第 5 章　MATLAB 数值计算

数值计算是有效使用数字计算机求数学问题近似解的方法与过程，主要研究如何利用计算机更好地解决各种数学问题，包括连续系统离散化和离散形式方程的求解，并考虑误差、收敛性和稳定性等问题。数值运算的研究领域包括数值逼近、数值微分和数值积分、数值代数、最优化方法、常微分方程数值解法、积分方程数值解法、偏微分方程数值解法、计算几何、计算概率统计等。MATLAB 提供了功能强大的命令和函数来解决这方面的数据分析和处理问题。本章介绍利用 MATLAB 进行线性方程组求解、非线性方程求解、插值与拟合、数值微分、数值积分、数据统计分析等。

5.1　线性方程组求解

科学计算中最常遇到的一个问题就是线性方程组的求解。本节介绍用于求解线性方程组的多种常用方法。

5.1.1　直接求解法

一个线性方程组通常可以表示为

$$\boldsymbol{Ax}=\boldsymbol{b} \tag{5.1}$$

在通常情况下，变量和方程的数目一样多。此时，$\boldsymbol{A}$ 为已知的 n 阶方阵，$\boldsymbol{b}$ 是含 n 个分量的已知列向量，而 $\boldsymbol{x}$ 为含 n 个分量的未知列向量。

例如，线性方程组

$$\begin{cases} 5x_1+4x_2-4x_3=11 \\ 2x_1-5x_2+6x_3=15 \\ -x_1+2x_2-3x_3=-9 \end{cases} \tag{5.2}$$

可以表示为

$$\begin{bmatrix} 5 & 4 & -4 \\ 2 & -5 & 6 \\ -1 & 2 & -3 \end{bmatrix} x=\begin{bmatrix} 11 \\ 15 \\ -9 \end{bmatrix} \tag{5.3}$$

1．逆矩阵解法

对于方程组 $\boldsymbol{Ax}=\boldsymbol{b}$，根据线性代数知识可知，$\boldsymbol{x}=\boldsymbol{A}^{-1}\boldsymbol{b}$，其中 $\boldsymbol{A}^{-1}$ 为矩阵 $\boldsymbol{A}$ 的逆矩阵。基于这一点，我们很容易通过矩阵运算求解线性方程组的解。

【例 5.1】采用逆矩阵解法求解式（5.3）所示方程组的解。

MATLAB 程序如下：

```
A=[5 4 -4;2 -5 6;-1 2 -3];
```

```
b=[11;15;-9];
x=inv(A)*b;            %inv表示求逆
```

运算结果如下：

```
x =
    3.0000
    3.0000
    4.0000
```

2．高斯消元法和列主元素法

高斯消元法是将一般的线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 中的矩阵 $\boldsymbol{A}$ 变换为三角（上三角）形式 $\boldsymbol{A}^{-1}$，构成三角形方程组 $\boldsymbol{A}'\boldsymbol{x}=\boldsymbol{b}'$。一般高斯消元法包括两个过程：先把方程组化为同解的上三角形方程组，再按相反顺序求解上三角形方程组，前者称为消元过程，后者称为回代过程。消元过程实际上是对增广矩阵 $\overline{\boldsymbol{A}}=[\boldsymbol{A}\ \ \boldsymbol{b}]$ 做初等行变换。具体过程可参考线性代数知识。

高斯法消元法求解线性方程组解的 MATLAB 程序代码如下：

```
function X=gaus(A,b)
    B=[A b];
   n=length(b);
   X=zeros(n,1);
 %%%%%%%%%%%%%%%%消元过程%%%%%%%%%%%%%%%%%
   for p=1:n-1
     for k=p+1:n
        m=B(k,p)/B(p,p);
        B(k,p:n+1)=B(k,p:n+1)-m*B(p,p:n+1);
     end
   end
 %%%%%%%%%%%%%%%%消元过程%%%%%%%%%%%%%%%%%

 %%%%%%%%%%%%%%%%回代过程%%%%%%%%%%%%%%%%%
      b=B(1:n,n+1);
      A=B(1:n,1:n);
      X(n)=b(n)/A(n,n);
      for q=n-1:-1:1
         X(q)=(b(q)-sum(A(q,q+1:n)*X(q+1:n)))/A(q,q);
      end
 %%%%%%%%%%%%%%%%回代过程%%%%%%%%%%%%%%%%%
end
```

【例 5.2】使用高斯消元法求方程组 $\begin{cases}5x_1+4x_2-4x_3=11\\2x_1-5x_2+6x_3=15\\-x_1+2x_2-3x_3=-9\end{cases}$ 的解。

解：将方程组写成 $\boldsymbol{Ax}=\boldsymbol{b}$ 的形式，分别写出 $\boldsymbol{A}$ 和 $\boldsymbol{b}$，在 MATLAB 工作窗口输入程序：

```
>>A=[5 4 -4;2 -5 6;-1 2 -3];
>>b=[11;15;-9];
>>x=gaus(A,b);      %高斯消元法求解方程
```

运算结果如下：

```
x =
    3.0000
```

```
    3.0000
    4.0000
```

由上述计算结果可以看出，采用高斯消元法所求解的结果与逆矩阵法一致。

列主元素消去法是为控制舍入误差而提出来的一种算法，在高斯消去法的消元过程中，若出现主对角线上的元素很小，就会导致其他元素量级的巨大增长和舍入误差的扩散，最后使计算结果不可靠。列主元素消去法此时需要进行初等行变换，将系数较大的列移至前面。使用列主元素消去法计算，基本上能控制舍入误差的影响，并且选主元素比较方便。列主元素消去法的具体过程请参考线性代数的相关知识。

列主元素消去法的 MATLAB 代码如下：

```
function X=liezhu(A,b)
   B=[A b];
   n=length(b);
   X=zeros(n,1);
   C=zeros(1,n+1);
%%%%%%%%%%%%%%%%消元过程%%%%%%%%%%%%%%%%%
   for p=1:n-1

      [Y,j]=max(abs(B(p:n,p)));   %查找系数最大的行
      C=B(p,:);
      B(p,:)=B(j+p-1,:);          %进行行掉换
      B(j+p-1,:)=C;
    for k=p+1:n
       m=B(k,p)/B(p,p);
       B(k,p:n+1)=B(k,p:n+1)-m*B(p,p:n+1);
    end
   end
%%%%%%%%%%%%%%%%消元过程%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%回代过程%%%%%%%%%%%%%%%%%
      b=B(1:n,n+1);
      A=B(1:n,1:n);
      X(n)=b(n)/A(n,n);
      for q=n-1:-1:1
         X(q)=(b(q)-sum(A(q,q+1:n)*X(q+1:n)))/A(q,q);
      end
%%%%%%%%%%%%%%%%回代过程%%%%%%%%%%%%%%%%%
end
```

【例 5.3】使用列主元素消去法求方程组 $\begin{cases}10x_1-7x_2=7\\-3x_1+2.099x_2+6x_3=3.901\\5x_1-x_2+5x_3=6\end{cases}$ 的解。

将方程组写成 $\boldsymbol{Ax}=\boldsymbol{b}$ 的形式，分别写出 $\boldsymbol{A}$ 和 $\boldsymbol{b}$，在 MATLAB 工作窗口输入程序：

```
>>A=[10 -7 0;-3 2.099 6;5 -1 5];
>>b=[7;3.901;6];
>>x=liezhu(A,b);
```

运算结果如下：

```
x =
   -0.0000
   -1.0000
    1.0000
```

对于例 5.3 所示的方程组，如果直接采用高斯消元法，若计算过程中有舍入误差，将导致最终计算结果产生较大偏差，读者可按照高斯消元法自行计算。

高斯消元法和列主元消元法的使用同样需要满足线性方程组有解的条件，并且其条件同逆矩阵法一致。

3．*LU* 分解法

LU 分解是矩阵分解的一种，可以将一个矩阵分解为一个下三角矩阵和一个上三角矩阵的乘积。设 ***A*** 是一个方阵，***A*** 的 ***LU*** 分解是将它分解成如下形式：

$$\boldsymbol{A}=\boldsymbol{LU} \tag{5.4}$$

其中 ***L*** 和 ***U*** 分别是下三角矩阵和上三角矩阵。***LU*** 分解主要应用在数值分析中，用来解线性方程、求反矩阵或计算行列式。***LU*** 分解的思想来源于高斯消元法，在高斯消元法的消元过程中，每一步都要进行一次初等行变换，初等行变换等价于在等式两边同时乘以一个下三角矩阵 $\boldsymbol{L}_k$，因此高斯消元法等价于

$$\begin{aligned}&\boldsymbol{Ax}=\boldsymbol{b}\\&\boldsymbol{L}_1\boldsymbol{Ax}=\boldsymbol{L}_1\boldsymbol{b}\\&\boldsymbol{L}_2\boldsymbol{L}_1\boldsymbol{Ax}=\boldsymbol{L}_2\boldsymbol{L}_1\boldsymbol{b}\\&\quad\vdots\\&\boldsymbol{L}_{n-1}\cdots\boldsymbol{L}_1\boldsymbol{Ax}=\boldsymbol{L}_{n-1}\cdots\boldsymbol{L}_1\boldsymbol{b}\end{aligned} \tag{5.5}$$

令 $\boldsymbol{L}_{n-1}\cdots\boldsymbol{L}_1\boldsymbol{A}=\boldsymbol{U},\boldsymbol{L}_{n-1}\cdots\boldsymbol{L}_1\boldsymbol{b}=\boldsymbol{y}$，则 $\boldsymbol{Ux}=\boldsymbol{y}$；令 $(\boldsymbol{L}_{n-1}\cdots\boldsymbol{L}_1)^{-1}=\boldsymbol{L}_1\cdots\boldsymbol{L}_n^{-1}=\boldsymbol{L}$，则 $\boldsymbol{Ly}=\boldsymbol{b}$，且 $\boldsymbol{A}=\boldsymbol{LU}$，也即 $\boldsymbol{Ax}=\boldsymbol{b}$ 可以分解为 $\begin{cases}\boldsymbol{Ux}=\boldsymbol{y}\\\boldsymbol{Ly}=\boldsymbol{b}\end{cases}$，其中 ***L*** 和 ***U*** 都是三角矩阵，***L*** 是下三角矩阵，***U*** 是上三角矩阵，分解后方程组的解可表示为

$$\boldsymbol{x}=\boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{b} \tag{5.6}$$

LU 分解的好处在于，对三角矩阵求逆比较简单方便，运算量较小。有关 ***LU*** 分解的详细过程可参考线性代数知识。

LU 分解的 MATLAB 程序代码如下：

```
function [L,U]=LUDecomp(A)
    [n n]=size(A);
    for j=1:n
       U(1,j)=A(1,j);
    end
    for k=2:n
       for i=2:n
             for j=2:n
                 L(1,1)=1;L(i,i)=1;
              if i>j
                 L(1,1)=1;L(2,1)=A(2,1)/U(1,1); L(i,1)=A(i,1)/U(1,1);
                 L(i,k)=(A(i,k)- L(i,1:k-1)*U(1:k-1,k))/U(k,k);
              else
```

```
                U(k,j)=A(k,j)-L(k,1:k-1)*U(1:k-1,j);
            end
        end
    end
end
```

【例 5.4】用 ***LU*** 分解法求线性方程组 $\begin{cases} x_1+2x_3=8 \\ x_2+x_4=5 \\ x_1+2x_2+4x_3+3x_4=25 \\ x_2+3x_4=7 \end{cases}$ 的解。

解：将方程组写成 $\boldsymbol{Ax}=\boldsymbol{b}$ 的形式，分别写出 $\boldsymbol{A}$ 和 $\boldsymbol{b}$，在 MATLAB 工作窗口输入程序：

```
>> A=[1 0 2 0;0 1 0 1;1 2 4 3;0 1 0 3];
>> b=[7;5;25;8];
>> [L,U]=LUDecomp(A)          %调用LU分解函数
>> x=inv(U)*inv(L)*b
```

运算结果如下：

```
 L =
1     0     0     0
0     1     0     0
1     2     1     0
0     1     0     1

 U =
1     0     2     0
0     1     0     1
0     0     2     1
0     0     0     2

 x =
2
4
3
1
```

5.1.2 迭代法

以上线性方程组的直接求解法是数学上的解析求解，而其数值求解法主要是采用迭代方式实现。迭代法的基本思想是，把 n 元线性方程组

$$\begin{cases} a_{11}x_1+a_{12}x_2+\cdots+a_{1n}x_n=b_1 \\ a_{21}x_1+a_{22}x_2+\cdots+a_{2n}x_n=b_2 \\ \vdots \\ a_{n1}x_1+a_{n2}x_2+\cdots+a_{nn}x_n=b_n \end{cases} \tag{5.7}$$

改写成等价的方程组

$$\begin{cases} x_1=m_{11}x_1+m_{12}x_2+\cdots+m_{1n}x_n+g_1 \\ x_2=m_{21}x_1+m_{22}x_2+\cdots+m_{2n}x_n+g_2 \\ \vdots \\ x_n=m_{n1}x_1+m_{n2}x_2+\cdots+m_{nn}x_n+g_n \end{cases} \tag{5.8}$$

迭代法是从某一取定的初始向量$\boldsymbol{x}^{(0)}$出发，按照适当的迭代公式，逐次计算出向量$\boldsymbol{x}^{(1)},\boldsymbol{x}^{(2)},\cdots$，使得向量序列$\{\boldsymbol{x}^{(k)}\}$收敛于方程组的精确解。迭代法是一类逐次近似的方法，其优点是算法简便，程序易于实现。由此建立方程组的迭代公式为

$$\boldsymbol{x}^{(k+1)}=\boldsymbol{M}\boldsymbol{x}^{(k)}+\boldsymbol{g},\quad k=0,1,2,\cdots \tag{5.9}$$

对任意取定的初始向量$\boldsymbol{x}^{(0)}$，由上式可逐次算出迭代向量$\boldsymbol{x}^{(k)}$, k = 1, 2, …，如果向量序列$\{\boldsymbol{x}^{(k)}\}$收敛于$\boldsymbol{x}^{*}$，即

$$\boldsymbol{x}^{*}=\boldsymbol{M}\boldsymbol{x}^{*}+\boldsymbol{g} \tag{5.10}$$

则$\boldsymbol{x}^{*}$是方程组$\boldsymbol{x}=\boldsymbol{M}\boldsymbol{x}+\boldsymbol{g}$的解，也就是方程组$\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$的解。

这种求解线性方程组的方法称为迭代法，若迭代序列$\{\boldsymbol{x}^{(k)}\}$收敛，则称迭代法收敛，否则称迭代法发散。

1．雅可比迭代法

雅可比迭代法是由方程组（5.5）中的第 k 个方程解出$\boldsymbol{x}^{(k)}$，得到等价方程组：

$$\begin{cases} x_1=-\frac{a_{12}}{a_{11}}x_2-\frac{a_{13}}{a_{11}}x_3-\cdots-\frac{a_{1n}}{a_{11}}x_n+\frac{b_1}{a_{11}} \\ x_2=-\frac{a_{21}}{a_{22}}x_1-\frac{a_{23}}{a_{22}}x_3-\cdots-\frac{a_{2n}}{a_{22}}x_n+\frac{b_2}{a_{22}} \\ \vdots \\ x_n=-\frac{a_{n1}}{a_{nn}}x_1-\frac{a_{n2}}{a_{nn}}x_2-\cdots-\frac{a_{nn-1}}{a_{nn}}x_{n-1}+\frac{b_n}{a_{nn}} \end{cases} \tag{5.11}$$

从而得迭代公式

$$\begin{cases} x_1^{(k+1)}=-\frac{a_{12}}{a_{11}}x_2^{(k)}-\frac{a_{13}}{a_{11}}x_3^{(k)}-\cdots-\frac{a_{1n}}{a_{11}}x_n^{(k)}+\frac{b_1}{a_{11}} \\ x_2^{(k+1)}=-\frac{a_{21}}{a_{22}}x_1^{(k)}-\frac{a_{23}}{a_{22}}x_3^{(k)}-\cdots-\frac{a_{2n}}{a_{22}}x_n^{(k)}+\frac{b_2}{a_{22}} \\ \vdots \\ x_n^{(k+1)}=-\frac{a_{n1}}{a_{nn}}x_1^{(k)}-\frac{a_{n2}}{a_{nn}}x_2^{(k)}-\cdots-\frac{a_{nn-1}}{a_{nn}}x_{n-1}^{(k)}+\frac{b_n}{a_{nn}},k=1,2,3,\cdots \end{cases} \tag{5.12}$$

式（5.10）称为雅可比迭代法，简称为 J 迭代法。J 迭代法也记为

$$x_i^{(k+1)}=\frac{1}{a_{ii}}\left(b_i-\sum_{j=1}^{i-1}a_{ij}x_j^{(k)}-\sum_{j=i+1}^{n}a_{ij}x_j^{(k)}\right),\quad i=1,2,\cdots,n,\quad k=0,1,2,\cdots \tag{5.13}$$

J 迭代法的迭代矩阵为

$$\boldsymbol{M}_J=\begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} & \cdots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & \cdots & -\frac{a_{2n}}{a_{22}} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & \cdots & 0 \end{bmatrix} \tag{5.14}$$

若记$\boldsymbol{g}=\left[\frac{b_1}{a_{11}},\frac{b_2}{a_{22}},\cdots,\frac{b_n}{a_{nn}}\right]^{\mathrm{T}}$，则 J 迭代法可写成

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{M}_J x^{(k)} + \boldsymbol{g},\ \ k = 0,1,2,\cdots \tag{5.15}$$

用雅可比迭代解线性方程组 $\boldsymbol{Ax} = \boldsymbol{b}$ 的 MATLAB 程序如下：

```
function X=jacdd(A,b,X0,detx,maxN)
  %A,b为方程组Ax=b中的矩阵A和向量b
  %X0为x的初始值
  %detx迭代终止条件，如果两次迭代x的距离（二范数）小于detx，则停止迭代
  %maxN为最大迭代次数，如果经过maxN次迭代仍不收敛，则停止迭代
  [n m]=size(A);
    for k=1:maxN
     for j=1:m
       X(j)=(b(j)-A(j,[1:j-1,j+1:m])*X0([1:j-1,j+1:m]))/A(j,j);
     end
     djwcX=norm(X'-X0);
     xdwcX=djwcX/(norm(X')+eps);
     X0=X';
     if (djwcX<detx)&(xdwcX<detx)
       return
    end
  end
  if (djwcX>detx)&(xdwcX>detx)
     disp('雅可比迭代次数已经超过最大迭代次数maxN')
  end
```

【例 5.5】用雅可比迭代法求解线性方程组 $\begin{cases} 10x_1 + 3x_2 + x_3 = 14 \\ 2x_1 - 10x_2 + 3x_3 = -5 \\ x_1 + 3x_2 + 10x_3 = 14 \end{cases}$ 的近似解。

MATLAB 程序如下：

```
A=[10 3 1;2 -10 3;1 3 10];
b=[14;-5;14];
X0=[0;0;0];
detx=0.00001;
maxN=20;
X=jacdd(A,b,X0,detx,maxN)     %调用雅可比迭代法函数
```

运行结果如下：

```
X =
1.0000    1.0000    1.0000
```

通过雅可比迭代法所得的结果与该方程组的精确解[1 1 1]十分接近，但通过该迭代法计算量非常小，通过 20 次以下的迭代即得到了与精确解非常近似的解。但雅可比迭代法并非任意情况下都能收敛，并得到较好的近似解。对于线性方程组 $\boldsymbol{Ax} = \boldsymbol{b}$，雅可比迭代法收敛必须满足：矩阵 $\boldsymbol{A}$ 为严格对角占优阵，则雅可比迭代法收敛。判断雅可比迭代法收敛的 MATLAB 程序如下：

```
function X=jacddPD(A)
 [n m]=size(A);
   for j=1:m
      a(j)=sum(abs(A(:,j)))-2*(abs(A(j,j)));
   end
   for i=1:n
```

```
            if a(i)>=0
                disp('系数矩阵A不是严格对角占优的，此雅可比迭代不一定收敛')
                return
            end
        end
        if a(i)<0
            disp('系数矩阵A是严格对角占优的，此方程组有唯一解，且雅可比迭代收敛')
        end
```

【例 5.6】用雅可比迭代法收敛条件判断以下线性方程组雅可比迭代法是否收敛。

$$(1)\begin{cases}10x_1-x_2-2x_3=7.2\\-x_1+10x_2-2x_3=8.3\\-x_1-x_2+5x_3=4.2\end{cases};\qquad (2)\begin{cases}10x_1-x_2-2x_3=7.2\\-x_1+10x_2-2x_3=8.3\\-x_1-x_2+0.5x_3=4.2\end{cases}$$

（1）在 MATLAB 工作窗口输入程序：

```
>>A=[10 -1 -2;-1 10 -2;-1 -1 5];
>>a=jacddPD(A)
```

运行结果如下：

```
系数矩阵A是严格对角占优的，此方程组有唯一解，且雅可比迭代收敛
a =
    -8    -8    -1
```

（2）在 MATLAB 工作窗口输入程序：

```
>>A=[10 -1 -2;-1 10 -2;-1 -1 0.5];
>>a=jacddPD(A)
```

运行结果如下：

```
系数矩阵A不是严格对角占优的，此雅可比迭代不一定收敛
a =
       -8.0000   -8.0000    3.5000
```

2. 高斯-塞德尔迭代法

若在雅可比迭代法中，如果充分利用每一步迭代得到的新值 $x_i^{(k+1)}$，则可以得到如下的迭代公式：

$$\begin{cases}x_1^{(k+1)}=-\frac{a_{12}}{a_{11}}x_2^{(k)}-\frac{a_{13}}{a_{11}}x_3^{(k)}-\cdots-\frac{a_{1n}}{a_{11}}x_n^{(k)}+\frac{b_1}{a_{11}}\\x_2^{(k+1)}=-\frac{a_{21}}{a_{22}}x_1^{(k)}-\frac{a_{23}}{a_{22}}x_3^{(k)}-\cdots-\frac{a_{2n}}{a_{22}}x_n^{(k)}+\frac{b_2}{a_{22}}\\\vdots\\x_n^{(k+1)}=-\frac{a_{n1}}{a_{nn}}x_1^{(k+1)}-\frac{a_{n2}}{a_{nn}}x_2^{(k+1)}-\cdots-\frac{a_{nn-1}}{a_{nn}}x_{n-1}^{(k+1)}+\frac{b_n}{a_{nn}},k=1,2,3,\cdots\end{cases}\tag{5.16}$$

式（5.16）称为高斯-塞德尔（Gauss-Seidel）迭代法，简称为 G-S 迭代法。

G-S 迭代法也可记为

$$x_i^{(k+1)}=\frac{1}{a_{ii}}\left(b_i-\sum_{j=1}^{i-1}a_{ij}x_j^{(k+1)}-\sum_{j=i+1}^{n}a_{ij}x_j^{(k)}\right),\ i=1,2,\cdots,n,\ k=0,1,2,\cdots\tag{5.17}$$

G-S 迭代法的 MATLAB 程序如下：

```
function X=gsdd(A,b,X0,detx,maxN)
    %A,b为方程组Ax=b中的矩阵A和向量b
    %X0为x的初始值
    %detx迭代终止条件，如果两次迭代x的距离（二范数）小于detx，则停止迭代
```

```
    %maxN为最大迭代次数，如果经过maxN次迭代仍不收敛，则停止迭代
    [n m]=size(A);
   X=X0;
    for k=1:maxN
       for j=1:m
          if j==1
             X(1)=(b(1)-A(1,2:m)*X0(2:m))/A(1,1);
          elseif j==m
             X(m)=(b(m)-A(m,1:m-1)*X(1:m-1)')/A(m,m);
          else
             X(j)=(b(j)-A(j,1:j-1)*X(1:j-1) -A(j,j+1:m)*X(j+1:m))/ A(j,j);
          end

       end
       djwcX=norm(X'-X0);
       xdwcX=djwcX/(norm(X')+eps);
       X0=X';
       if (djwcX<detx)&(xdwcX<detx)
          return
     end
    end
    if (djwcX>detx)&(xdwcX>detx)
       disp('雅可比迭代次数已经超过最大迭代次数maxN')
    end
```

【例 5.7】用 G-S 迭代法求解线性方程组 $\begin{cases}10x_1+3x_2+x_3=14\\2x_1-10x_2+3x_3=-5\\x_1+3x_2+10x_3=14\end{cases}$ 的近似解。

MATLAB 代码如下：

```
A=[10 3 1;2 -10 3;1 3 10];
     b=[14;-5;14];
     X0=[0;0;0];
     detx=0.00001;
     maxN=20;
     X=gsdd(A,b,X0,detx,maxN)   %调用Gauss-Seidel迭代函数
```

运行结果如下：

```
X =
1.0000    1.0000    1.0000
```

以上结果表明通过 G-S 迭代法在 20 次之内可以很好地收敛并十分接近精确结果（精确结果为[1 1 1]）。例 5.8 和例 5.6 分别用的雅可比迭代法和 G-S 迭代法对同一方程组进行了迭代求解，从 MATLAB 显示结果来看，似乎效果完全一样。为了能更好地查看两种算法的效果差别，我们改变 MATLAB 结果显示的精度（默认情况只显示小数点后 4 位），并将迭代次数改为 8 次，分别用雅可比迭代法和 G-S 迭代法对上述方程组再进行求解，其结果如下：

雅可比迭代法显示结果：

```
雅可比迭代次数已经超过最大迭代次数maxN
X =
  1.000138710000000   0.999118200000000   1.000138710000000
```

G-S 迭代法显示结果：

```
X =
   1.000001301799667   1.000000930880776   0.999999590555801
```

可以看出，如果最多只迭代 8 次，此时雅可比迭代法还没有收敛至迭代终止条件，而 G-S 迭代法已经达到迭代终止条件。通过二者得到的结果也可以看出 G-S 迭代法与精确结果更为接近。

G-S 迭代法的收敛条件同雅可比迭代法一致，可以通过上述雅可比迭代法收敛判断程序对 G-S 迭代法的收敛性进行判断。

3. 超松弛迭代法

将雅可比迭代法的迭代公式（5.15）改写成

$$x_i^{(k+1)} = x_i^{(k)} + \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} \right),\ \ i = 1,2,\cdots,n,\ \ k = 0,1,2,\cdots \tag{5.18}$$

或写成向量形式

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{D}^{-1}(\boldsymbol{b} + \boldsymbol{L}\boldsymbol{x}^{(k+1)} + (\boldsymbol{U} - \boldsymbol{D})\boldsymbol{x}^{(k)}),\ \ k = 0,1,2,\cdots \tag{5.19}$$

构造迭代公式

$$x_i^{(k+1)} = x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} \right),\ \ i = 1,2,\cdots,n,\ \ k = 0,1,2,\cdots \tag{5.20}$$

此迭代法称为 SOR 松弛迭代法，其中参数 ω 称为松弛因子，当 $\omega > 1$ 时称为超松弛迭代，当 $\omega < 1$ 时称为欠松弛迭代。其矩阵形式为

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \omega\boldsymbol{D}^{-1}\left(\boldsymbol{b} + \boldsymbol{L}\boldsymbol{x}^{(k+1)} + (\boldsymbol{U} - \boldsymbol{D})\boldsymbol{x}^{(k)}\right),\ \ k = 0,1,2,\cdots \tag{5.21}$$

式中 $\boldsymbol{L}$、$\boldsymbol{U}$ 分别为矩阵 $\boldsymbol{A}$ 的下三角矩阵（不含对角线元素）和上三角矩阵（不含对角线元素），$\boldsymbol{D}$ 为矩阵 $\boldsymbol{A}$ 的对角矩阵。整理可得

$$\boldsymbol{x}^{(k+1)} = (\boldsymbol{D} - \omega\boldsymbol{L})^{-1}[(1-\omega)\boldsymbol{D} + \omega\boldsymbol{U}]\boldsymbol{x}^{(k)} + \omega(\boldsymbol{D} - \omega\boldsymbol{L})^{-1}\boldsymbol{b},\quad k = 0,1,2,\cdots \tag{5.22}$$

因此，SOR 方法的迭代矩阵为

$$\mathfrak{I}_{\omega} = (\boldsymbol{D} - \omega\boldsymbol{L})^{-1}[(1-\omega)\boldsymbol{D} + \omega\boldsymbol{U}] \tag{5.23}$$

有关 SOR 迭代法原理及推导过程的详细内容，请参见数值分析相关的内容。SOR 迭代法的 MATLAB 代码如下：

```
function X=sordd(A,b,X0,omg,detx,maxN)
  %A,b为方程组Ax=b中的矩阵A和向量b
   %X0为x的初始值
   %omg为松弛系数
   %detx为迭代终止条件，如果两次迭代X的距离（二范数）小于detx，则停止迭代
   %maxN为最大迭代次数，如果经过maxN次迭代仍不收敛，则停止迭代
[n m]=size(A);
  D=diag(diag(A));
  U=-triu(A,1);
  L=-tril(A,-1);
  jX=A\b;
  iD=inv(D-omg*L);
  B2=iD*(omg*U+(1-omg)*D);
```

```
    H=eig(B2);
    mH=norm(H,inf);
     for k=1:maxN
        iD=inv(D-omg*L); B2=iD*(omg*U+(1-omg)*D);
        f2=omg*iD*b;
        X=B2*X0+f2; X0=X;
        djwcX=norm(X-jX,inf);
        xdwcX=djwcX/(norm(X,inf)+eps);
        if (djwcX<detx)|(xdwcX<detx)
           return;
        end
     end
     disp('迭代次数已经超过最大迭代次数');
```

【例 5.8】用 SOR 迭代法求解线性方程组 $\begin{cases}4x_1-2x_2-x_3=10\\-2x_1+17x_2+10x_3=3\\-4x_1+10x_2+9x_3=-7\end{cases}$，该方程组的精确解是 $\boldsymbol{x}^*=(2,1,-1)^{\mathrm{T}}$。

取 $\omega=1.5$，MATLAB 代码如下：

```
A=[4 -2 -4;-2 17 10;-4 10 9]
b=[10;3;-7]
X=sor(A,b,X0,1.5,0.0001,20)      %调用SOR迭代法函数
```

计算结果如下：

```
X =
    1.9999
    1.0001
   -1.0001
```

SOR 迭代法在 20 次迭代以内收敛，并取得了较为接近精确解的结果。若取 $\omega=1.1$，做相同的运算，其结果如下：

```
迭代次数已经超过最大迭代次数
X =
    2.0301
    0.9784
   -0.9632
```

以上结果表明，当 $\omega=1.1$ 时，通过 20 次迭代与精确解还有较大的误差。测试表明，要想达到预设的误差范围，迭代次数需达到 50 次以上。可见 SOR 迭代法的收敛速度与系数 ω 有关。SOR 法收敛的充要条件为 $\rho(\mathfrak{I}_\omega)<1$，即 SOR 方法的迭代矩阵的谱半径小于 1。判断 SOR 法收敛的 MATLAB 程序如下：

```
function H=sorddPD(A,omg)
    D=diag(diag(A));
    U=-triu(A,1);
    L=-tril(A,-1);
    iD=inv(D-omg*L);
    B2=iD*(omg*U+(1-omg)*D);
    H=eig(B2);
    mH=norm(H,inf);
    if mH>=1
        disp('因为谱半径不小于1，所以超松弛迭代序列发散，谱半径mH和B的所有特征值H如下：')
    else
```

```
        disp('因为谱半径小于1，所以超松弛迭代序列收敛，谱半径mH和B的所有特征值H如下：')
    end
    mH
```

【例 5.9】取 $\omega=1.15$ 和 3 时，分别判断方程组 $\begin{cases}5x_1+x_2-x_3-2x_4=4\\2x_1+8x_2+x_3+3x_4=1\\x_1-2x_2-4x_3-x_4=6\\-x_1+3x_2+2x_3+7x_4=-3\end{cases}$ 使用 SOR 方法的收敛性。

（1）取 $\omega=1.15$ 时，在 MATLAB 工作窗口输入程序：

```
>>A=[5 1 -1 -2;2 8 1 3;1 -2 -4 -1;-1 3 2 7];
>>H=sorddPD(A,1.15)          %调用收敛性判断函数
```

运行结果如下：

```
因为谱半径小于1，所以超松弛迭代序列收敛，谱半径mH和B的所有特征值H如下：
mH =
    0.1596
H =
   0.1049 + 0.1203i
   0.1049 - 0.1203i
  -0.1295 + 0.0556i
  -0.1295 - 0.0556i
```

（2）取 $\omega=3$ 时，在 MATLAB 工作窗口输入程序：

```
>>A=[5 1 -1 -2;2 8 1 3;1 -2 -4 -1;-1 3 2 7];
>>H=sorddPD(A,3)
```

运行结果如下：

```
因为谱半径不小于1，所以超松弛迭代序列发散,谱半径mH和B的所有特征值H如下：
mH =
    3.6574
H =
  -3.6574 + 0.0000i
  -0.1248 + 1.6647i
  -0.1248 - 1.6647i
  -1.5697 + 0.0000i
```

SOR 方法收敛的快慢与松弛因子 ω 的选择有密切关系。但如何选取最佳松弛因子，即选取 $\omega=\omega^*$，使 $\rho(\mathfrak{I}_\omega)$ 达到最小，是一个尚未很好解决的问题。实际上可采用试算的方法来确定较好的松弛因子。经验上可取 $1.4<\omega<1.6$。

5.2　非线性方程求解

非线性方程（组）也是科学计算中最经常遇到的一个问题。线性方程组的求解相对较为容易，而非线性方程组通常很难求其解析解，更多的是通过数值分析求其近似解。本节介绍几种用于求解非线性方程（组）的方法以及 MATLAB 实现。

5.2.1　非线性方程数值求解基本原理

与线性方程相比，非线性方程问题无论是从理论上还是从计算公式上，都要复杂得多。对于一般的非线性方程 $f(x)=0$，计算方程的根既无章可循也无直接法可言。例如，求解高

次方程组 $5x^5-7x^4+3x^2-x+3=0$ 的根，求解含有指数和正弦函数的超越方程 $\mathrm{e}^x-\cos(\pi x)=0$ 的零点。

在解方程方面，牛顿提出了方程求根的一种迭代方法，这种方法被后人称为牛顿算法。牛顿算法可以说是数值计算方面最有影响的计算方法。对于方程式 $f(x)=0$，如果 $f(x)$ 是线性函数，则它的求根是容易的。牛顿法实质上是一种线性化方法，其基本思想是将非线性方程式 $f(x)$ 逐步归结为某种线性方程来求解。解非线性方程组只是非线性方程的一种延伸和扩展：

$$\begin{cases} f_1(x_1,\cdots,x_n)=0 \\ \quad\vdots \\ f_n(x_1,\cdots,x_n)=0 \end{cases} \tag{5.24}$$

式中 $f_1,\cdots,f_n$ 均为 $x_1,\cdots,x_n$ 多元函数。若用向量记号记 $\boldsymbol{x}=(x_1,\cdots,x_n)^{\mathrm{T}}\in R^n$，$\boldsymbol{F}=(f_1,\cdots,f_n)^{\mathrm{T}}$，式（5.24）就可以写成

$$\boldsymbol{F}(x)=0 \tag{5.25}$$

当 $n\geq 2$ 且 $f_1,\cdots,f_n$ 中至少有一个是自变量 $x_1,\cdots,x_n$ 的非线性函数时，方程组（5.25）为非线性方程组。非线性方程组求根问题是前面介绍的方程求根的直接推广，实际上只要把单变量函数 $f(x)$视为向量函数 $F(\boldsymbol{x})$，就可将单变量方程求根方法推广到方程组（5.25）。若已给出方程（5.25）的一个近似根 $\boldsymbol{x}^{(k)}=(x_1^k,\cdots,x_n^k)^{\mathrm{T}}$，将函数 $F(\boldsymbol{x})$ 的分量 $f_i(\boldsymbol{x}),i=1,\cdots,n$ 在 $\boldsymbol{x}^{(k)}$ 用多元函数泰勒展开，并取其线性部分，则可表示为

$$F(\boldsymbol{x})\approx F(\boldsymbol{x}^{(k)})+F'(\boldsymbol{x}^{(k)})(\boldsymbol{x}-\boldsymbol{x}^{(k)}) \tag{5.26}$$

令上式左端为零，得到线性方程组

$$F'(\boldsymbol{x}^{(k)})(\boldsymbol{x}-\boldsymbol{x}^{(k)})\approx -F(\boldsymbol{x}^{(k)}) \tag{5.27}$$

式中，

$$F'(\boldsymbol{x})=\begin{bmatrix} \dfrac{\partial f_1(x)}{\partial x_1} & \dfrac{\partial f_1(x)}{\partial x_2} & \cdots & \dfrac{\partial f_1(x)}{\partial x_n} \\ \dfrac{\partial f_2(x)}{\partial x_1} & \dfrac{\partial f_2(x)}{\partial x_2} & \cdots & \dfrac{\partial f_2(x)}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial f_n(x)}{\partial x_1} & \dfrac{\partial f_n(x)}{\partial x_2} & \cdots & \dfrac{\partial f_n(x)}{\partial x_n} \end{bmatrix} \tag{5.28}$$

称 $F'(\boldsymbol{x})$ 为 $F(\boldsymbol{x})$ 的雅可比矩阵，求解线性方程组（5.27），并记解为 $\boldsymbol{x}^{(k+1)}$，可得

$$\boldsymbol{x}^{(k+1)}=\boldsymbol{x}^{(k)}-F'(\boldsymbol{x}^{(k)})^{-1}F(\boldsymbol{x}^{(k)}),\quad k=0,1,\cdots \tag{5.29}$$

这就是解非线性方程组（5.27）的基本思想，也即牛顿法思想。牛顿法的主要思想是用 $\boldsymbol{x}^{(k+1)}=\boldsymbol{x}^{(k)}-F'(\boldsymbol{x}^{(k)})^{-1}F(\boldsymbol{x}^{(k)}),k=0,1,\cdots$ 进行迭代。因此先要算出 $F(\boldsymbol{x})$ 的雅可比矩阵 $F'(\boldsymbol{x})$，再求它的逆 $F'(\boldsymbol{x})^{-1}$，当它达到设定精度时即停止迭代。算法步骤如下：

（1）首先定义方程组 $F(\boldsymbol{x})$，确定步长 $\Delta\boldsymbol{x}$ 和精度 $\delta\boldsymbol{x}$。

（2）求 $F(\boldsymbol{x})$ 的雅可比矩阵 $F'(\boldsymbol{x})$，可用

$$\frac{\partial f_i(x_1,\cdots,x_j,\cdots,x_n)}{\partial x_j}=\frac{f_i(x_1,\cdots,x_j+\Delta x,\cdots,x_n)-f_i(x_1,\cdots,x_j,\cdots,x_n)}{\Delta x}$$

求出雅可比矩阵。

（3）求雅可比矩阵 $F'(\boldsymbol{x})$ 的逆 $F'(\boldsymbol{x})^{-1}$。

（4）用式（5.27）进行迭代运算。

（5）当精度达到设定的阈值 $\delta\boldsymbol{x}$ 时停止迭代，否则重复步骤（2）～（4）。

5.2.2 非线性方程求根的 MATLAB 命令

1．solve 命令求解非线性方程

求非线性方程 $f(x)=0$ 的根可以用 MATLAB 命令 `solve` 进行求解，调用格式如下：

```
x=solve('f(x)=0','x');
```

求非线性方程组 $\begin{cases} f_1(x_1,\cdots,x_n)=0 \\ \quad\vdots \\ f_n(x_1,\cdots,x_n)=0 \end{cases}$ 的根可以用以下 MATLAB 命令：

```
E1=sym('f1(x1,…,xn)=0')
      ⋮
En=sym('fn(x1,…,xn)=0')
```

【例 5.10】解非线性方程 $2x^5+17x^4-x^3-157x^2-x+140=0$。

根据 `solve` 命令的调用格式，在 MATLAB 工作窗口输入程序：

```
>> x=solve('2*x^5+17*x^4-x^3-157*x^2-x+140=0')
```

运行结果如下：

```
x =
   1
  -4
  -1
  -7
 5/2
```

【例 5.11】解非线性方程组 $\begin{cases} x^2y-x^2+xy-x-6y+6=0 \\ -x^2y+y^2-3xy-2x+3y+2=0 \end{cases}$。

根据 `solve` 命令解方程组的调用格式，在 MATLAB 工作窗口输入程序：

```
>> E1=sym('x^2*y-x^2+x*y-x-6*y+6=0');
>> E2=sym('-x*y^2+y^2-3*x*y-2*x+3*y+2=0');
>> [x,y]=solve(E1,E2)
```

运行结果如下：

```
x =
  1
  2
  2
 -3
 -3
y =
  1
 -2
 -1
 -1
 -2
```

2. 多项式方程求解

如果 $f(x)$ 为多项式，则可分别用如下命令求方程 $f(x)=0$ 的根，或求导数 $f'(x)$，其调用格式见表 5.1。

表 5.1　roots 命令格式

命　令	功　能
xk=roots(fa)	输入多项式 $f(x)$ 的系数 fa（按降幂排列），运行后输出 xk 为 $f(x)=0$ 的全部根
dfa=polyder(fa)	输入多项式 $f(x)$ 的系数 fa（按降幂排列），运行后输出 dfa 为多项式 $f(x)$ 的导数 $f'(x)$ 的系数
dfx=poly2sym(dfa)	输入多项式 $f(x)$ 的导数 $f'(x)$ 的系数 dfa（按降幂排列），运行后输出 dfx 为多项式 $f(x)$ 的导 $f'(x)$

【例 5.12】解多项式方程 $2x^5+17x^4-x^3-157x^2-x+144=0$，并求其导数。

（1）求方程的根。在 MATLAB 工作窗口输入程序：

```
>> fa=[2,17,-1,-157,-1,140];
>> xk=roots(fa)
```

运行结果如下：

```
xk =
   -7.0000
   -4.0000
    2.5000
   -1.0000
1.0000
```

（2）求方程的导数。以多项式系数方式显示，在 MATLAB 工作窗口输入程序：

```
>> fa=[2,17,-1,-157,-1,140];
>> dfa=polyder(fa)
```

运行结果如下：

```
dfa =
    10    68    -3  -314    -1
```

以多项式符号表达式形式显示，再在 MATLAB 工作窗口输入程序：

```
>> dfx=poly2sym(dfa)
```

运行结果如下：

```
dfx =
10*x^4 + 68*x^3 - 3*x^2 - 314*x - 1
```

这两类求方程根的方法都有缺点，roots(fa)命令只能求 $f(x)$为多项式时方程 $f(x)=0$ 的根，而 solve 命令不能求出周期函 x 对应的方程 $f(x)=0$ 的全部根。下面介绍求方程根的近似值方法。

3. fsolve 命令求解非线性方程

如果非线性方程（组）是多项式形式，那么求这种方程（组）的数值解可以直接调用上面已经介绍过的 roots 命令。如果非线性方程（组）含有超越函数，则无法使用 roots 命令，需要调用 MATLAB 系统中提供的另一个程序 fsolve 来求解。同时 fsolve 命令也可以用于多项式方程（组），但它的计算量明显比 roots 命令的大。

fsolve 命令使用最小二乘法解非线性方程（组）

$$F(\boldsymbol{X})=\boldsymbol{0} \tag{5.30}$$

的数值解，其中 $\boldsymbol{X}$ 和 $F(\boldsymbol{X})$可以是向量或矩阵。此种方法需要输入方程解 $\boldsymbol{X}$ 的初始值（向量或

矩阵 $\boldsymbol{X}_0$），即使程序中的迭代序列收敛，也不一定收敛到 $F(\boldsymbol{X})=\boldsymbol{0}$ 的根。fsolve 命令的调用格式如下：

```
X=fsolve(F,X0)
```

其中 F 表示非线性方程组函数文件，X0 表示 X 的初始值。

【例 5.13】使用 fsolve 命令求方程 $2x^5+17x^4-x^3-157x^2-x+144=0$ 的实根。

（1）建立 MATLAB 函数文件并保存为 Fun1.m，函数文件代码如下：

```
function F=Fun1(x)
        F=2*x^5+17*x^4-x^3-157*x^2-x+140;
```

（2）假设取初始值 X0=0，在 MATLAB 工作窗口输入命令：

```
>> X0=0.5;
>> X=fsolve('Fun1',X0),F=Fun1(X)
```

运行后结果如下：

```
X =
   1.000000000000001
F =
    -3.126388037344441e-13
```

通过上述运行结果可以看出 fsolve 命令可求解给定初始值附近一个解的近似值。

【例 5.14】求方程组 $\begin{cases}x^3-y^2=0\\ \mathrm{e}^{-x}-y=0\end{cases}$ 的一个实根。

（1）建立 MATLAB 函数文件并保存为 Fun2.m，函数文件代码如下：

```
function F=Fun2(X)
        x=X(1);y=X(2);F(1)=x^3-y^2;
        F(2)=exp(-x)-y;
```

（2）去初始值向量 X0=(1,1)，在 MATLAB 工作窗口输入命令：

```
>> X0=[1,1];
>> X=fsolve('Fun2',X0),F=Fun2(X)
```

运行结果如下：

```
X =
   0.648844236109663   0.522649455145584
F =
   1.0e-06 *
   0.219551996138989   0.032079490175363
```

【例 5.15】求方程 $\sin(2x+1)=0$ 的两个实根。

取两个实根的初始值为 X0=(-1,1)，在 MATLAB 工作窗口输入命令：

```
>> fun=inline('sin(2*x+1)');
>> X=fsolve(fun,[-0.9 0.9],optimset('Display','off')),F=fun(X)
```

运算结果如下：

```
X =
  -0.499999988313669   1.070796326794895
F =
   1.0e-07 *
   0.233726610288087   0.000000027869999
```

5.2.3　非线性方程数值解法及 MATLAB 实现

非线性方程（组）的数值解法有很多，包括逐步搜索法、二分法、牛顿法、割线法等，本节只介绍逐步搜索法和二分法。

1. 逐步搜索法

逐步搜索法也称试算法，它是求方程 $f(x)=0$ 的根的近似值位置的一种常用方法。逐步搜索法依赖于寻找连续函数 $f(x)$ 满足 $f(a)$ 与 $f(b)$ 异号的区间$[a, b]$。一旦找到区间，无论区间多大，通过某种方法总会找到一个根。

逐步搜索法确定方程 $f(x)=0$ 的根 x^*的范围的步骤如下：

步骤 1：取含 $f(x)=0$ 的根的区间$[a, b]$，即 $f(a)*f(b)<0$。

步骤 2：从 a 开始，按某个预定的步长 h（如取 $h=(b-a)/n$，n 为正整数），不断地向右跨步，每跨一步进行一次搜索，即检查节点 $x_k=a+kh$ 上的函数 $f(x_k)$ 值的符号，若 $f(x_{k-1})*f(x_k)<0$，则可以确定一个有根区间$[x_{k-1}, x_k]$。

步骤 3：继续向右搜索，直到找出$[a, b]$上的全部有根区间$[x_{k-1}, x_k]$, $k=1,2,\cdots,n$。

步长 h 的选取要根据函数 $f(x)$ 的性质来确定。一般来说，对振荡剧烈、零点密集的函数，h 要选得小一些，但这要计算较多的函数值。

【例 5.16】用逐次搜索法确定方程 $x^3-3x^2-x+3=0$ 根的范围。

（1）首先用 MATLAB 程序选取有根区间[−6, 6]，取 h = 2，输入程序：

```
>> x=-6:2:6,y=x.^3-3*x.^2-x+3
```

输出结果如下：

```
x =
    -6    -4    -2     0     2     4     6
y =
  -315  -105   -15     3    -3    15   105
```

从上面的数据可以看出 $f(-2)*f(0)<0, f(0)*f(2)<0, f(2)*f(4)<0$，所以该三次多项式方程的三个根分别在三个区间(−2, 0)、(0, 2)和(2, 4)之间。

（2）在(-2, 4)内取 h=0.3，进一步进行搜索，在 MATLAB 工作窗口输入命令：

```
>> x=-2:0.3:4,y=x.^3-3*x.^2-x+3
```

输出结果如下：

```
x =
  Columns 1 through 11
   -2.0000   -1.7000   -1.4000   -1.1000   -0.8000   -0.5000   -0.2000
    0.1000    0.4000    0.7000    1.0000
  Columns 12 through 21
    1.3000    1.6000    1.9000    2.2000    2.5000    2.8000    3.1000
    3.4000    3.7000    4.0000
y =
  Columns 1 through 11
  -15.0000   -8.8830   -4.2240   -0.8610    1.3680    2.6250    3.0720
    2.8710    2.1840    1.1730         0
  Columns 12 through 21
   -1.1730   -2.1840   -2.8710   -3.0720   -2.6250   -1.3680    0.8610
    4.2240    8.8830   15.0000
```

从上述运算结果可以看出，1.0 是方程组的一个根，另外两个根分布在$[-1.1, -0.8]$和$[2.8, 3.1]$区间，根的分布区间相比 $h=2$ 时有所缩小，如果继续缩小步长，可进一步缩小根的分布区间，直至等于或接近方程的精确解。

从例 5.16 可见，在具体运用上述方法时，步长 h 的选取是关键的。很明显，只要步长 h 取得足够小，利用这种方法可以得到具有任意精度的近似根。不过当 h 缩小时，所要搜索的步数相应增多，从而使计算量增大。因此，我们有必要研究一种机械化算法。

收敛判定准则：设方程 $f(x)=0$ 中的函数 $y=f(x)$ 在闭区间$[a, b]$上连续，开区间(x_{k-1}, x_k)和(x_k, x_{k+1})都是$[a, b]$的子区间。

（1）若 $f(x_{k-1})*f(x_k)<0$，那么该方程在(x_{k-1}, x_k)内至少有一个根。

（2）若 $f(x_{k-1})*f(x_k)\geq 0$，而$|f(x_k)|<\varepsilon$（其中ε是给定的精度）且

$(f(x_{k+1})-f(x_k))*(f(x_k)-f(x_{k-1}))<0$，那么 x_k 是这个方程的近似解。

根据逐步搜索法的计算步骤和其收敛判定准则，编写算法代码如下：

```
function r=StepSearch(a,b,h,tol)
% 输入的量——a和b是闭区间[a,b]的左、右端点；
%h是步长；
%tol是预先给定的精度.
%是方程在[a,b]上的实根的近似值，其精度是tol；
X=a:h:b;n=(b-a)/h+1;m=0;
Y=funs(X);%funs函数根据实际带求解方程组确定
X(n+1)=X(n);Y(n+1)=Y(n);
k=2;
while(k<=n)
    sk=Y(k)*Y(k-1);
     if sk<=0
        m=m+1;
        if(abs(Y(k))<abs(Y(k-1)))  %取更接近精确解的一个解
            r(m)=X(k);
            k=k+1;
        else
            r(m)=X(k-1);
        end
     else
         xielv=(Y(k+1)-Y(k))*(Y(k)-Y(k-1));
        if (abs(Y(k))<tol)&( xielv<=0)
            m=m+1;r(m)=X(k);
     end
  end
    k=k+1;
end
```

【例 5.17】使用逐步搜索法求方程 $2x^5+17x^4-x^3-157x^2-x+144=0$ 的实根。

（1）建立方程函数文件 funs.m，代码如下：

```
function y=funs(x)
   y=2*x.^5+17*x.^4-x.^3-157*x.^2-x+140;
```

（2）取步长 h=0.001，精度 tol=0.00001，在 MATLAB 工作窗口输入命令：

```
>>X=StepSearch(-10,10,0.001,0.00001)
```

运行结果如下：

```
X =
          -7.0000   -4.0000   -1.0000    1.0000    2.5000
```

逐步搜索法原理比较简单，按照设定步长逐个计算函数 $y=f(x)$ 的值，取最接近 $f(x)=0$ 的 x 值作为方程的近似解。但逐步搜索法难以确定方程根的所在区间和步长，区间设置不当很容易漏掉一些根，区间设置过大会导致计算量增大，步长设置过大会导致解不够精确，步长设置过小会导致计算量增大。