计算方法作业三

姓名 龚小航 学号 **PB18151866** 日期 **2020.12.17**

第一题 a) 任取 $1 < i, j \le n, i \ne j$，只需要证明在第一列消去后 $a_{ij}^{(1)} = a_{ji}^{(1)}$ 即可。

第 k 行第一步操作后结果是第一行乘以 $-\frac{a_{k1}}{a_{11}}$ 然后加到第 i 得到的。又由于矩阵 A 是对称矩阵，因此 $a_{mn} = a_{nm}$。综上，有：

$$
\begin{aligned}
a_{ij}^{(1)} &= a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j} && (1)\\
a_{ji}^{(1)} &= a_{ji} - \frac{a_{j1}}{a_{11}} a_{1i} && (2)\\
&= a_{ij} - \frac{a_{1j}}{a_{11}} a_{i1} && (3)\\
&= a_{ij}^{(1)} && (4)
\end{aligned}
$$

此即需要证明的结论，得出 $A^{(1)}$ 是对称矩阵。

b) LU 分解在本质上是高斯消元法的一种表达形式。实质上是将 A 通过初等行变换变成一个上三角矩阵，其变换矩阵就是一个单位下三角矩阵。利用对称性，构造以下伪代码：

正定对称矩阵 LU 分解算法：

```
for(diag=1;diag<=n;diag++){//以对角线元素来算
    l_{diag,diag} = 1;            //L 矩阵对角线上为 1
    for(row=diag+1;row<=n;row++){
        l_{row,diag} = -a_{row,diag}/a_{diag,diag};
    }
    for(idiag=diag;idiag<=n;idiag++){//idiag 遍历 diag 即其后的对角线元素
        for(row=diag+1;row<=n;row++){//更新矩阵 A 的元素，利用对称性节约计算
            a_{row,idiag} = a_{row,idiag} + a_{diag,idiag} * l_{row,diag}
            a_{idiag,row} = a_{row,idiag}
        }//最后得到的矩阵 A 即为上三角 U 矩阵
    }
}
```

图 1: 构造正定矩阵 LU 分解的算法

c) MATLAB 程序如下：

```
format rat
A=[4,-2,4,2;
   -2,10,-2,-7;
   4,-2,8,4;
   2,-7,4,7];
b=[8;2;16;6];
l=zeros(4,4);
n = 4;

%STEP1:求分解出的下三角矩阵L
for j = 1 : n
    sum2=0;
    for k=1:j-1
        sum2 = sum2+l(j,k)*l(j,k);
    end
    l(j,j) = sqrt(A(j,j)-sum2);
    for i = j+1 : n
        sum=0;
        for k=1:j-1
            sum  = sum+l(j,k)*l(i,k);
        end
        l(i,j) = (A(i,j)-sum)/l(j,j);
    end
end

l
l*l'

%STEP2:求解下三角方程组Ly=b得y
y = zeros(n,1);
y(1)=b(1)/l(1,1);
for i = 2 : n
    sum = 0;
    for k = 1 : i-1
        sum = sum + l(i,k)*y(k);
```

```
    end
    y(i) = (b(i)-sum)/l(i,i)
end

%STEP3:求解上三角方程组L^T x = y得x
x = zeros(n,1);
x(n) = y(n)/l(n,n);
for i = n-1 : -1 : 1
    sum = 0;
    for k = 1 : n
        sum = sum + l(k,i)*x(k);
    end
    x(i)=(y(i)-sum)/l(i,i);
end
x
```

运行结果如图所示。从图中可知分解出的下三角矩阵 L 以及解出的 x。

```
l =
     2     0     0     0
    -1     3     0     0
     2     0     2     0
     1    -2     1     1

ans =
     4    -2     4     2
    -2    10    -2    -7
     4    -2     8     4
     2    -7     4     7

y =
     4
     2
     4
     2

x =
     1
     2
     1
     2
```

图 2: 上述程序运行结果

第二题　$a)$ 先将迭代格式展开，将迭代矩阵表示出来：

$$\begin{aligned}
x^{(k+1)} &= (\frac{I}{\omega})^{-1}(\frac{I}{\omega}-A)x^{(k)}+(\frac{I}{\omega})^{-1}b \qquad (5)\\
&= \omega I(\frac{I}{\omega}-A)x^{(k)}+\omega Ib \qquad (6)\\
&= (I-\omega A)x^{(k)}+\omega b \qquad (7)
\end{aligned}$$

由于矩阵 A 是正定矩阵，因此若 A 的特征值记作 $\lambda_1,\lambda_2,\cdots,\lambda_n$ 时，必有 $\lambda_i>0, 0\le i\le n$；题中已经给出了 A 的最小/最大特征值为 λ_1,λ_n, 迭代矩阵的特征值为 $1-\omega\lambda_i$。

迭代收敛的充分必要条件是迭代矩阵的谱半径小于 1, 由于 $\lambda_i>0$, 只需要让迭代矩阵的特征值不小于 -1 即可。而 $\omega<2/\lambda_n$ 时，$1-\omega\lambda_i>-1$ 因此当 ω 满足题中给出的条件时迭代矩阵的谱半径必然小于 1。即迭代方法收敛。

$b)$ 对一个对称矩阵 T, 收敛速度 $R(T)=-ln\rho(T)$，因此想要收敛速度最快只需要令这个矩阵的谱半径尽量小即可。而谱半径是矩阵绝对值最大的特征值。因此 ω 的最佳值满足使 $max|1-\omega\lambda_i|, 1\le i\le n$ 最小。由题意可知 $\omega>0$，而 A 为正定矩阵，$\lambda_i>0$。因此 $\rho(I-\omega A)$ 随 λ 增大单调递减，随 ω 增大也单调递减。记 $F()$ 为矩阵的特征值，则有：

$$1-\omega\lambda_n\le F(I-\omega A)\le 1-\omega\lambda_1 \qquad (8)$$

而另一方面，ω 变化时最好的结果就是令 $\rho(I-\omega A)$ 的上限尽可能小，下限尽可能大。也就是说它们互为相反数，此时若 ω 稍大一点则下限负的更多，稍小一点则上限正的更多：

$$\begin{aligned}
1-\omega_b\lambda_n+1-\omega_b\lambda_1 &= 0 \qquad (9)\\
\Longrightarrow \omega_b &= \frac{2}{\lambda_1+\lambda_n} \qquad (10)
\end{aligned}$$

接下来计算迭代矩阵 $G_\omega=I-\omega A$ 的谱半径。由刚才得出的结论，若 ω 稍小则上限较大，最大特征值对应 $1-\omega\lambda_1$; 若 ω 稍大，则下限绝对值更大，此时对应 $1-\omega\lambda_n$，但在算谱半径时需要取绝对值; 若 ω 取最佳值时谱半径对

应 $1-\omega\lambda_n = 1-\frac{2}{\lambda_1+\lambda_n}\lambda_n = \frac{\lambda_n-\lambda_1}{\lambda_n+\lambda_1}$。综上:

$$\rho(G_\omega) = \begin{cases} 1-\omega\lambda_1, & \omega \le \omega_b \\ \dfrac{\lambda_n-\lambda_1}{\lambda_n+\lambda_1}, & \omega = \omega_b \\ \omega\lambda_n - 1, & \omega \ge \omega_b. \end{cases} \tag{11}$$

c) 程序如下所示:

```
%生成正定对称方阵, 特征值1, 2, 3, 4, 5
M=[1,0,0,0,0;
   0,2,0,0,0;
   0,0,3,0,0;
   0,0,0,4,0;
   0,0,0,0,5];

B=rand(5,5);
[Q,R]=qr(B);%利用Q矩阵得到正交阵

A = Q*M*Q' %同时作相合、相似变换
[x,y]=eig(A);%A即所需随机方阵
y
b = rand(5,1)

format long
x0=[0 0 0 0 0]';
%输出w=0.3情况下的迭代解以及迭代次数
[x,n]=richason(A,b,x0,10e-10,1000)

%以下求w和谱半径之间的关系
T = [1,2,3,4,5]';
w = 0:0.002:0.4; %w<2/5时收敛
p = max(abs(1.-w.*T))

plot(w, p, '-*');
xlabel('w');
ylabel('p(I-wA)');
```

```
function [x,n]=richason(A,b,x0,eps,M)
%Richardson法求解线性方程组 Ax=b
%方程组系数矩阵:A
%方程组之常数向量:b
%迭代初始向量:X0
%e解的精度控制:eps
%迭代步数控制: M
%返回值线性方程组的解: x
%返回值迭代步数: n

I =eye(size(A));
x1=x0;
x=(I-0.3*A)*x0+0.3*b;
n=1;

while(norm(x-x1)>eps)
    x1=x;
    x=(I-0.3*A)*x1+0.3*b;
    n = n + 1;
   if(n>=M)
       disp('Warning: 迭代次数太多, 现在退出! ');
       return;
   end
end

end
```

运行结果如下图所示:

可见随机构建的矩阵为 A,方程的迭代解为 x。最后迭代矩阵的谱半径随 ω 变化的情况如图所示,显然通过对几个点的采样就可以得出 (1) 的结论。

```
A =
   3.541371622414796  -0.453231220976768   0.237398599429906  -1.030349662012114  -0.986018727097795
  -0.453231220976768   3.519839103869801  -0.793552192261977  -0.408736852681887   0.540875856212270
   0.237398599429907  -0.793552192261976   1.980293040754737  -0.712232995579264  -0.362938955882612
  -1.030349662012114  -0.408736852681887  -0.712232995579264   2.978872779407781  -0.417183422932820
  -0.986018727097795   0.540875856212270  -0.362938955882612  -0.417183422932821   2.979623453552885

y =
   1.000000000000001                   0                   0                   0                   0
                   0   5.000000000000000                   0                   0                   0
                   0                   0   2.000000000000002                   0                   0
                   0                   0                   0   3.000000000000001                   0
                   0                   0                   0                   0   3.999999999999998

b =
   0.330423609606603
   0.619471555177780
   0.360636571002203
   0.756509543501944
   0.413900748690190

x =
   0.383420372008190
   0.373107902818039
   0.575735985620591
   0.625248028725220
   0.355734711297761

n =
    56

p =
   1.000000000000000   0.998000000000000   0.996000000000000   0.994000000000000   0.992000000000000   0.990000000000000
```

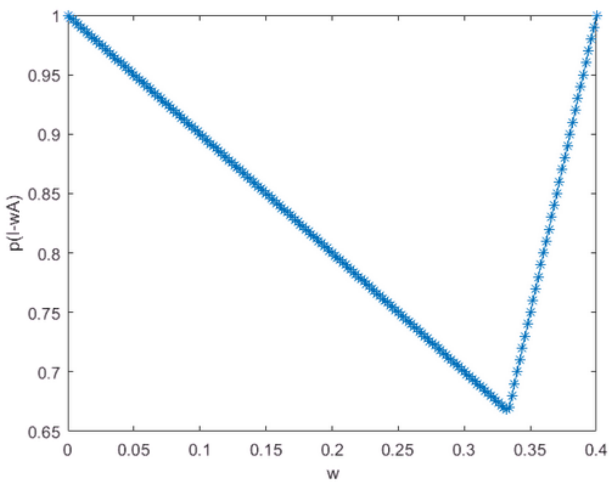

图 3: 上述程序运行结果图

第三题 $a)$ 高斯积分 $n=6$ 时，对于任意一个不高于 $2n-1$ 阶的多项式 $f(x)$ 都有：

$$\int_{-1}^{1} f(x)\mathrm{d}x = \sum_{i=1}^{6} \alpha_i f(x_i) \tag{12}$$

因此就可以得到 12 个方程组成的非线性方程组：

$$\int_{-1}^{1} x^k \mathrm{d}x = \sum_{i=1}^{6} \alpha_i x_i^k, \quad 0 \le k \le 11 \tag{13}$$

而其中 x_i, x_{7-i} 关于原点对称，可以简化一部分方程。例如奇次方程积分结果为 0。而高斯积分的积分节点和积分权重关于原点对称，因此只需要计算

6 个未知量即可。此处取正半轴的三个节点及其权重。

因此所需方程组为：

$$\int_{-1}^{1} x^k \mathrm{d}x = 2\sum_{i=1}^{3} \alpha_i x_i^k, \quad k = 0, 2, 4, 6, 8, 10 \tag{14}$$

总共六个方程。

b) 写出其雅可比行列式：

$$J(X) = \begin{pmatrix} \frac{\partial f_1(X)}{\partial x_1} & \frac{\partial f_1(X)}{\partial x_2} & \cdots & \frac{\partial f_1(X)}{\partial x_6} \\ \frac{\partial f_2(X)}{\partial x_1} & \frac{\partial f_2(X)}{\partial x_2} & \cdots & \frac{\partial f_2(X)}{\partial x_6} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_6(X)}{\partial x_1} & \frac{\partial f_6(X)}{\partial x_2} & \cdots & \frac{\partial f_6(X)}{\partial x_6} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 2 & 2 & 2 \\ 4x_1x_4 & 4x_2x_5 & 4x_3x_6 & 2x_1^2 & 2x_2^2 & 2x_3^2 \\ 8x_1^3x_4 & 8x_2^3x_5 & 8x_3^3x_6 & 2x_1^4 & 2x_2^4 & 2x_3^4 \\ 12x_1^5x_4 & 12x_2^5x_5 & 12x_3^5x_6 & 2x_1^6 & 2x_2^6 & 2x_3^6 \\ 16x_1^7x_4 & 16x_2^7x_5 & 16x_3^7x_6 & 2x_1^8 & 2x_2^8 & 2x_3^8 \\ 20x_1^9x_4 & 20x_2^9x_5 & 20x_3^9x_6 & 2x_1^{10} & 2x_2^{10} & 2x_3^{10} \end{pmatrix}$$

记$\alpha_1, \alpha_2, \alpha_3$为$x_4, x_5, x_6$，统一变量名称

图 4: 上述方程组的雅可比行列式

c) 在 $(-1, 1)$ 上等距选取六个积分节点即 $-1, -0.6, -0.2, 0.2, 0.6, 1$，在正半轴上即 $0.2, 0.6, 1$。初始的积分权重取为 $\alpha_1 = 1, \alpha_2 = 1, \alpha_3 = 1$，即等权值即可.

MATLAB 程序如下：

```
x0=[0.2 0.6 1 1 1 1];
[allx,ally,r,n]=mulNewton(fun,x0,1e-8)
%allx用于记录每一步迭代输入的点的矩阵，ally是每一步迭代
    利用迭代点算得的函数值
function [allx,ally,r,n]=mulNewton(F,x0,eps)
  if nargin==2
    eps=1.0e-8;
  end
  x0 = transpose(x0);
  Fx = subs(F,transpose(symvar(F)),x0);
  var = transpose(symvar(F));
  dF = jacobian(F,var);
  dFx = subs(dF,transpose(symvar(F)),x0);
  n=dFx;
  r=x0-inv(dFx)*Fx';
  n=1;
```

```
  tol=1;
  N=100;
  symx=length(x0);
  ally=zeros(symx,N);
  allx=zeros(symx,N);

  while tol>eps
    x0=r;
    Fx = subs(F,transpose(symvar(F)),x0);
    dFx = subs(dF,transpose(symvar(F)),x0);
    r=vpa(x0-inv(dFx)*Fx');
    tol=norm(r-x0)
    if(n>N)
        disp('迭代步数太多，可能不收敛！');
        break;
    end
    allx(:,n)=x0;
    ally(:,n)=Fx;
    n=n+1;
  end
end

function f = fun(x)
  k=6; %1~3为xi，4~6为ai
    for i=1:k
      x(i)=sym (['x',num2str(i)]);
    end
  f(1)=2*x(4)+2*x(5)+2*x(6)-2;
  f(2)=2*x(4)*(x(1))^2+2*x(5)*(x(2))^2+2*x(6)*(x(3))
     ^2-2/3;
  f(3)=2*x(4)*(x(1))^4+2*x(5)*(x(2))^4+2*x(6)*(x(3))
     ^4-2/5;
  f(4)=2*x(4)*(x(1))^6+2*x(5)*(x(2))^6+2*x(6)*(x(3))
     ^6-2/7;
  f(5)=2*x(4)*(x(1))^8+2*x(5)*(x(2))^8+2*x(6)*(x(3))
```

```
    ^8-2/9;
f(6)=2*x(4)*(x(1))^10+2*x(5)*(x(2))^10+2*x(6)*(x(3))
    ^10-2/11;
end
```

程序运行结果如下所示:

```
tol = 0.10818236307059877887154558407 52

tol = 0.060405808940703769641998854048717

tol = 0.016928205983225226482567308708485

tol = 0.00068806825103110246822279344797 85

tol = 0.00000037893074650590682652266031584555

tol = 0.0000000000000378838286240400855119253831656 05
allx =

    0.2410    0.2527    0.2443    0.2389    0.2386    0.2386         0         0         0 ...
    0.6473    0.7082    0.6701    0.6611    0.6612    0.6612         0         0         0
    0.9944    0.9594    0.9336    0.9324    0.9325    0.9325         0         0         0
    0.5568    0.5010    0.4789    0.4682    0.4679    0.4679         0         0         0
    0.3033    0.3625    0.3544    0.3602    0.3608    0.3608         0         0         0
    0.1399    0.1366    0.1667    0.1716    0.1713    0.1713         0         0         0
ally =

         0         0   -0.0000         0    0.0000         0         0         0         0 ...
   -0.0712    0.0123   -0.0006   -0.0000    0.0000    0.0000         0         0         0
   -0.0162    0.0179   -0.0004    0.0000   -0.0000    0.0000         0         0         0
    0.0297    0.0190   -0.0006    0.0000   -0.0000    0.0000         0         0         0
    0.0641    0.0197   -0.0010    0.0000   -0.0000    0.0000         0         0         0
    0.0906    0.0217   -0.0012    0.0000   -0.0000    0.0000         0         0         0
r =
```

$$\begin{pmatrix} 0.238619186083196908630501804465 26 \\ 0.661209386466264513661399508630 64 \\ 0.932469514203152027812301516148 4 \\ 0.467913934572691047389870497469 38 \\ 0.360761573048138607569833258591 5 \\ 0.171324492379170345040296243939 12 \end{pmatrix}$$

```
n = 7
```

图 5: 上述程序运行结果

从图中可知,最后得出的 r 即为六个未知数的解。即:

$$x = \pm 0.238619186, \quad \alpha = 0.467913935 \tag{15}$$

$$x = \pm 0.661209386, \quad \alpha = 0.360761573 \tag{16}$$

$$x = \pm 0.932469514, \quad \alpha = 0.171324492 \tag{17}$$

d) 若在 $(-1, 1)$ 上选取切比雪夫点,则选取 n 个点时 $x_i = cos(\frac{i\pi}{n})$。对不同的 n 来说,仍然可以列出 $2n-1$ 个含有 $2n-1$ 个未知数的非线性方程组

```
for n=2:2:20;
```

```
    [x,w]=gauss(n)
    x0=zeros(1,n);
    for i=1:n/2
        x0(i)=cos(i*pi/n);
    end
    for i=n/2+1:n
        x0(i)=cos(i*pi/n);
    end
    [allx,ally,r,n]=mulNewton(fun(n),x0,1e-8)

end

function [allx,ally,r,n]=mulNewton(F,x0,eps)
  if nargin==2
    eps=1.0e-8;
  end
  x0 = transpose(x0);
  Fx = subs(F,transpose(symvar(F)),x0);
  var = transpose(symvar(F));
  dF = jacobian(F,var);
  dFx = subs(dF,transpose(symvar(F)),x0);
  n=dFx;
  r=x0-inv(dFx)*Fx';
  n=1;
  tol=1;
  N=100;
  symx=length(x0);
  ally=zeros(symx,N);
  allx=zeros(symx,N);

  while tol>eps
    x0=r;
    Fx = subs(F,transpose(symvar(F)),x0);
    dFx = subs(dF,transpose(symvar(F)),x0);
    r=vpa(x0-inv(dFx)*Fx');
```

```
    tol=norm(r-x0)
    if(n>N)
        disp('迭代步数太多，可能不收敛！');
        break;
    end
    allx(:,n)=x0;
    ally(:,n)=Fx;
    n=n+1;
  end
end

function f = fun(n,x)
  k=n;
    for i=1:k
      x(i)=sym (['x',num2str(i)]);
    end
    for i=1:k
      sum=0;
      for j=1:k/2
         sum=sum+2*x(k/2+j)*x(j)^(2*i-2);
      end
      f(i)=sum-2/(2*i-1);
    end
end

function [x,w] = gauss(N)
  beta = .5./sqrt(1-(2*(1:N-1)).^(-2));
  T = diag(beta,1) + diag(beta,-1);
  [V,D] = eig(T);
  x = diag(D); [x,i] = sort(x);
  w = 2*V(1,i).^2;
end
```

对于计算方法来说，只需要将 c 问的程序函数定义改为由输入 n 自动生成，然后对不同的 n 反复调用即可。迭代算法在 n=8 时报错，n=6 时得出准确解. 再计算 n=7 时的结果，发现可以正常运行。因此 n 最大只能为 7 对比运

行结果，在精度 10^{-8} 下程序得到了准确解。

```
allx =

   1.0e+15 *                                                                                       ...

   -2.7219   -1.3609   -0.6805   -0.3402   -0.1701   -0.0851   -0.0425   -0.0213   -0.0106
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
ally =

   1.0e+31 *                                                                                       ...

         0         0         0         0         0         0         0         0         0
    1.4817    0.3704    0.0926    0.0232    0.0058    0.0014    0.0004    0.0001    0.0000
r =
```

$$\begin{pmatrix} -0.57735026918962576450914878374217 \\ 1.0 \end{pmatrix}$$

```
n = 58
```

图 6: n=2 时运行结果