作业一

姓名胡毅翔　学号 PB18000290　日期 2021年5月4日

第一题 **求解线性方程的迭代方法**

本题考虑使用有限差分方法(finite difference method)解决两点边值问题(boundary value problem)

$$-u''(x)=f(x)\quad (0<x<1)\ \text{使得}\ u(0)=T_0 \quad \text{and} \quad u(1)=T_1 \tag{1}$$

时产生的离散化线性系统

$$Ax=b \tag{2}$$

的求解问题。适当选取离散化的步长后我们会得到一个10×10的系统：

$$A=\begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & \ddots & & \\ & & \ddots & \ddots & -1 & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix},\quad b=\begin{bmatrix} 2 & -2 & 2 & -1 & 0 & 0 & 1 & -2 & 2 & -2 \end{bmatrix}^T \tag{3}$$

此处，A中空白部分的元素皆为0。我们容易验证上述线性系统的精确解为

$$x_{\text{exact}}=\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 \end{bmatrix}^T \tag{4}$$

(a) （20分）分别使用Jacobi和Gauss-Siedel方法求解上述问题。利用精确解(4)将误差大小和迭代次数的关系用semilogy图表示出来 (横轴为迭代次数 n, 纵轴为迭代解与精确解的差距)。

Jacobi和Gauss-Siedel方法运行结果见图1，算法见(b)题代码部分。

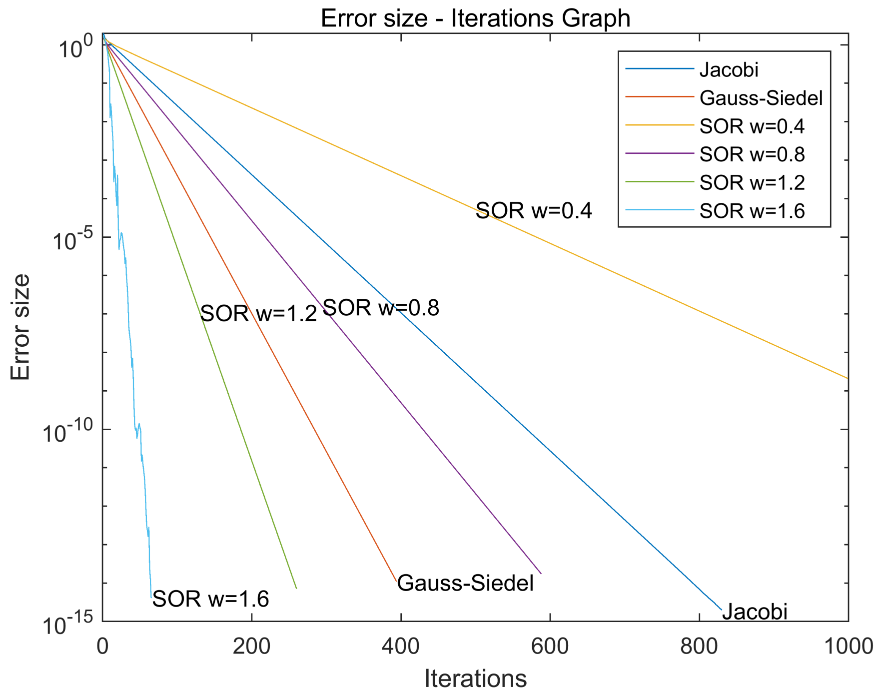


图 1: 迭代方法的误差大小与迭代次数的semilogy图

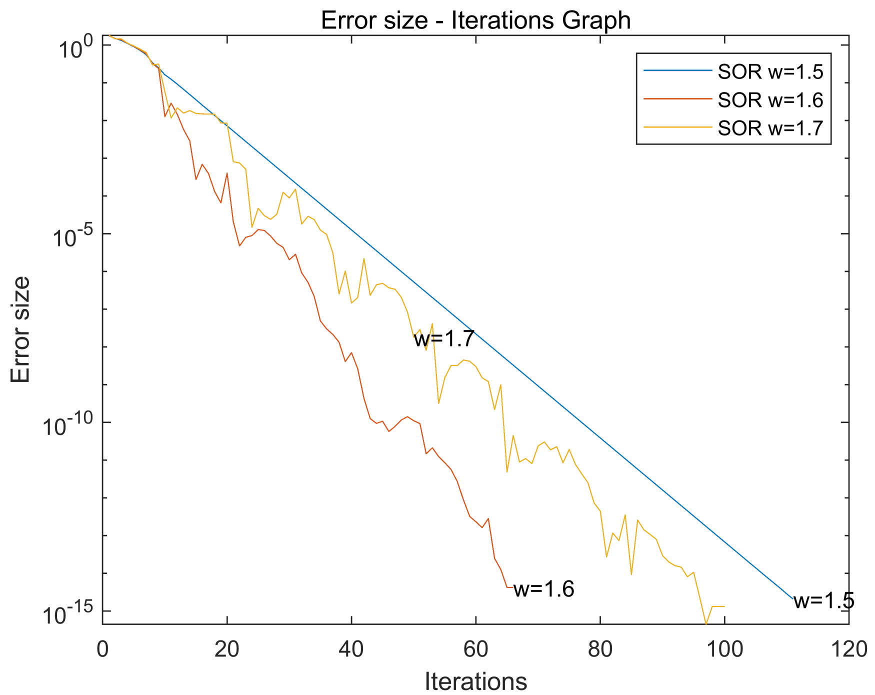


图 2: 不同 ω 的SOR迭代的误差大小与迭代次数的semilogy图

(b) （10分）选取若干不同的松他因子 ω 使用SOR方法解上述问题，并将收敛结果画在上一问的图中。请在图上相应的收敛线旁标示出这些 ω 的值。以迭代次数做为判断标准，指出对应于10^{-15} 的误差目标哪个大概的 ω 值收敛速度最快。

由图1分析，在 $\omega = 1.6$ 时收敛速度较快。在图2中，将 $\omega = 1.5, 1.6, 1.7$ 进行比较，得出结论：

在本题所给的数据下，对应于10^{-15} 的误差目标，$\omega = 1.6$ 时收敛速度较快。

注：本题代码部分为绘制图1所用的代码，绘制图2只需调用其中的SOR函数，修改参数 ω 的值并重新绘图即可。根据报告从简原则，不再重复。

(a),(b)两题所用MATLAB程序显示如下：

```
clear, clc;
% The value of A and B:
A = [2, -1, 0, 0, 0, 0, 0, 0, 0, 0;
    -1, 2, -1, 0, 0, 0, 0, 0, 0, 0;
    0, -1, 2, -1, 0, 0, 0, 0, 0, 0;
    0, 0, -1, 2, -1, 0, 0, 0, 0, 0;
    0, 0, 0, -1, 2, -1, 0, 0, 0, 0;
    0, 0, 0, 0, -1, 2, -1, 0, 0, 0;
    0, 0, 0, 0, 0, -1, 2, -1, 0, 0;
    0, 0, 0, 0, 0, 0, -1, 2, -1, 0;
    0, 0, 0, 0, 0, 0, 0, -1, 2, -1;
    0, 0, 0, 0, 0, 0, 0, 0, -1, 2];
b = [2; -2; 2; -1; 0; 0; 1; -2; 2; -2];
epsilon = 10^(-15);
% Jacobi Part:
x_exact = [1; 0; 1; 0; 0; 0; 0; -1; 0; 1];
tmp = abs(max(x_exact));
x_1 = [0; 0; 0; 0; 0; 0; 0; 0; 0; 0];
x_2 = [1; 1; 1; 1; 1; 1; 1; 1; 1; 1];
% initialization
D = diag(diag(A));
R = speye(10) - D \ A;
g = D \ b;
count_J = []; % count: used to save the error size
number_J = 1; % Iterations
```

```
while abs(max(x_1 - x_2)) > epsilon
    x_1 = x_2;
    x_2 = R * x_1 + g;
    count_J(number_J) = abs(max(x_2 - x_exact)) / tmp;
    number_J = number_J + 1;
end

% Print the output of Jacobi
x = x_2

% Gauss-Siedel Part:
x_1 = [0; 0; 0; 0; 0; 0; 0; 0; 0; 0];
x_2 = [1; 1; 1; 1; 1; 1; 1; 1; 1; 1];
% initialization
L = tril(A, -1);
U = triu(A, 1);
D = diag(diag(A));
S = -(D + L) \ U;
f = (D + L) \ b;
count_G = []; % count: used to save the error size
number_G = 1; % Iterations

while abs(max(x_1 - x_2)) > epsilon
    x_1 = x_2;
    x_2 = S * x_1 + f;
    count_G(number_G) = abs(max(x_2 - x_exact)) / tmp;
    number_G = number_G + 1;
end

% Print the output of Gauss-Siedel
x = x_2

% SOR Part:
% w=0.4:
```

```
[count_S_0, x, state] = SOR(0.4, A, x_exact, b);
% w=0.8:
[count_S_1, x, state] = SOR(0.8, A, x_exact, b);
% w=1.2:
[count_S_2, x, state] = SOR(1.2, A, x_exact, b);
% w=1.6:
[count_S_3, x, state] = SOR(1.6, A, x_exact, b);
semilogy([1:length(count_J)], count_J, ...
    [1:length(count_G)], count_G, ...
    [1:length(count_S_0)], count_S_0, ...
    [1:length(count_S_1)], count_S_1, ...
    [1:length(count_S_2)], count_S_2, ...
    [1:length(count_S_3)], count_S_3);
legend('Jacobi', 'Gauss-Siedel', 'SOR w=0.4', ...
    'SOR w=0.8', 'SOR w=1.2', 'SOR w=1.6', ...
    'Location', 'northeast');
xlabel('Iterations');
ylabel('Error size');
title('Error size - Iterations Graph');
text(length(count_J), ...
    count_J(length(count_J)), 'Jacobi');
text(length(count_G), ...
    count_G(length(count_G)), 'Gauss-Siedel');
text(length(count_S_0) / 2, ...
    count_S_0(floor(length(count_S_0) / 2)), 'SOR w=0.4');
text(length(count_S_1) / 2, ...
    count_S_1(length(count_S_1) / 2), 'SOR w=0.8');
text(length(count_S_2) / 2, ...
    count_S_2(length(count_S_2) / 2), 'SOR w=1.2');
text(length(count_S_3), ...
    count_S_3(length(count_S_3)), 'SOR w=1.6');

function [count, x, state] = SOR(w, A, x, b)
    x_exact = x;
    tmp = abs(max(x_exact));
```

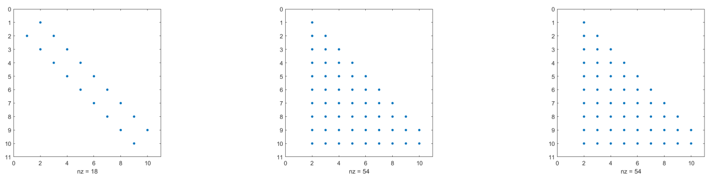
```
    x_1 = [0; 0; 0; 0; 0; 0; 0; 0; 0; 0];
    x_2 = [1; 1; 1; 1; 1; 1; 1; 1; 1; 1];
    % initialization
    L = tril(A, -1);
    U = triu(A, 1);
    D = diag(diag(A));
    D_INV = inv(D);
    epsilon = 10^(-15);
    S = (speye(10) + w * D_INV * L) \ ...
(speye(10) - w * (D_INV * U + speye(10)));
    f = w * inv(speye(10) + w * D_INV * L) * (D_INV * b);
    count = []; % count: used to save the error size
    number = 1; % Iterations

    while abs(max(x_1 - x_2)) > epsilon & number < 1000
        x_1 = x_2;
        x_2 = (S * x_1 + f);
        count(number) = abs(max(x_2 - x_exact)) / tmp;
        number = number + 1;
    end
    % if not convergence, return state = 0
    if number == 1000
        state = 0
    else
        state = 1
        % Print the output of SOR
        x_2
    end

end
```

(c) （15分）注意到题目中的矩阵A是一个稀疏矩阵(sparse matrix)，即有大量元素为0的矩阵。更改你的程序，省略那些和零元素相关的运算，使得你的程序得到加速。使用MATLAB中的tic和toc命令统计上述三种方法得到较为精确的解的时候的计算用时，并和改进后的程序的（在使用相同迭代次数的情况下的）耗时列表做对比（左边一列为未加速的程序的计算时间，右边一列为加速后的时间）。

注意你需要将每种方法反复运行N 次（比如10次）然后忽略第一次的运行时间，求后面 $N-1$ 次运行时间的平均值或者总和。这是由于MATLAB需要在第一次运算时对程序进行编译并分配存储空间。这类花费被统称为overhead，中文有时会勉强地将其译为“额外开销”。

因为迭代过程中的主要开销来自循环中对新的$\overrightarrow{X^{k+1}}$的计算，其中迭代矩阵和$\overrightarrow{X^{k}}$的乘法计算开销较大。因此，优化的思路为将乘法展开，只对非零元素进行计算。

对于Jacobi, Gauss-Siedel, SOR方法，迭代矩阵的非零元分布如下：



(a) Jacobi (b) Gauss-Siedel (c) SOR

图 3: 不同方法的迭代矩阵的非零元分布

根据非零元分布，设计优化算法，见代码部分，优化前后花销如表1。可以看出优化效果并不明显，甚至有的方法优化后时间开销反而更长。这与MATLAB的底层实现有关，可能对原来的矩阵乘法有并行化的处理。

Iterative method	**Before optimization**	**After optimization**
Jacobi	0.645113	0.588853
Gauss-Siedel	0.361099	0.470360
SOR ω=1.6	0.095594	0.121773

表 1: 不同迭代方法优化前后迭代1000次所用时间表(单位：s)

(c)题所用MATLAB程序显示如下：

```
% Jacobi Part:

while abs(max(x_1 - x_2)) > 10^(- 15)
    x_1 = x_2;
    x_2(1) = R(1, 2) * x_1(2);

    for i = 2:9
        x_2(i) = R(i, i - 1) * x_1(i - 1) + ...
            R(i, i + 1) * x_1(i + 1);
```

```
    end

    x_2(10) = R(10, 9) * x_1(9);
    x_2 = x_2 + g;
    %x_2 = R * x_1 + g;
    count_J(number_J) = abs(max(x_2 - x_exact)) / tmp;
    number_J = number_J + 1;
end

% Gauss-Siedel Part:

while abs(max(x_1 - x_2)) > 10^(- 15)
    x_1 = x_2;

    for i = 1:9
        x_2(i) = 0;

        for j = 2:i + 1
            x_2(i) = x_2(i) + S(i, j) * x_1(j);
        end

    end

    x_2(10) = 0;

    for j = 2:10
        x_2(10) = x_2(10) + S(10, j) * x_1(j);
    end

    x_2 = x_2 + f;
    count_G(number_G) = abs(max(x_2 - x_exact)) / tmp;
    number_G = number_G + 1;
end

% SOR Part:
```

```
while abs(max(x_1 - x_2)) > 10^(- 15) & number < 1000
    x_1 = x_2;

    for i = 1:9
        x_2(i) = 0;

        for j = 1:i + 1
            x_2(i) = x_2(i) + S(i, j) * x_1(j);
        end

    end

    x_2(10) = 0;

    for j = 1:10
        x_2(10) = x_2(10) + S(10, j) * x_1(j);
    end

    x_2 = x_2 + f;
    count(number) = abs(max(x_2 - x_exact)) / tmp;
    number = number + 1;
end
```

第二题 **Newton法求解线性方程**

本题将利用求解方程

$$x^3 - 3x^2 + 2 = 0 \tag{5}$$

的根来深入我们关于Newton方法的收敛速度的讨论。容易验证(??)的三个根分别位于 $[-3, 0], [0, 2], [2, 4]$ 三个区间内。我们依从左向右的顺序分别称这三个根为 x_l, x_m, x_r。

(a) （10分）适当选取迭代的初始点，写程序用Newton法求解这三个根，并将每一步迭代的新的近似值打印出来。

(b) （10分）设计一个估计收敛阶数的方法，在上一问求解的过程中同时求出大概的收敛阶数。

收敛阶数的估计，可用每一步计算出来的近似值作为精确解的估计，具体如下：

$$\alpha \approx \frac{\log\left|\left(x_{n+1}-x_n\right)/\left(x_n-x_{n-1}\right)\right|}{\log\left|\left(x_n-x_{n-1}\right)/\left(x_{n-1}-x_{n-2}\right)\right|} \tag{6}$$

对于三个根分别选取起始点，每步近似值及收敛阶数如下。

Iter	**x**	**order**
1	-1.8444444444444446	
2	-1.1637243901747001	1.3714391501568288
3	-0.8342806448688520	1.7240301893065662
4	-0.7400073072593034	1.9813265535540170
5	-0.7321050507279875	2.0094284562727243
6	-0.7320508101168153	2.0005928692388242
7	-0.7320508075688773	2.0000020944499259
8	-0.7320508075688773	2.0000000000492029

表 2: 每步近似值及收敛阶数（选取$x_l = -3$为起始点）

Iter	**x**	**order**
1	0.8888888888888888	
2	1.0009259259259260	2.8269574598756742
3	0.9999999994707780	2.9973992898742279
4	1.0000000000000000	2.9999999403585704
5	1.0000000000000000	3.0000000000000000

表 3: 每步近似值及收敛阶数（选取$x_m = 1.5$为起始点）

Iter	**x**	**order**
1	2.7777777777777777	
2	2.7337566137566136	2.0087046715549723
3	2.7320533217303780	2.0043558409397728
4	2.7320508075743515	2.0000999497969754
5	2.7320508075688772	2.0000000742240944
6	2.7320508075688772	2.0000000000000808

表 4: 每步近似值及收敛阶数（选取$x_r = 3$为起始点）

(c) （10分）Newton是一个二阶收敛的方法。上一问中你是否观测到了比二阶收敛更快的现象?如果有，请尽可能详细地解释其原因。

其中，在求x_m时，出现了比二阶收敛更快的现象，收敛阶数约为3。可能的原因为：

$$f(x) = x^3 - 3x^2 + 2 \tag{7}$$

求二阶导：

$$f''(x) = 6x - 6 \tag{8}$$

又由：

$$0 = f(\xi) = f(x_n) + (\xi - x_n)f'(x_n) + \frac{(\xi - x_n)_2}{2} f''(\eta_n) \tag{9}$$

可得：

$$\frac{\xi - x_{n+1}}{(\xi - x_n)^2} = -\frac{f''(\eta_n)}{2f'(x^n)} \tag{10}$$

由(??)知，$f''(1) = 0$。故(??),退化为：

$$0 = f(\xi) = f(x_n) + (\xi - x_n)f'(x_n) + \frac{(\xi - x_n)^2}{2} f''(x_n) + \frac{(\xi - x_n)^3}{6} f'''(\eta_n) \tag{11}$$

可得：

$$\frac{\xi - x_{n+1}}{(\xi - x_n)^3} \approx -\frac{f'''(\eta_n)}{6f'(x_n)} \tag{12}$$

即收敛阶数约为3，与数据实验结果相符。

(a)(b)两题所用MATLAB程序显示如下：

```matlab
clear,clc;
syms x;
maxrept = 1000;
f(x) = x^3 - 3 * x^2 + 2; %Equation to be solved
g(x) = diff(f, x); %df/dx
epsilon = 10^(-15);

fprintf("set x_l=-3,find out x_l in [-3,0]\n");
x_l = Newton(-3, f, g, epsilon, maxrept);
fprintf("The answer of x_l is %.16f\n", x_l);
fprintf("set x_m=1.5,find out the answer in [0,2]\n");
x_m = Newton(1.5, f, g, epsilon, maxrept);
fprintf("The answer of x_m is %.16f\n", x_m);
fprintf("set x_r=3,find out the answer in [2,4]\n");
x_r = Newton(3, f, g, epsilon, maxrept);
fprintf("The answer of x_m is %.16f\n", x_r);

```

```
function x = Newton(x_0, f, g, epsilon, maxrept)
    x_1 = x_0 - (f(x_0) / g(x_0));
    fprintf("iter        x                    order\n");

    for i = 1:maxrept
        x_2 = x_1 - (f(x_1) / g(x_1));

        if i >= 2
            % Calculate the order of convergence
            order_0 = order_1;
            order_1 = log(abs(x_2 - x_1) / ...
                abs(x_1 - x_0));
            Alpha = order_1 / order_0;
            fprintf("%d        %.16f     %.16f\n", ...
                i, x_1, Alpha);
        else
            % First iteration
            order_1 = log(abs(x_2 - x_1) / ...
                abs(x_1 - x_0));
            fprintf("%d        %.16f     \n", i, x_1);
        end

        % convergence or not
        if abs(x_1 - x_0) < epsilon
            break
        end

        % update
        x_0 = x_1;
        x_1 = x_2;
    end

    x = x_1;

end
```

第三题 幂法求解特征值问题

(a) （15分）设计一个能够求解问题存在一个（绝对值意义下的）最大特征值和存在最大的两个特征值，大小相同但符号相反的情况的算法并仿照课堂上所介绍的伪代码的格式写出一个清晰易懂的伪代码。

Algorithm 1 Power Method to Find Matrix Eigenvalues

Input: A: Matrix;
$maxrept$: the maximum number of iterations;
ε: error size;
$\overrightarrow{q^{old}}$: initial solution;

Output: λ: the eigenvalue of A which has the largest magnitude;
$\overrightarrow{q^{new}}$: the eigenvector of λ;
$errormessage$: if the input cannot be resolved by algorithm;

$\overrightarrow{q^{old}} \leftarrow \overrightarrow{q^{old}} / \left\|\overrightarrow{q^{old}}\right\|_\infty$;
for $i = 1$ **to** $maxrept$ **do**
    $\overrightarrow{q^{update}} \leftarrow A\ \overrightarrow{q^{old}}$;
    $index \leftarrow j\ \left(\left|\overrightarrow{q^{update}}_j\right| = \left\|\overrightarrow{q^{update}}\right\|_\infty\right)$;
    $flag \leftarrow \overrightarrow{q^{update}}_{index}$;
    $\overrightarrow{q^{new}} \leftarrow A\ \overrightarrow{q^{update}}$;
    $\lambda^2 \leftarrow \left\|\overrightarrow{q^{new}}\right\|_\infty$;
    $\overrightarrow{q^{new}} \leftarrow \overrightarrow{q^{new}}\ /\ \lambda^2$;
    if $\left\|\overrightarrow{q^{new}} - \overrightarrow{q^{old}}\right\|_\infty < \varepsilon$ **and** $flag > 0$ **then**
        return $\sqrt{\lambda^2}$, $\overrightarrow{q^{new}}$, **exit**
    else if $\left\|\overrightarrow{q^{new}} - \overrightarrow{q^{old}}\right\|_\infty < \varepsilon$ **and** $flag < 0$ **then**
        return $-\sqrt{\lambda^2}$, $\overrightarrow{q^{new}}$, **exit**
    else
        $\overrightarrow{q^{old}} \leftarrow \overrightarrow{q^{new}}$;
    end if
end for
error message;

(b) (10分）用程序实现上一问中你设计的算法，用于求解

$$A=\begin{bmatrix} -148 & -105 & -83 & -67 \\ 488 & 343 & 269 & 216 \\ -382 & -268 & -210 & -170 \\ 50 & 38 & 32 & 29 \end{bmatrix} \tag{13}$$

的模最大的特征值和特征向量（你提供的特征向量的 ∞ -范数应为1）。如果用你的程序求一 A 的模最大的特征值和特征向量呢?你需要保证你的程序对负值的特征值也有效。

(c) （10分）用程序实现第一问中你设计的算法，用于求解

$$\begin{bmatrix} 222 & 580 & 584 & 786 \\ -82 & -211 & -208 & -288 \\ 37 & 98 & 101 & 132 \\ -30 & -82 & -88 & -109 \end{bmatrix} \tag{14}$$

的模最大的特征值和特征向量（你提供的特征向量的∞ -范数应为1）。

(d) （10分）在MATLAB中设定随机数种子为rng(2)使用rand命令生成一个 100×100 的随机矩阵。用你的程序求解离 $0.8-0.6i$ 最近的特征值和特征向量（你提供的特征向量的 ∞ -范数应为1）。