MATLAB 第 12 次作业参考答案

几点反馈：

- 作业的参考答案来自同学们的作业，只是稍作调整，为保护隐私，隐去姓名。
- 本次作业大家做的普遍很好，成绩都很高。

第一题

$$\begin{cases} 2x-2y-z=-2 \\ 4x+y-2z=1 \\ -2x+y-z=-3 \end{cases}$$

使用高斯消去法求解上述方程组，并与 matlab 自带函数 \ 进行比较，给出结果和代码。

参考 1：

编写代码中首先假设高斯消去过程中默认的消元基准行是从第一行逐一到倒数第二行，这样的假设建立在消元过程中，新一行的第一个元素必须不是 0 才行，就本体的 3 阶方程这个结论刚好是成立的，因此编写了具有特殊性的 ge 函数，尽管采用的是高斯消元的思路，但只针对以上假设的成立条件函数结果才有效，于是又编写了监督消元基准行的强化 STRge 函数，可以接受满秩方阵的任意输入而不需要顾虑消元时出现的 0 项。

最终代码反复执行，保证内存预载充分后，比较 ge 与 STRge 和“\”函数的执行时间与结果：

```
时间已过 0.000044 秒。
时间已过 0.000069 秒。
时间已过 0.000030 秒。
```

求解结果均为[1;1;2]

可以发现 3 种算法的时间耗时级别是同级的，而 STRge 函数因为引入了相当多的监督过程的状态判断，所以耗时最长。

```
A=[[2 -2 -1];[4 1 -2];[-2 1 -1]];b=[-2;1;-3];A1=[[0 -1 -2];[4 1 -2];[-2 1 -1]];b1=[-5;1;-3];
tic;res1=ge(A,b);toc;
tic;res2=STRge(A1,b1);toc;
tic;res3=A\b;toc;
function [res] = ge(A,b)
n=size(A,1);M=[A,b];res=zeros(length(b)+1,1);
for b=1:n-1
    for a=b:n-1
        X=M(b,b);
        M(a+1,:)=M(a+1,:)-M(a+1,b)/X*M(b,:);
    end
end
for c=n:-1:1
    T=sum([M(c,c+1:end-1),1]'.*res(c+1:end));
    res(c)=(M(c,4)-T)/M(c,c);
end
res=res(1:3);
end
```

```
function [res] = STRge(A,b)
n=size(A,1);M=[A,b];res=zeros(length(b)+1,1);index=zeros(n-1,1);
for b=1:n-1
    base=judebase(M(b,1:end),index);index(b)=base;
    exceptb=1:n;
    for e=1:length(index)
        exceptb(find(exceptb==index(e)))=[];
    end
    for a=exceptb
        X=M(base,b);Y=M(base,:);
        M(a,:)=M(a,:)-M(a,b)/X*Y;
    end
end
last=n*(n+1)/2-sum(index);index=[index;last];
M=M(index,:);
for c=n:-1:1
    T=sum([M(c,c+1:end-1),1]'.*res(c+1:end));
    res(c)=(M(c,end)-T)/M(c,c);
end
res=res(1:end-1);
end
function [index] = judebase(T,index1)
index=1;
for a=1:length(T)
    if T(a)~=0&&(sum(index1==a)==0)
        index=a;
        break;
    end
end
end
```

参考 2：

```
mldivide求解的结果为：
[1.0000000000,1.0000000000,2.0000000000]
高斯消去求解的结果为：
[1.0000000000,1.0000000000,2.0000000000]
```

Hw12_1.m

```
clc;
clear;

A=[2 -2 -1;4 1 -2;-2 1 -1];
b=[-2 1 -3];
x1=A\b';
fprintf('mldivide 求解的结果为：\n[%.10f,%.10f,%.10f]\n',x1(1),x1(2),x1(3))
```

```
%不适用无穷多解情况
%高斯消去
rank_A=rank(A);
Ab=[A b'];
rank_Ab=rank(Ab);
m=rank_Ab;
n=size(Ab,2);
k=0;
num_pivot=0;
pivot=zeros(1,n-1); %所有列中的主元
for j=1:n %列
    %判断是否完成
    if num_pivot==m
        break;
    end
    %搜索各列主元
    if Ab(j,j)~=0
        pivot(j)=Ab(j,j);
        num_pivot=num_pivot+1;
        k=0;
    else
        k=k+1; %无主元列数
        continue;
    end
    for i=j+1-k:m %行
        Ab(i,:)=Ab(i,:)-Ab(j,:)*(Ab(i,j)/pivot(j));
    end
end
%回代
x(m)=Ab(m,end)/Ab(m,m);
for l=m-1:-1:1
    x(l)=(Ab(l,end)-Ab(l,l+1:end-1)*x(l+1:end)')/pivot(l);
end
fprintf('高斯消去求解的结果为：\n[%.10f,%.10f,%.10f]\n',x(1),x(2),x(3))
```

第二题 $\begin{bmatrix} 3 & 1 & 2 \\ 6 & 3 & 4 \\ 3 & 1 & 5 \end{bmatrix}$

编程实现对上述矩阵的 LU 分解，并与 matlab 自带函数 lu 进行比较，给出结果和代码。

这位同学编了 PA=LU，同时对 lu 函数的不同用法作了比较详细的解释

参考 1：

解：在 matlab 的内置函数中，进行的是 PA=LU 分解，若输出变量中没有 P，输出的 L 实际为 P'*L。为保证 LU 分解稳定性并方便与内置函数比较，自定义函数同样进行 PA=LU 分解。结果如下，可见两者的运算结果相同但是内置函数的运算速度要快一个数量级。

命令行窗口

```
自定义函数的运算结果为：
Lmy =

    1.0000         0         0
    0.5000    1.0000         0
    0.5000    1.0000    1.0000


Umy =

    6.0000    3.0000    4.0000
         0   -0.5000         0
         0         0    3.0000


Pmy =

     0     1     0
     1     0     0
     0     0     1

平均耗时（s）：
tmy =

   3.8832e-05
```

```
内置函数的运算结果为：
L =

    1.0000         0         0
    0.5000    1.0000         0
    0.5000    1.0000    1.0000


U =

    6.0000    3.0000    4.0000
         0   -0.5000         0
         0         0    3.0000


P =

     0     1     0
     1     0     0
     0     0     1

平均耗时（s）：
ti =

   1.9440e-06
```

myLU.m

```
function [L,U,P] = myLU(A)
% 进行 PA=LU 分解
    n=size(A,1);
    L=eye(n);% 初始化 L
    P=L;% 初始化 P
    for j=1:n-1
        [~,i]=max(abs(A(j:end,j)));% 找到主元位置
        i=i+j-1;
        P([j,i],:)=P([i,j],:);% 更新 P
        A([j,i],:)=A([i,j],:);% 更新 A
        L(j+1:n,j)=A(j+1:n,j)/A(j,j);% 更新 L
```

```
            invL=eye(n);
            invL(j+1:n,j)=-A(j+1:n,j)/A(j,j);
            A=invL*A;% 再次更新 A
        end
        U=A;% 输出 U
end
```

test2.m

```
clear;clc;
A=[3 1 2
    6 3 4
    3 1 5];
tmy=0;
ti=0;
% 自定义函数
fprintf('自定义函数的运算结果为：')
[Lmy,Umy,Pmy]=myLU(A)
for i=1:100
tic
[Lmy,Umy,Pmy]=myLU(A);
tmy=tmy+toc;
end
fprintf('平均耗时（s）：')
tmy=tmy/100
% 内置函数
fprintf('内置函数的运算结果为：')
[L,U,P]=lu(A)
for i=1:100
tic
[L,U,P]=lu(A);
ti=ti+toc;
end
fprintf('平均耗时（s）：')
ti=ti/100
```

参考 2：

如果只作 naïve LU 分解则是这个结果

手写LU分解结果：

Lm=

1	0	0
2	1	0
1	0	1

Um=

3	1	2
0	1	0
0	0	3

内置PA=LU分解结果：

L2=

0.5000	1.0000	0
1.0000	0	0
0.5000	1.0000	1.0000

U2=

6.0000	3.0000	4.0000
0	-0.5000	0
0	0	3.0000

P=

0	1	0
1	0	0
0	0	1

内置LU分解结果：

L1=

0.5000	1.0000	0
1.0000	0	0
0.5000	1.0000	1.0000

U1=

6.0000	3.0000	4.0000
0	-0.5000	0
0	0	3.0000

比较：

LU 分解结果不唯一，手写的 LU 分解只是将第一题的高斯消元的消元矩阵作了记录，属于 naïve Gaussian elimination，无法处理主元为 0、以及大数吃小数的情况，matlab 内置函数 lu 是带排序操作(partial pivoting protocol)的 lu 分解，这样可以避免上述两种误差。

Hw12_2.m

```
clc;
clear;
A=[3 1 2;6 3 4;3 1 5];
%lu 函数
[L2,U2,P]=lu(A);
disp('内置 PA=LU 分解结果：');
disp('L2=');disp(L1);
disp('U2=');disp(U1);
disp('P=');disp(P);

%手写 lu 分写
rank_A=rank(A);
m=rank_A;
n=size(A,2);
k=0;
num_pivot=0;
pivot=zeros(1,n); %所有列中的主元
Lm=eye(3);
for j=1:n %列
    %判断是否完成
    if num_pivot==m
        break;
    end
    %搜索各列主元
    if A(j,j)~=0
        pivot(j)=A(j,j);
        num_pivot=num_pivot+1;
```

```
        k=0;
    else
        k=k+1; %无主元列数
        continue;
    end
    for i=j+1-k:m %行
        Lm(i,j)=(A(i,j)/pivot(j));
        A(i,:)=A(i,:)-A(j,:)*(A(i,j)/pivot(j));
    end
end
Um=A;
disp('手写 LU 分解结果：');
disp('Lm=');disp(Lm);
disp('Um=');disp(Um);
```

第三题 $\begin{pmatrix} 2 & -1 & & & \\ 1 & 2 & -1 & & \\ & 1 & 2 & -1 & \\ & & 1 & 2 & -1 \\ & & & 1 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}=\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

实际应用中我们常常遇到三对角线性方程组的情况，这时可以使用追赶法加速求解。请自行搜索并自学追赶法，编程实现对上述方程组的求解，给出结果和代码。

三对角追赶法原理：赶法的基本原理是矩阵的 LU 分解，即将矩阵 A 分解为 A=LU
其中，L 为一个对角线上元素为 1 的下三角矩阵，U 为一个上三角矩阵. 容易验证，一个三对角矩阵作 LU 分解以后，得到一个下二对角矩阵与一个上二对角矩阵的乘积，即

$$A=\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & 0 \\ 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & 0 & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & 0 & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

$$L=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 & 0 & 0 \\ 0 & l_{32} & 1 & 0 & 0 & 0 \\ 0 & 0 & \cdots & \cdots & 0 & 0 \\ 0 & 0 & 0 & l_{n-1,n-2} & 1 & 0 \\ 0 & 0 & 0 & 0 & l_{n,n-1} & 1 \end{bmatrix}$$

$$U=\begin{bmatrix} u_{11} & u_{12} & 0 & 0 & 0 & 0 \\ 0 & u_{22} & u_{23} & 0 & 0 & 0 \\ 0 & 0 & u_{33} & u_{34} & 0 & 0 \\ 0 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 0 & 0 & u_{n-1,n-1} & u_{n-1,n} \\ 0 & 0 & 0 & 0 & 0 & u_{n,n} \end{bmatrix}$$

在计算过程中，将下三角矩阵 L 和上三角矩阵 U 的值保存在原矩阵 A 中。
实现方式为：

```
for i = 2:1:n
    A(i,i-1) = A(i,i-1)/A(i-1,i-1);
    A(i,i) = A(i,i) - A(i-1,i) * A(i,i-1);
end
```

$$A == \begin{bmatrix} u_{11} & u_{12} & 0 & 0 & 0 & 0 \\ l_{21} & u_{22} & u_{23} & 0 & 0 & 0 \\ 0 & l_{32} & u_{33} & u_{34} & 0 & 0 \\ 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & 0 & l_{n-1,n-2} & u_{n-1,n-1} & u_{n-1,n} \\ 0 & 0 & 0 & 0 & l_{n,n-1} & u_{n,n} \end{bmatrix}$$

分为两个过程：

第一个是追的过程，也就是分解的过程，令$Ux = y$

$$Ux = L^{-1}b$$

如下代码：

```
for i = 2 to n
    A(i,i-1) = A(i,i-1)/A(i-1,i-1);
    A(i,i) = A(i,i) - A(i-1,i) * A(i,i-1);
    b(i) = b(i) - b(i-1) * A(i,i-1);
end
```

循环里面的前两行与 LU 分解完全相同，第三行负责对常数项做相应的变换．在计算过程中，上三角矩阵 U 的值保存在原矩阵 A 中，变换后的常数$y = L^{-1}b$保存在 b 中

第二个过程是赶的过程，也就是回代的过程：

$$\begin{bmatrix} u_{11} & u_{12} & 0 & 0 & 0 & 0 \\ 0 & u_{22} & u_{23} & 0 & 0 & 0 \\ 0 & 0 & u_{33} & u_{34} & 0 & 0 \\ 0 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 0 & 0 & u_{n-1,n-1} & u_{n-1,n} \\ 0 & 0 & 0 & 0 & 0 & u_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \cdots \\ y_{n-1} \\ y_n \end{bmatrix}$$

如下代码：

```
x(n) = b(n) / A(i,i);
for i = n-1 to 1
    x(i) = (b(i) - A(i,i+1) * x(i+1)) / A(i,i);
end
```

最后结果：

```
追赶法得到的结果为：
    0.4143
   -0.1714
    0.0714
   -0.0286
    0.0143
```

此结果与 matlab 的\函数得到的结果一致 。

代码：

```
function x=Thomas(A,b)
```

```
n=length(b);
for i = 2:1:n
    A(i,i-1) = A(i,i-1)/A(i-1,i-1);
    A(i,i) = A(i,i) - A(i-1,i) * A(i,i-1);
    b(i) = b(i) - b(i-1) * A(i,i-1);
end
x(n) = b(n) / A(i,i);
for i = n-1:-1:1
    x(i) = (b(i) - A(i,i+1) * x(i+1)) / A(i,i);
end
x=x'
end
```

主程序：

```
clear all;
clc;
addpath(genpath('.'));
A=[2 -1 0 0 0;1 2 -1 0 0;0 1 2 -1 0;0 0 1 2 -1;0 0 0 1 2];
b=[1;0;0;0;0];
x1=Thomas(A,b);
x2=A\b;
disp('追赶法得到的结果为：');
disp(x1);
rmpath(genpath('.'))
```

第四题

$$\begin{bmatrix} 3 & -1 & 0 & 0 & 0 & \frac{1}{2} \\ -1 & 3 & -1 & 0 & \frac{1}{2} & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & 0 \\ 0 & \frac{1}{2} & 0 & -1 & 3 & -1 \\ \frac{1}{2} & 0 & 0 & 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ \frac{3}{2} \\ 1 \\ 1 \\ \frac{3}{2} \\ \frac{5}{2} \end{bmatrix}$$

使用 Jacobi Method、Gauss-Seidel Method 求解上述线性方程组，已知精确解为 x=[1,1,1,1,1,1]，要求结果精确到小数点后 3 位（使用 forward error with infinity norm），给出结果及迭代步数，并对两个方法进行比较，给出代码。

```
初值为：
x0=[0.0000,0.0000,0.0000,0.0000,0.0000,0.0000]
Jacobi法结果为：
x=[0.9999,0.9997,0.9996,0.9996,0.9997,0.9999]
迭代次数为：13
```

```
初值为：
x0=[0.0000,0.0000,0.0000,0.0000,0.0000,0.0000]
Gauss-Seidel法结果为：
x=[0.9997,0.9998,1.0000,1.0001,1.0002,1.0001]
迭代次数为：9
```

比较：Jacobi 法就是线性方程组的不动点迭代法，对于本题的严格对角占优矩阵，对任意初值均可收敛；Gauss-Seidel 法是对 Jacobi 法迭代公式的改进，使求同一向量不同元素时使用前一元素的计算结果而非只用前一向量的计算结果，能够加速收敛。

```
clc;
clear;
A=[3 -1 0 0 0 1/2;
    -1 3 -1 0 1/2 0;
    0 -1 3 -1 0 0;
    0 0 -1 3 -1 0;
    0 1/2 0 -1 3 -1;
    1/2 0 0 0 -1 3];
b=[5/2 3/2 1 1 3/2 5/2]';

TOL=0.5e-3;
x_real=ones(6,1);

%Jacobi
tic;
inival1=zeros(6,1);
x0=inival1;
D=diag(diag(A,0));
L=tril(A)-D;
U=triu(A)-D;
x1=D\(b-(L+U)*x0);
iterJ=1;
while norm(x1-x_real,Inf)>TOL
    x0=x1;
    x1=D\(b-(L+U)*x0);
    iterJ=iterJ+1;
end
t1=toc
fprintf('初值为：\nx0=[%.4f,%.4f,%.4f,%.4f,%.4f,%.4f]\n',...
    inival1(1),inival1(2),inival1(3),inival1(4),inival1(5),inival1(6));
fprintf('Jacobi 法结果为：\nx=[%.4f,%.4f,%.4f,%.4f,%.4f,%.4f]\n',...
    x1(1),x1(2),x1(3),x1(4),x1(5),x1(6));
fprintf('迭代次数为：%d \n',iterJ);

%Gauss-Seidel
```

精确到小数点后 3 位，TOL 应该取为 0.5e-3，这是为了保证四舍五入后仍保持小数点后 3 位不变

```
tic;
inival2=zeros(6,1);
xG0=inival2;
D=diag(diag(A,0));
L=tril(A)-D;
U=triu(A)-D;
xG1=(L+D)\(b-U*xG0);
iterG=1;
while norm(xG1-x_real,Inf)>TOL
    xG0=xG1;
    xG1=(L+D)\(b-U*xG0);
    iterG=iterG+1;
end
t2=toc
fprintf('初值为：\nx0=[%.4f,%.4f,%.4f,%.4f,%.4f,%.4f]\n',...
    inival2(1),inival2(2),inival2(3),inival2(4),inival2(5),inival2(6));
fprintf('Gauss-Seidel 法结果为：\nx=[%.4f,%.4f,%.4f,%.4f,%.4f,%.4f]\n',...
    xG1(1),xG1(2),xG1(3),xG1(4),xG1(5),xG1(6));
fprintf('迭代次数为：%d \n',iterG);
```