjugate gradient with Jacobi...')
M = D;
% 设定初始解 x 向量
x = zeros(length(b),1);
old_r = b;
old_d = inv(M) * old_r;
old_z = old_d;

zero_r = zeros(size(old_r));

Error_CJ = zeros(1, LOOP);
for count = 1: LOOP
    if old_r == zero_r
        break
    else
        alpha = (old_r' * old_z) / (old_d' * A * old_d);
        Conjugate_Jacobi = x + alpha * old_d;  % new x
        new_r = old_r - alpha * A * old_d;
        new_z = inv(M) * new_r;
        my_beta = (new_r' * new_z) / (old_r' * old_z);
        new_d = new_z + my_beta * old_d;
        % calculate error
        Error_CJ(count) = ErrorFunc(Conjugate_Jacobi, real_x);
        % update d r z x
        old_d = new_d;
        old_r = new_r;
        old_z = new_z;
        x = Conjugate_Jacobi;
    end
end
% 显示解
%disp('Conjugate Gradient method with Jacobi preconditioner:')
%disp(x')


%% 作误差图
figure(1)
% Jacobi
plot(1:LOOP, Error_Jacobi, 'ok-', 'linewidth', 1.1, 'markerfacecolor', [217, 57, 47]/255)
hold on
% Gauss
plot(1:LOOP, Error_Gauss_seidel, 'sk-', 'linewidth', 1.1, 'markerfacecolor', [78, 165, 236]/255)
% SOR
plot(1:LOOP, Error_SOR, '*k-', 'linewidth', 1.1, 'markerfacecolor', [53, 116, 66]/255)
% Conjugate
plot(1:LOOP, Error_Conjugate, 'xk-', 'linewidth', 1.1, 'markerfacecolor', [159, 144, 191]/255)
% Conjugate with Jacobi
plot(1:LOOP, Error_CJ, 'pk-', 'linewidth', 1.1, 'markerfacecolor', [132, 134, 134]/255)
% Figure 属性
set(gca, 'linewidth', 1.1, 'fontsize', 16, 'fontname', 'times')
xlabel('Epochs')
ylabel('Error')
title('Problem 1 Performance')
legend('Jacobi', 'Gauss', 'SOR', 'Conjugate', 'Con & Jacobi')
grid on

%% 保存答案至 excel
title = {"解析解","雅可比","高斯赛德","SOR","共轭梯度","共轭梯度 with 预条件"};
output = [real_x,Jacobi,Gauss_seidel,SOR,Conjugate,Conjugate_Jacobi];
output = [title; num2cell(output)];
```