计算方法上机作业

任课教师：张晓丹

学生：赵朝阳

学号：b20170427

班级：国科博 17

目录

3-1 3
3-2-1 10
3-3 11
3-4 13
3-5 15
3-6 17
5-1 20
5-2 30
6-1 35
6-2 38
7-2 41
9-1 47

3-1

实验目的：考察不动点迭代法的局部收敛性

试验内容：构造如下方程 $2x-e^x+3=0$ 至少采用 3 种不动点迭代法，迭代 100 次，考察收敛性，改变初值符号，再做迭代。分析收敛与发散的原因。

迭代方法一：

构造迭代方程：

$x_{k+1}=\Psi_1(x_k)$

$\Psi_1(x_k)=(e^x-3)/2$

程序截图

```
function x1 = a_1(x)
%First iterative method
i = 0;
x0 = x;
while(i < 100)
    x1 = (exp(x0)-3)/2;
    %Terminate the process and output the result when the abs between x1 and x0 is less than 1e-4
    while( abs( x1 - x0 ) < 1e-4)
        fprintf('在初始值为%i的条件下，在第%i次迭代后，结果为%i，误差小于0.0001\r\n',[x,i,x1]);
        break;
    end
    x0 = x1;
    i = i + 1;
end
```

运行结果截图

当初始值 $x_0=1$

```
>> c = a_1(1)
在第次6迭代后，结果为-1.373363e+00，误差小于0.0001

在第次7迭代后，结果为-1.373373e+00，误差小于0.0001

在第次8迭代后，结果为-1.373374e+00，误差小于0.0001

在第次9迭代后，结果为-1.373375e+00，误差小于0.0001
```

```
…

在初始值为1的条件下，在第97次迭代后，结果为-1.373375e+00，误差小于0.0001

在初始值为1的条件下，在第98次迭代后，结果为-1.373375e+00，误差小于0.0001

在初始值为1的条件下，在第99次迭代后，结果为-1.373375e+00，误差小于0.0001


c =

   -1.3734
```

迭代 9 次后，结果收敛。

当 x_0=-1

```
>> c = a_1(-1)
在初始值为-1的条件下，在第5次迭代后，结果为-1.373373e+00，误差小于0.0001

在初始值为-1的条件下，在第6次迭代后，结果为-1.373374e+00，误差小于0.0001

在初始值为-1的条件下，在第7次迭代后，结果为-1.373375e+00，误差小于0.0001

…

在初始值为-1的条件下，在第97次迭代后，结果为-1.373375e+00，误差小于0.0001

在初始值为-1的条件下，在第98次迭代后，结果为-1.373375e+00，误差小于0.0001

在初始值为-1的条件下，在第99次迭代后，结果为-1.373375e+00，误差小于0.0001


c =

   -1.3734
```

迭代 7 次后收敛。

迭代方法二：

构造迭代方程：

$x_{k+1}=\Psi_2(x_k)$

$\Psi_2(x_k)=\ln(2*x_k+3)$

程序截图：

```
function x1 = a_2(x)
%Second iterative method
i = 0;
x0 = x;
while(i < 100)
    x1 = log(2*x0+3);
    x0 = x1;
    i = i + 1;
    %Terminate the process and output the result when the abs between x1 and x0 is less than 1e-4
    while( abs( x1 - x0 ) < 1e-4)
        fprintf('在初始值为%i的条件下，在第%i次迭代后，结果为%i，误差小于0.0001\r\n',[x,i,x1]);
        break;
    end
end
```

程序运行结果：

当初始值 $x_0=1$

命令行窗口

```
>> res = a_2(1)
在初始值为1的条件下，在第1次迭代后，结果为1.609438e+00，误差小于0.0001

在初始值为1的条件下，在第2次迭代后，结果为1.827589e+00，误差小于0.0001

在初始值为1的条件下，在第3次迭代后，结果为1.895395e+00，误差小于0.0001

在初始值为1的条件下，在第4次迭代后，结果为1.915567e+00，误差小于0.0001

在初始值为1的条件下，在第5次迭代后，结果为1.921491e+00，误差小于0.0001

在初始值为1的条件下，在第6次迭代后，结果为1.923224e+00，误差小于0.0001

在初始值为1的条件下，在第7次迭代后，结果为1.923730e+00，误差小于0.0001

在初始值为1的条件下，在第8次迭代后，结果为1.923878e+00，误差小于0.0001

在初始值为1的条件下，在第9次迭代后，结果为1.923921e+00，误差小于0.0001

在初始值为1的条件下，在第10次迭代后，结果为1.923934e+00，误差小于0.0001

在初始值为1的条件下，在第11次迭代后，结果为1.923937e+00，误差小于0.0001

在初始值为1的条件下，在第12次迭代后，结果为1.923938e+00，误差小于0.0001

在初始值为1的条件下，在第13次迭代后，结果为1.923939e+00，误差小于0.0001
```

...

```
在初始值为1的条件下，在第98次迭代后，结果为1.923939e+00，误差小于0.0001

在初始值为1的条件下，在第99次迭代后，结果为1.923939e+00，误差小于0.0001

在初始值为1的条件下，在第100次迭代后，结果为1.923939e+00，误差小于0.0001


res =

    1.9239
```

迭代 13 次后收敛。

当初始值 $x_0=-1$

命令行窗口

```
>> res = a_2(-1)
在初始值为-1的条件下，在第1次迭代后，结果为0，误差小于0.0001

在初始值为-1的条件下，在第2次迭代后，结果为1.098612e+00，误差小于0.0001

在初始值为-1的条件下，在第3次迭代后，结果为1.648125e+00，误差小于0.0001

在初始值为-1的条件下，在第4次迭代后，结果为1.839954e+00，误差小于0.0001

在初始值为-1的条件下，在第5次迭代后，结果为1.899104e+00，误差小于0.0001

在初始值为-1的条件下，在第6次迭代后，结果为1.916659e+00，误差小于0.0001

在初始值为-1的条件下，在第7次迭代后，结果为1.921810e+00，误差小于0.0001

在初始值为-1的条件下，在第8次迭代后，结果为1.923317e+00，误差小于0.0001

在初始值为-1的条件下，在第9次迭代后，结果为1.923757e+00，误差小于0.0001

在初始值为-1的条件下，在第10次迭代后，结果为1.923886e+00，误差小于0.0001

在初始值为-1的条件下，在第11次迭代后，结果为1.923923e+00，误差小于0.0001

在初始值为-1的条件下，在第12次迭代后，结果为1.923934e+00，误差小于0.0001

在初始值为-1的条件下，在第13次迭代后，结果为1.923937e+00，误差小于0.0001

在初始值为-1的条件下，在第14次迭代后，结果为1.923938e+00，误差小于0.0001

在初始值为-1的条件下，在第15次迭代后，结果为1.923939e+00，误差小于0.0001

...

在初始值为-1的条件下，在第98次迭代后，结果为1.923939e+00，误差小于0.0001

在初始值为-1的条件下，在第99次迭代后，结果为1.923939e+00，误差小于0.0001

在初始值为-1的条件下，在第100次迭代后，结果为1.923939e+00，误差小于0.0001


res =

    1.9239
```

迭代方法三：

构造迭代方程：

$$x_{k+1}=\Psi_3(x_k)$$

$$\Psi_3(x_k)=e^{x_k}-x_k-3$$

程序截图：

```
function x1 = a_3(x)
%Third iterative method
i = 0;
x0 = x;
while(i < 100)
    x1 = exp(x0) - x0 -3;
    x0 = x1;
    i = i + 1;
    %Terminate the process and output the result when the abs between x1 and x0 is less than 1e-4
    while( abs( x1 - x0 ) < 1e-4)
        fprintf('在初始值为%i的条件下，在第%i次迭代后，结果为%i，误差小于0.0001\r\n',[x,i,x1]);
        break;
    end
end
```

程序运行截图：

当初始值 $x_0=1$

命令行窗口

```
>> res = a_3(1)
在初始值为1的条件下，在第1次迭代后，结果为-1.281718e+00，误差小于0.0001

在初始值为1的条件下，在第2次迭代后，结果为-1.440722e+00，误差小于0.0001

在初始值为1的条件下，在第3次迭代后，结果为-1.322521e+00，误差小于0.0001
```

...

```
在初始值为1的条件下，在第47次迭代后，结果为-1.373374e+00，误差小于0.0001

在初始值为1的条件下，在第48次迭代后，结果为-1.373375e+00，误差小于0.0001

在初始值为1的条件下，在第49次迭代后，结果为-1.373374e+00，误差小于0.0001

在初始值为1的条件下，在第50次迭代后，结果为-1.373375e+00，误差小于0.0001
```

…

在初始值为1的条件下，在第99次迭代后，结果为-1.373375e+00，误差小于0.0001

在初始值为1的条件下，在第100次迭代后，结果为-1.373375e+00，误差小于0.0001

```
res =

   -1.3734
```

迭代 50 次后收敛

当初始值 x_0=-1

命令行窗口

```
>> res = a_3(-1)
在初始值为-1的条件下，在第1次迭代后，结果为-1.632121e+00，误差小于0.0001

在初始值为-1的条件下，在第2次迭代后，结果为-1.172365e+00，误差小于0.0001

在初始值为-1的条件下，在第3次迭代后，结果为-1.518001e+00，误差小于0.0001
```

…

在初始值为-1的条件下，在第51次迭代后，结果为-1.373375e+00，误差小于0.0001

在初始值为-1的条件下，在第52次迭代后，结果为-1.373374e+00，误差小于0.0001

在初始值为-1的条件下，在第53次迭代后，结果为-1.373375e+00，误差小于0.0001

在初始值为-1的条件下，在第54次迭代后，结果为-1.373374e+00，误差小于0.0001

在初始值为-1的条件下，在第55次迭代后，结果为-1.373375e+00，误差小于0.0001

…

在初始值为-1的条件下，在第99次迭代后，结果为-1.373375e+00，误差小于0.0001

在初始值为-1的条件下，在第100次迭代后，结果为-1.373375e+00，误差小于0.0001

```
res =

   -1.3734
```

迭代 55 次后收敛

分析收敛与发散的原因：三种方法收敛依次加快，主要是其误差余项依次减小。

3-2-1

（1）实验目的：考察 Newton 法求单根的收敛速度

实验内容：应用 Newton 法求解实验 3-1 中的方程，并与实验 3-1 中收敛的迭代法进行比较，考察收敛速度。精确到 10^{-4}。

按照 Newton 法得如下推导：

$f(x)=e^x-2x-3$

$f'(x)=e^x-2$

$u(x)=f(x)/f'(x)$

$\Phi(x)=x-u(x)=x-\frac{e^x-2x-3}{e^x-2}$

迭代公式为：

$X_{k+1}=\Phi(x_k)$

程序截图：

```
1    function x1 = b(x)
2    %This slice of codes responds to the 3-2 problem.
3 -  x0 = x;
4 -  a = 1;
5 -  i = 0;
6 -  while(a > 1e-4)
7 -      x1 = x0 - (exp(x0)-2*x0-3)/(exp(x0)-2);
8 -      a = abs(x1 - x0);
9 -      x0 = x1;
10 -     i = i + 1;
11 -   end
12 -   fprintf('在初始值为%i的条件下，在第%i次迭代后，结果为%i，误差小于0.0001\r\n',[x,i,x1]);
```

程序运行截图：

当初始值为 $x_0=1$

```
>> res = b(1)
在初始值为1的条件下，在第8次迭代后，结果为1.923939e+00，误差小于0.0001


res =

    1.9239
```

迭代 8 次后收敛

当初始值为 $x_0=-1$

命令行窗口

```
>> res = b(-1)
在初始值为-1的条件下，在第3次迭代后，结果为-1.373375e+00，误差小于0.0001


res =

   -1.3734
```

迭代 3 次后收敛。

收敛速度明显快于 3-1 中的不动点迭代法

3-3

实验目的：掌握求重根的方法

实验内容：分别用 Newton 法与不动点迭代法求解方程 x-sinx=0 考察收敛速度，再用求重根的两种方法求方程的根，精确到 10^{-4}。

不动点迭代法求解方程：

构造迭代方程：

$x_{k+1}=\Psi(x_k)$

$\Psi(x_k)=\sin x_k$

程序截图：

```
function x1 = c_n(x)
%This slice of codes responds to the 3-3 problem.
%Newton method
x0 = x;
a = 1;
i = 0;
while(a > 1e-4)
    x1 = sin(x0);
    a = abs(x1 - x0);
    x0 = x1;
    i = i + 1;
end
fprintf('在初始值为%i的条件下，在第%i次迭代后，结果为%f，误差小于0.0001\r\n',[x,i,x1]);
```

程序运行截图：

当初始值为 $x_0=1$

```
命令行窗口
>> res = c_n(1)
在初始值为1的条件下，在第417次迭代后，结果为0.084209，误差小于0.0001


res =

    0.0842
```

迭代 417 次后达到误差精度

Newton 法求解方程

构造迭代方程：

$f(x)=x-\sin x$

$f'(x)=1-\cos(x)$

$u(x)=f(x)/f'(x)$

$\Phi(x)=x-u(x)=x-\frac{x-\sin x}{1-\cos x}$

迭代公式为：

$x_{k+1}=\Phi(x_k)$

程序截图：

```
function x1 = c_s(x)
%This slice of codes responds to the 3-3 problem.
%Newton method
x0 = x;
i = 0;
a = 1;
while(a > 1e-4)
    x1 = x0 - (x0-sin(x0))/(1-cos(x0));
    a = abs(x1 - x0);
    x0 = x1;
    i = i + 1;
end
fprintf('在初始值为%i的条件下，在第%i次迭代后，结果为%f，误差小于0.0001\r\n',[x,i,x1]);
```

程序运行截图：

当初始值为 $x_0=1$

```
命令行窗口
>> res = c_s(1)
在初始值为1的条件下，在第21次迭代后，结果为0.000194，误差小于0.0001


res =

   1.9449e-04
```

迭代 21 次后达到误差精度

经比较，Newton 法收敛更快。

3-4

实验目的：体验 Steffensen's method 加速技巧

实验内容：先用 Newton 法求解方程 x-tanx=0

再用 Steffensen's method 求解，比较迭代步数。精确到 10^{-4}。

Newton 法：

f(x)=x-tanx

$f'(x)=1-1/\cos^2 x$

$u(x)=f(x)/f'(x)$

$\Phi(x)=x-u(x)=x-\frac{x\cos^2(x)-\sin(x)\cos(x)}{\sin^2(x)}$

迭代公式：

$x_{k+1}=\Phi(x_k)$

程序截图：

```
1   function x1 = d_n(x)
2   %This slice of codes responds to the 3-4 problem.
3   %Newton method
4   x0 = x;
5   i = 0;
6   a = 1;
7   while(a > 1e-4)
8       x1 = x0 + (x0*cos(x0)^2-sin(x0)*cos(x0))/((sin(x0))^2);
9       a = abs(x1 - x0);
10      i = i + 1;
11      x0 = x1;
12  end
13  fprintf('根据Newton迭代法，在初始值为%i的条件下：\r\n\t在第%i次迭代后，结果为%f，误差小于0.0001\r\n',[x,i,x1]);
```

程序运行截图：

```
命令行窗口
>> res = d_n(1)
根据Newton迭代法，在初始值为1的条件下：

    在第22次迭代后，结果为0.000182，误差小于0.0001


res =

   1.8168e-04
```

Steffensen's method：

构造不动点迭代方程

$\Psi(x)=\tan(x)$

$x_{k+1}=\Psi(x_k)$

程序截图：

```
function p1 = d_s(x)
%This slice of codes responds to the 3-4 problem.
%Steffensen method
n = 0;
p(1) = x;
while(n <= 1000)
    %get p(1),p(2),p(3)
    for k = 1 : 2
        p(k+1) = tan(p(k));
    end
    %The acceleration of Atiken
    p1 = p(1) - ( p(2) - p(1) )^2/(p(3) - 2*p(2) + p(1));
    f0 = p1 - tan(p1);
    if( abs(f0) < 1e-4 )
        break;
    end
    n = n + 1;
    p(1) = p1;
end
fprintf('根据Atiken加速，在初始值为%i的条件下：\r\n\t在第%i次迭代后，结果为%f，误差小于0.0001\r\n',[x,n,p1]);
```

程序运行截图：

```
命令行窗口
>> res = d_s(1)
根据Atiken加速，在初始值为1的条件下：

    在第13次迭代后，结果为0.047783，误差小于0.0001


res =

    0.0478
```

3-5

分别用不动点迭代与 Newton 法求解方程 $3x-5^x+4=0$ 的正根与负根。

不动点迭代法：

构造不动点方程：

$\Psi(x)=(5^x-4)/3$

$x_{k+1}=\Psi(x_k)$

程序截图：

```
function res = e_s(a)
x(1) = a;
i = 0;
err = 1;
while(err > 1e-4)
    x(2) = (5^(x(1))-4)/3;
    % for test: fprintf('x(2)=%f\r\n',x(2));
    f = 3 * x(2) - 5^(x(2)) + 4;
    % for test: fprintf('f=%f\r\n',f);
    err = abs(f);
    i = i + 1;
    x(1) = x(2);
end
res = x(2);
fprintf('根据不动点迭代法，在初始值为%i的条件下：\r\n\t在第%i次迭代后，结果为%f，误差小于0.0001\r\n',[a,i,res]);
```

程序运行截图：

命令行窗口

```
>> res = e_s(1)
根据不动点迭代法，在初始值为1的条件下：

    在第6次迭代后，结果为-1.291623，误差小于0.0001


res =

   -1.2916
```

程序计算负根时，求得根为-1.2916

程序计算正根时，程序发散

Newton：

$f(x)=3x-5^x+4$

$f'(x)=3-\ln(5)*5^x$

$u(x)=f(x)/f'(x)$

$\Phi(x)=x-u(x)=\frac{5^x(1-x\,In5)-4}{3-In5*5^x}$

$x_{k+1}=\Phi(x_k)$

程序截图：

```
function res = e_n(c)
%This slice of codes responds to the 3-5 problem.
%Newton method
x(1) = c;
i = 0;
a = 1;
while(a > 1e-4)
    x(2) = ((1-x(1)*log(5))*5^(x(1)) - 4)/(3-log(5)*5^(x(1)));
    f = 3*x(2) - 5^(x(2)) + 4;
    a = abs(f);
    i = i + 1;
    x(1) = x(2);
end
res = x(2);
fprintf('根据不动点迭代法，在初始值为%i的条件下：\r\n\t在第%i次迭代后，结果为%f，误差小于0.0001\r\n',[c,i,res]);
```

程序运行截图：

命令行窗口

```
>> res = e_n(2)
根据不动点迭代法，在初始值为2的条件下：

    在第5次迭代后，结果为1.279314，误差小于0.0001


res =

    1.2793

>> res = e_n(-2)
根据不动点迭代法，在初始值为-2的条件下：

    在第2次迭代后，结果为-1.291651，误差小于0.0001


res =

   -1.2917
```

程序计算负根时，求得根为-1.2917

程序计算正根时，求得根为 1.2793

3-6

用 Newton 法与重根计算法求解方程 x-sinx = 0 的根。再用

Steffensen's method 加速 Newton 法收敛，比较结果。

Newton:

f(x)=x-sin(x)

f'(x)=1-cos(x)

u(x)=f(x)/f'(x)

$\Phi(x)=x-u(x)=x-\frac{x-\sin(x)}{1-\cos(x)}$

$x_{k+1}=\Phi(x_k)$

程序截图：

```
1   function res = f_n(q)
2   %This slice of codes responds to the 3-6 problem.
3   %Newton method
4   x(1) = q;
5   a = 1;
6   i = 0;
7   while(a > 1e-4)
8       x(2) = x(1) - (x(1) - sin(x(1)))/(1 - cos(x(1)));
9       f = x(2) - sin(x(2));
10      a = abs(f);
11      i = i + 1;
12      x(1) = x(2);
13  end
14  res = x(2);
15  fprintf('根据不动点迭代法，在初始值为%i的条件下：\r\n\t在第%i次迭代后，结果为%f，误差小于0.0001\r\n',[q,i,res]);
```

程序运行截图：

命令行窗口

```
>> res = f_n(2)
根据不动点迭代法，在初始值为2的条件下：

    在第8次迭代后，结果为0.068734，误差小于0.0001


res =

    0.0687
```

重根计算法:

f''(x)=sin(x)

f'''(x)=cos(x)

f(0)=f'(0)=f''(0)=0　　f'''(0)=1

x=0 是方程的三重根。故 m=3

构造迭代方程：

Φ(x)=x-m*u(x)= $x - 3 * \frac{x - \sin(x)}{1 - \cos(x)}$

程序截图：

```
function res = f_r(q)
%This slice of codes responds to the 3-6 problem.
%Double root iterative method
x(1) = q;
a = 1;
i = 0;
while(a > 1e-4)
    x(2) = x(1) - 3*(x(1) - sin(x(1)))/(1 - cos(x(1)));
    f = x(2) - sin(x(2));
    a = abs(f);
    i = i + 1;
    x(1) = x(2);
end
res = x(2);
fprintf('根据不动点迭代法，在初始值为%i的条件下：\r\n\t在第%i次迭代后，结果为%f，误差小于0.0001\r\n',[q,i,res]);
```

程序运行截图：

```
>> res = f_r(2)
根据不动点迭代法，在初始值为2的条件下：

    在第2次迭代后，结果为0.001002，误差小于0.0001


res =

    0.0010
```

Steffensen's method 加速 Newton 法：

程序截图：

```
function res = f_s(q)
%This slice of codes responds to the 3-6 problem.
%Steffensen method
x(1) = q;
i = 0;
a = 1;
while(a > 1e-4)
    for k = 1 : 2
        x(k+1) = x(k) - (x(k) - sin(x(k)))/(1 - cos(x(k)));
    end
    %The acceleration of Atiken
    p = x(1) - (x(2) - x(1))^2/(x(3) - 2*x(2) + x(1));
    f = p - sin(p);
    a = abs(f);

    i = i + 1;
    x(1) = p;
end
res = p;
fprintf('根据Steffensen加速Newton，在初始值为%i的条件下：\r\n\t在第%i次迭代后，结果为%f，误差小于0.0001\r\n',[q,i,p]);
```

程序运行截图：

```
>> res = f_s(2)
根据Steffensen加速Newton，在初始值为2的条件下：

    在第2次迭代后，结果为0.000558，误差小于0.0001


res =

   5.5753e-04
```

结果比较：

	迭代次数	迭代结果
Newton	8	0.0687
不动点迭代	2	0.001002
Newton+Steffensen	2	0.000558

三种方法的迭代次数逐渐减小，精度逐渐提高。

5-1

实验目的：熟悉 Jacobi、Seidel、Sor 迭代法，了解松弛因子对收敛

速度的影响。

实验内容：分别用 Jacobi、Seidel、Sor(w=0.8,1.1,1.2,1.3,1.4,1.5)迭代法求解下面的方程组，并做结果分析。

初值 $x^{(0)}=(0，0，0，0，0)^T$，精度要求： $\frac{\left\|X^{(k+1)}-X^{(k)}\right\|_\infty}{\left\|X^{(k+1)}\right\|_\infty}<10^{-5}$ **。**

$$(1)\begin{cases}12.3x_1-2x_2-x_3+3.4x_4-3.7x_5=4.8\\1.4x_1+9x_2-3x_3+2.4x_4+2.7x_5=2.3\\2.1x_1+x_2+8x_3+2.6x_4+5.8x_5=2.5\\3.5x_1-2.1x_2+x_3+13x_4+4.6x_5=3.6\\2.5x_1-x_2-2x_3+5.3x_4+14.8x_5=2.2\end{cases}$$

$$(2)\begin{cases}13.3x_1-4x_2-x_3+3.5x_4-3.8x_5=5.8\\3.4x_1+9x_2-3x_3+4.4x_4+2.3x_5=4.3\\4.1x_1+x_2+7x_3+2.7x_4+5.9x_5=2.6\\2.5x_1-2.4x_2+x_3+13x_4+5.6x_5=3.8\\1.5x_1-x_2-3x_3+4.3x_4+14.9x_5=4.2\end{cases}$$

(1)

1、根据 Jacobi：

程序截图：

```
function res = a_j()
%This slice of codes responds to the 5-1 problem.
%Jacobi method
A = [12.3,-2,-1,3.4,-3.7;1.4,9,-3,2.4,2.7;2.1,1,8,2.6,5.8;3.5,-2.1,1,13,4.6;2.5,-1,-2,5.3,14.8];
b = [4.8,2.3,2.5,3.6,2.2]';
len = length(b);
D1 = zeros(len,len);
for i = 1 : len
    %At this moment, D is transfered into the inverse of D1
    D1(i,i) = 1/A(i,i);
end
B = eye(len) - D1*A;
g = D1*b;
%The iteration part;
x1 = [0,0,0,0,0]';
x2 = [0,0,0,0,0]';
dx = 1;
k = 0;
while(dx > 1e-5)
    x2 = B*x1 + g;
    dx = norm(x2-x1,inf)/norm(x2,inf);
    %fprintf('dx = %f',dx);
    k = k + 1;
    x1 = x2;
end
res = x2;
fprintf('根据Jacobi迭代：\r\n\t在第%i次迭代后的迭代结果：\r\n',[k]);
disp(res);
```

程序运行截图：

```
>> res = a_j()
根据Jacobi迭代：

    在第17次迭代后的迭代结果：

    0.3906
    0.1678
    0.0996
    0.1754
    0.0447
```

2、根据 Seidel：

程序截图：

```
function a_gs()
%This slice of codes responds to the 5-1 problem.
%Gauss-Seidel method for test
A = [12.3,-2,-1,3.4,-3.7;1.4,9,-3,2.4,2.7;2.1,1,8,2.6,5.8;3.5,-2.1,1,13,4.6;2.5,-1,-2,5.3,14.8];
b = [4.8,2.3,2.5,3.6,2.2]';
len = length(b);
D = eye(len);
D1 = eye(len);
x0 = [0,0,0,0,0]';
x1 = [0,0,0,0,0]';
for i = 1 : len
    D(i,i) = A(i,i);
    D1(i,i) = 1/A(i,i);
end
%The core part of Gauss Seidel
err = 1;
k = 0;
while(err > 1e-5)
    x1 = -1*D1*(A-D)*x0 + D1*b;
    err = norm((x1 - x0),inf)/norm(x1,inf);
    k = k + 1;
    x0 = x1;
end
fprintf('根据Gauss Seidel迭代: \r\n\t在第%i次迭代后的迭代结果: \r\n',[k]);
disp(x1);
```

程序运行截图：

命令行窗口

```
>> a_gs()
根据Gauss Seidel迭代:

    在第17次迭代后的迭代结果:

    0.3906
    0.1678
    0.0996
    0.1754
    0.0447
```

3、根据 Sor

程序截图：

```
function a_sor(w)
%w:松弛因子
%This slice of codes responds to the 5-1 (1)problem.
%Sor method
A = [12.3,-2,-1,3.4,-3.7;1.4,9,-3,2.4,2.7;2.1,1,8,2.6,5.8;3.5,-2.1,1,13,4.6;2.5,-1,-2,5.3,14.8];
b = [4.8,2.3,2.5,3.6,2.2]';
len = length(b);
D = eye(len);
for i = 1 : len
    D(i,i) = A(i,i);
end
L = tril(A - D);
U = triu(A - D);
%The main part of Sor
err = 1;
B = inv(D + w*L)*((1-w)*D - w*U);
d = w*inv(D + w*L)*b;
x0 = [0,0,0,0,0]';
x1 = [0,0,0,0,0]';
k = 0;
while(err > 1e-5)
    x1 = B*x0 + d;
    err = norm(x1-x0,inf)/norm(x1,inf);
    x0 = x1;
    k = k + 1;
end
fprintf('根据Sor迭代: \r\n\t在第%i次迭代后的迭代结果: \r\n',[k]);
disp(x1);
```

程序运行截图：

w	对应运行结果
0.8	>> a_sor(0.8) 根据Sor迭代: 在第11次迭代后的迭代结果: 0.3906 0.1678 0.0996 0.1754 0.0447

1.1	>> a_sor(1.1) 根据Sor迭代： 在第12次迭代后的迭代结果： 0.3906 0.1678 0.0996 0.1754 0.0447
1.2	>> a_sor(1.2) 根据Sor迭代： 在第14次迭代后的迭代结果： 0.3906 0.1678 0.0996 0.1754 0.0447
1.3	>> a_sor(1.3) 根据Sor迭代： 在第25次迭代后的迭代结果： 0.3906 0.1678 0.0996 0.1754 0.0447
1.4	>> a_sor(1.4) 根据Sor迭代： 在第65次迭代后的迭代结果： 0.3906 0.1678 0.0996 0.1754 0.0447

1.5	>> a_sor(1.5) 根据Sor迭代： 在第8355次迭代后的迭代结果： 1.0e+307 * 4.5090 8.5953 -Inf -1.2454 -6.4767

（2）

1、根据 Jacobi：

程序截图：

```
1   function res = a_j_2()
2   %This slice of codes responds to the 5-1 (2)problem.
3   %Jacobi method
4   A = [13.3,-4,-1,3.5,-3.8;3.4,9,-3,4.4,2.3;4.1,1,7,2.7,5.9;2.5,-2.4,1,13,5.6;1.5,-1,-3,4.3,14.9];
5   b = [5.8,4.3,2.6,3.8,4.2]';
6   len = length(b);
7   D1 = zeros(len,len);
8   for i = 1 : len
9       %At this moment, D is transfered into the inverse of D1
10      D1(i,i) = 1/A(i,i);
11  end
12  B = eye(len) - D1*A;
13  g = D1*b;
14  %The iteration part;
15  x1 = [0,0,0,0,0]';
16  x2 = [0,0,0,0,0]';
17  dx = 1;
18  k = 0;
19  while(dx > 1e-5)
20      x2 = B*x1 + g;
21      dx = norm(x2-x1,inf)/norm(x2,inf);
22      k = k + 1;
23      x1 = x2;
24  end
25  res = x2;
26  fprintf('根据Jacobi迭代：\r\n\t在第%i次迭代后的迭代结果：\r\n',[k]);
```

程序运行截图：

命令行窗口

```
>> res = a_j_2()
根据Jacobi迭代:

    在第26次迭代后的迭代结果:


res =

    0.4720
    0.1330
   -0.1302
    0.1628
    0.1701
```

2、根据 Gauss Seidel：

程序截图：

```
function a_gs_2()
%This slice of codes responds to the 5-1 (2)problem.
%Gauss-Seidel method for test
A = [13.3,-4,-1,3.5,-3.8;3.4,9,-3,4.4,2.3;4.1,1,7,2.7,5.9;2.5,-2.4,1,13,5.6;1.5,-1,-3,4.3,14.9];
b = [5.8,4.3,2.6,3.8,4.2]';
len = length(b);
D = eye(len);
D1 = eye(len);
x0 = [0,0,0,0,0]';
x1 = [0,0,0,0,0]';
for i = 1 : len
    D(i,i) = A(i,i);
    D1(i,i) = 1/A(i,i);
end
%The core part of Gauss Seidel
err = 1;
k = 0;
while(err > 1e-5)
    x1 = -1*D1*(A-D)*x0 + D1*b;
    err = norm((x1 - x0),inf)/norm(x1,inf);
    k = k + 1;
    x0 = x1;
end
fprintf('根据Gauss Seidel迭代: \r\n\t在第%i次迭代后的迭代结果: \r\n',[k]);
disp(x1);
```

程序运行截图：

```
>> a_gs_2()
根据Gauss Seidel迭代:

    在第26次迭代后的迭代结果:

    0.4720
    0.1330
   -0.1302
    0.1628
    0.1701
```

3、根据 Sor

程序截图：

```
function a_sor_2(w)
%w:松弛因子
%This slice of codes responds to the 5-1 (2)problem.
%Sor method
A = [13.3,-4,-1,3.5,-3.8;3.4,9,-3,4.4,2.3;4.1,1,7,2.7,5.9;2.5,-2.4,1,13,5.6;1.5,-1,-3,4.3,14.9];
b = [5.8,4.3,2.6,3.8,4.2]';
len = length(b);
D = eye(len);
for i = 1 : len
    D(i,i) = A(i,i);
end
L = tril(A - D);
U = triu(A - D);
%The main part of Sor
err = 1;
B = inv(D + w*L)*((1-w)*D - w*U);
d = w*inv(D + w*L)*b;
x0 = [0,0,0,0,0]';
x1 = [0,0,0,0,0]';
k = 0;
while(err > 1e-5)
    x1 = B*x0 + d;
    err = norm(x1-x0,inf)/norm(x1,inf);
    x0 = x1;
    k = k + 1;
end
fprintf('根据Sor迭代: \r\n\t在第%i次迭代后的迭代结果: \r\n',[k]);
disp(x1);
```

程序运行截图：

w	对应程序运行结果

0.8	>> a_sor_2(0.8) 根据Sor迭代： 在第10次迭代后的迭代结果： 0.4720 0.1330 -0.1302 0.1628 0.1701
1.1	>> a_sor_2(1.1) 根据Sor迭代： 在第18次迭代后的迭代结果： 0.4720 0.1330 -0.1302 0.1628 0.1701
1.2	>> a_sor_2(1.2) 根据Sor迭代： 在第28次迭代后的迭代结果： 0.4720 0.1330 -0.1302 0.1628 0.1701
1.3	>> a_sor_2(1.3) 根据Sor迭代： 在第64次迭代后的迭代结果： 0.4720 0.1330 -0.1302 0.1628 0.1701

1.4	>> a_sor_2(1.4) 根据Sor迭代： 在第8056次迭代后的迭代结果： 1.0e+307 * 3.1114 4.5017 -Inf -2.4248 -7.1981
1.5	>> a_sor_2(1.5) 根据Sor迭代： 在第2258次迭代后的迭代结果： 1.0e+307 * 4.3453 1.3372 -Inf -3.6806 -8.1765

5-2

实验目的：掌握 Newton 法与最速下降法求解非线性方程组，观察各自的优势。

实验内容：(1) 分别用 Newton 法与最速下降法求解下面非线性方程组

$$\begin{cases} 3x_1 - \cos(x_2 x_3) - 0.5 = 0 \\ x_1^2 - 81(x_2 + 0.1)^2 + sinx_3 + 1.06 = 0 \\ e^{-x_1 x_2} + 20x_3 + \dfrac{10\pi - 3}{3} = 0 \end{cases}$$

初值 $x^{(0)}=(0.1, 0.1, -0.1)^T$，精度要求：$\frac{\left\|X^{(k+1)}-X^{(k)}\right\|_\infty}{\left\|X^{(k+1)}\right\|_\infty} < 10^{-5}$。

（2）改变初值 $x^{(0)}=(20, 20, 20)^T$，再用如上两种方法求解，得到什么结果。

（3）采用初值 $x^{(0)}=(20, 20, 20)^T$，先用最速下降法求解 3 步，再用 Newton 迭代法，得到什么结果？对以上运算结果做分析。

（1）

Newton:

$$f_1(x_1,x_2,x_3)=3x_1-\cos(x_2x_3)-0.5$$

$$f_2(x_1,x_2,x_3)=x_1^2-81(x_2+0.1)^2+\sin x_3+1.06$$

$$f_3(x_1,x_2,x_3)=e^{-x_1x_2}+20x_3+\frac{10\pi-3}{3}$$

$$\mathbf{F}(x_1,x_2,x_3)=\begin{pmatrix}f_1(x_1,x_2,x_3)\\f_2(x_1,x_2,x_3)\\f_3(x_1,x_2,x_3)\end{pmatrix}$$

$$\mathbf{F}'(x_1,x_2,x_3)=\begin{bmatrix}3 & x_3\sin(x_2x_3) & x_2\sin(x_2x_3)\\2x_1 & -162(x_2+0.1) & \cos(x_3)\\-x_2e^{-x_1x_2} & -x_1e^{-x_1x_2} & 20\end{bmatrix}$$

程序截图：

```
function b_n(a)
%This slice of codes responds to the 5-2 problem.
%Newton method
err = 1;
k = 0;
b = [0,0,0];
while(err > 1e-5)
    f1 = 3*a(1) - cos(a(2)*a(3)) - 0.5;
    f2 = a(1)^2 - 81*((a(2) + 0.1)^2) + sin(a(3)) + 1.06;
    f3 = exp(-a(1)*a(2)) + 20*a(3) + (10*pi - 3)/3;
    F = [f1,f2,f3]';
    dF = [3,a(3)*sin(a(2)*a(3)),a(2)*sin(a(2)*a(3));2*a(1),-162*(a(2) + 0.1),cos(a(3));-a(2)*exp(-a(1)*a(2)),-a(1)*exp(-a(1)*a(2)),20];
    b = a - inv(dF)*F;
    err = norm(b - a,inf)/norm(b,inf);
    k = k + 1;
    a = b;
end
fprintf('根据Newton method迭代: \r\n\t在第%i次迭代后的迭代结果: \r\n',[k]);
disp(b);
```

程序运行截图：

```
a =

    0.1000
    0.1000
   -0.1000

>> b_n(a)
根据Newton method迭代:

    在第5次迭代后的迭代结果:

    0.5000
   -0.0000
   -0.5236
```

最速下降法：

程序截图：

```
function b_e(x)
%This slice of codes responds to the 5-2 problem.
%Steepest descent method
syms x1 x2 x3 r;
y = (3*x1 - cos(x2*x3) - 0.5)^2 + (x1^2 - 81*(x2 + 0.1)^2 + sin(x3) + 1.06)^2 + (exp(-x1*x2) + 20*x3 + (10*pi - 3)/3)^2;
ydx1 = diff(y,x1);
ydx2 = diff(y,x2);
ydx3 = diff(y,x3);
%Iteration part
err = 1;
a = x;
G = [0,0,0]';
Ymin = 1000;
R = 0;
B = [0,0,0]';
k = 0;
while(err > 1e-5)
    G(1) = -subs(ydx1,{x1,x2,x3},{a(1),a(2),a(3)});
    G(2) = -subs(ydx2,{x1,x2,x3},{a(1),a(2),a(3)});
    G(3) = -subs(ydx3,{x1,x2,x3},{a(1),a(2),a(3)});
    %for r = 0 : 0.0001 : 1
    for r = 0 : 0.00002 : 0.0001
        b = a + r*G;
        y1 = subs(y,{x1,x2,x3},{b(1),b(2),b(3)});
        if(Ymin > y1)
            Ymin = y1;
            R = r;
            B = b;
        end
    end
    k = k + 1;
    err = norm(B-a,inf)/norm(B,inf);
    a = B;
end
fprintf('根据Steepest descent method迭代: \r\n\t在第%i次迭代后的迭代结果: \r\n',[k]);
disp(B);
```

程序运行截图：

命令行窗口

```
>> b_e(a)
根据Steepest descent method迭代:

    在第2738次迭代后的迭代结果:

    0.4971
   -0.0002
   -0.5236
```

（2）将 $x^{(0)}=(20, 20, 20)^T$ 带入程序，

Newton 法程序运行截图：

```
>> b_n(a)
警告: 矩阵为奇异值、接近奇异值或缩放错误。结果可能不准确。RCOND = NaN。
> In b_n (line 13)
根据Newton method迭代:

    在第2次迭代后的迭代结果:

   NaN
   NaN
   NaN
```

最速下降法程序运行截图：

```
>> b_e(a)
根据Steepest descent method迭代:

    在第2866次迭代后的迭代结果:

    0.4971
   -0.0002
   -0.5236
```

（3）对之前的最速下降法的程序进行下幅度的修改后，程序截图：

```
function res = b_e_1(x)
syms x1 x2 x3 r;
y = (3*x1 - cos(x2*x3) - 0.5)^2 + (x1^2 - 81*(x2 + 0.1)^2 + sin(x3) + 1.06)^2 + (exp(-x1*x2) + 20*x3 + (10*pi - 3)/3)^2;
ydx1 = diff(y,x1);
ydx2 = diff(y,x2);
ydx3 = diff(y,x3);
%Iteration part
%err = 1;
a = x;
G = [0,0,0]';
Ymin = 1000;
R = 0;
B = [0,0,0]';
k = 0;
while(k < 3)
    G(1) = -subs(ydx1,{x1,x2,x3},{a(1),a(2),a(3)});
    G(2) = -subs(ydx2,{x1,x2,x3},{a(1),a(2),a(3)});
    G(3) = -subs(ydx3,{x1,x2,x3},{a(1),a(2),a(3)});
    %for r = 0 : 0.0001 : 1
    for r = 0 : 0.00002 : 0.0001
        b = a + r*G;
        y1 = subs(y,{x1,x2,x3},{b(1),b(2),b(3)});
        if(Ymin > y1)
            Ymin = y1;
            R = r;
            B = b;
        end
    end
    k = k + 1;
    err = norm(B-a,inf)/norm(B,inf);
    a = B;
end
fprintf('根据Steepest descent method迭代：\r\n\t在第%i次迭代后的迭代结果：\r\n',[k]);
res = B;
```

程序运行截图：

```
>> b_n(b_e_1(x))
根据Steepest descent method迭代：

    在第3次迭代后的迭代结果：

    0.0018
    0.0014
   -0.0805

根据Newton method迭代：

    在第5次迭代后的迭代结果：

    0.5000
   -0.0000
   -0.5236
```

6-1

用规范的幂法与反幂法求矩阵 A 的按模最大、最小特征值与对应的特征向量。

$A=\begin{bmatrix}4 & 1 & 1 & 1\\ 1 & 3 & -1 & 1\\ 1 & -1 & 2 & 0\\ 1 & 1 & 0 & 2\end{bmatrix}$，$e = 10^{-5}$。

幂法：

程序截图：

```
function a1()
%This slice of codes responds to the 6-1 (1)problem.
%Use Power law method for eigenvalue
%A = [133,6,135;44,5,46;-88,-6,-90];
%V0 = [1,1,1]';

A = [4,1,1,1;1,3,-1,1;1,-1,2,0;1,1,0,2];
V0 = [1,1,1,1]';
err = 1;
%当前后两次迭代的结果误差在1e-5以内，则认为结果符合误差精度
m = 0;
k = 0;
x = 0;
while(err > 1e-5)
    for i = 1 : length(V0)-1
        if(abs(V0(i)) > abs(V0(i+1)))
            m = V0(i);
        else
            m = V0(i+1);
        end
    end
    U = V0/m;
    V1 = A*U;
    err = abs(x-m);
    k = k + 1;
    x = m;
    V0 = V1;
end
fprintf('根据幂法迭代: \r\n\t在第%i次迭代后的迭代结果: \r\n\t',[k]);
fprintf('矩阵A的最大特征值是: \r\n\t\t%f\r\n\t',[m]);
fprintf('对应的特征向量是: \r\n');
disp(U);
```

程序运行截图：

```
>> a1()
根据幂法迭代：

    在第25次迭代后的迭代结果：

    矩阵A的最大特征值是：

        5.236086

    对应的特征向量是：

    2.0000
    1.2360
    0.2361
    1.0000
```

```
>> a1()
根据幂法迭代：

    在第2次迭代后的迭代结果：

    矩阵A的特征值是：

        5.428571

    对应的特征向量是：

    1.0000
    0.5714
    0.2857
    0.5714
```

反幂法：

程序截图：

```
function a2()
%This slice of codes responds to the 6-1 (2) problem.
%Use Iverse Power law method for eigenvalue
A = [4,1,1,1;1,3,-1,1;1,-1,2,0;1,1,0,2];
V0 = [1,1,1,1]';

%A = [-1,2,1;2,-4,1;1,1,-6];
%A = [1.5,0,0;0,2,0;0,0,3];
%V0 = [1,1,1]';
A1 = inv(A);
err = 1;
%当前后两次迭代的结果误差在1e-5以内，则认为结果符合误差精度
m = 0;
k = 0;
x = 0;
while(err > 1e-10)
    for i = 1 : (length(V0) - 1)
        if(abs(V0(i)) > abs(V0(i+1)))
            m = V0(i);
        else
            m = V0(i+1);
        end
    end
    %fprintf('m=%f\r\n',[m]);
    U = V0/m;
    V1 = A1*U;
    %disp(V1);
    err = abs(x - m);
    x = m;
    k = k + 1;
    V0 = V1;
end
fprintf('根据反幂法，经过%i次的迭代计算，矩阵A的最小特征值为：%f\r\n\t',[k,1/m]);
fprintf('对应的特征向量为：%f,%f,%f',[1/U(1),1/U(2),1/U(3)]);
```

程序运行截图：

命令行窗口

```
>> a2()
根据反幂法，经过37次的迭代计算，矩阵A的最小特征值为：0.763932

    对应的特征向量为：-2.118034,1.309017,1.000000>>
```

6-2

用 Householder 变换求矩阵 A 的 QR 方法做三次迭代。

$A=\begin{bmatrix}4 & 1 & 1 & 1\\ 1 & 3 & -1 & 1\\ 1 & -1 & 2 & 0\\ 1 & 1 & 0 & 2\end{bmatrix}$。

程序截图：

```
function b1()
A = [4,1,1,1;1,3,-1,1;1,-1,2,0;1,1,0,2];
e = eye(4);

x1 = A(1:4,1);
q1 = -sign(x1(1))*norm(x1,2);
p1 = q1*(q1 - x1(1));
u1 = x1 - q1*e(1:4,1);
h1 = e - (1/p1)*u1*(u1');
fprintf('h1=\r\n');
disp(h1);
A1 = h1*A;
fprintf('A1=\r\n');
disp(A1);

x2 = A1(1:4,2);
q2 = -sign(x2(2))*norm(x2(2:4),2);
p2 = q2*(q2 - x2(2));
u2 = [0,x2(2) - q2,x2(3),x2(4)]';
h2 = e - (1/p2)*u2*u2';
fprintf('h2=\r\n');
disp(h2);
A2 = h2*A1;
fprintf('A2=\r\n');
disp(A2);

x3 = A2(1:4,3);
q3 = -sign(x3(3))*norm(x3(3:4),2);
p3 = q3*(q3 - x3(3));
u3 = [0,0,x3(3)-q3,x3(4)]';
h3 = e - (1/p3)*u3*u3';
fprintf('h3=\r\n');
disp(h3);
A3 = h3*A2;
fprintf('A3=\r\n');
disp(A3);
```

程序运行截图：

命令行窗口

```
>> b1()
h1=

   -0.9177   -0.2294   -0.2294   -0.2294
   -0.2294    0.9726   -0.0274   -0.0274
   -0.2294   -0.0274    0.9726   -0.0274
   -0.2294   -0.0274   -0.0274    0.9726

A1=

   -4.3589   -1.6059   -1.1471   -1.6059
   -0.0000    2.6882   -1.2569    0.6882
   -0.0000   -1.3118    1.7431   -0.3118
         0    0.6882   -0.2569    1.6882

h2=

    1.0000         0         0         0
         0   -0.8758    0.4274   -0.2242
         0    0.4274    0.9026    0.0511
         0   -0.2242    0.0511    0.9732

A2=

   -4.3589   -1.6059   -1.1471   -1.6059
    0.0000   -3.0694    1.9034   -1.1146
   -0.0000    0.0000    1.0231    0.0990
    0.0000   -0.0000    0.1209    1.4727

h3=

    1.0000         0         0         0
         0    1.0000         0         0
         0         0   -0.9931   -0.1173
         0         0   -0.1173    0.9931

A3=

   -4.3589   -1.6059   -1.1471   -1.6059
    0.0000   -3.0694    1.9034   -1.1146
    0.0000   -0.0000   -1.0303   -0.2711
    0.0000   -0.0000    0.0000    1.4510
```

7-2

实验目的：观察 lagrange 插值的 Runge 现象，了解若能采用合适的节点分布，则可以避免 Runge 现象。熟悉三次样条插值。

实验内容：对于函数$\mathbf{f(x)=\frac{1}{1+25x^2}(-1\le x\le 1)}$进行 lagrange 插值。取不同的等分数 n=5，10，将区间[-1，1]n 等分，取等距节点。把 f(x)和 5 次，10 次插值多项式的曲线画在同一张图上进行比较。再取 Chebyshev 节点$x_k=-\cos\frac{k\pi}{n}$，k=0，1，…，10 进行 lagrange 插值，把 f(x)和 Chebyshev 节点的 10 次插值多项式的曲线画在同一张图上。

（1）把 f(x)和 5 次，10 次插值多项式的曲线画在同一张图上进行比较：

程序截图：

```
function al()
syms x;
f = 1/(1 + 25*x^2);
a = -1 : 2/5 : 1;
%将区间五等分
for i = 1 : 6
    y5(i) = subs(f,x,a(i));
end
fprintf('y5=\r\n');
disp(y5);
%计算多项式插值公式
fa = 0;
for i = 1 : 6
    fa1 = 1;
    for k = 1 : 6
        if(i ~= k)
            fa1 = fa1*(x-a(k))/(a(i) - a(k));
        end
    end
    fa = fa + y5(i)*fa1;
end
fprintf('将区间五等分后，根据Lagrange插值得到的多项式公式为：fa=\r\n\t')
disp(fa);
disp(subs(fa,x,-1));
%将区间十等分
b = -1 : 2/10 : 1;
for i = 1 : 11
    y10(i) = subs(f,x,b(i));
end
fprintf('y10=\r\n');
disp(y10);
%计算多项式插值公式
fb = 0;
for i = 1 : 11
    fb1 = 1;
    for k = 1 : 11
        if(i ~= k)
            fb1 = fb1*(x-b(k))/(b(i) - b(k));
        end
    end
```

```
41 -        fb = fb + y10(i)*fb1;
42 -    end
43 -    fprintf('将区间十等分后，根据Lagrange插值得到的多项式公式为：fb=\r\n\t');
44 -    disp(fb);
45
46      %Draw graphic
47 -    u = -1 : 2/100 : 1;
48 -    for i = 1 : length(u)
49 -        y(i) = subs(f,x,u(i));
50 -        y5(i) = subs(fa,x,u(i));
51 -        y10(i) = subs(fb,x,u(i));
52 -    end
53 -    plot(u,v,'-.b',u,v5,'--r*',u,v10,'-.gh');
```

程序运行截图：

```
>> al()
y5=

[ 1/26,  1/10,  1/2,  1/2,  1/10,  1/26]

将区间五等分后，根据Lagrange插值得到的多项式公式为：fa=

    (625*(x/2 + 1/2)*(x - 1/5)*(x + 1/5)*(x - 3/5)*(x + 3/5))/9

1/26

y10=

[ 1/26,  1/17,  1/10,  1/5,  1/2,  1,  1/2,  1/5,  1/10,  1/17,  1/26]

将区间十等分后，根据Lagrange插值得到的多项式公式为：fb=

    (78125*x*(5*x + 4)*(x - 1)*(x - 1/5)*(x + 1/5)*(x - 2/5)*(x
```

拟合公式太长：单独黏贴出来为：

五次插值多项式拟合公式：

$f_5(x)$= (625*(x/2 + 1/2)*(x - 1/5)*(x + 1/5)*(x - 3/5)*(x + 3/5))/9984 -

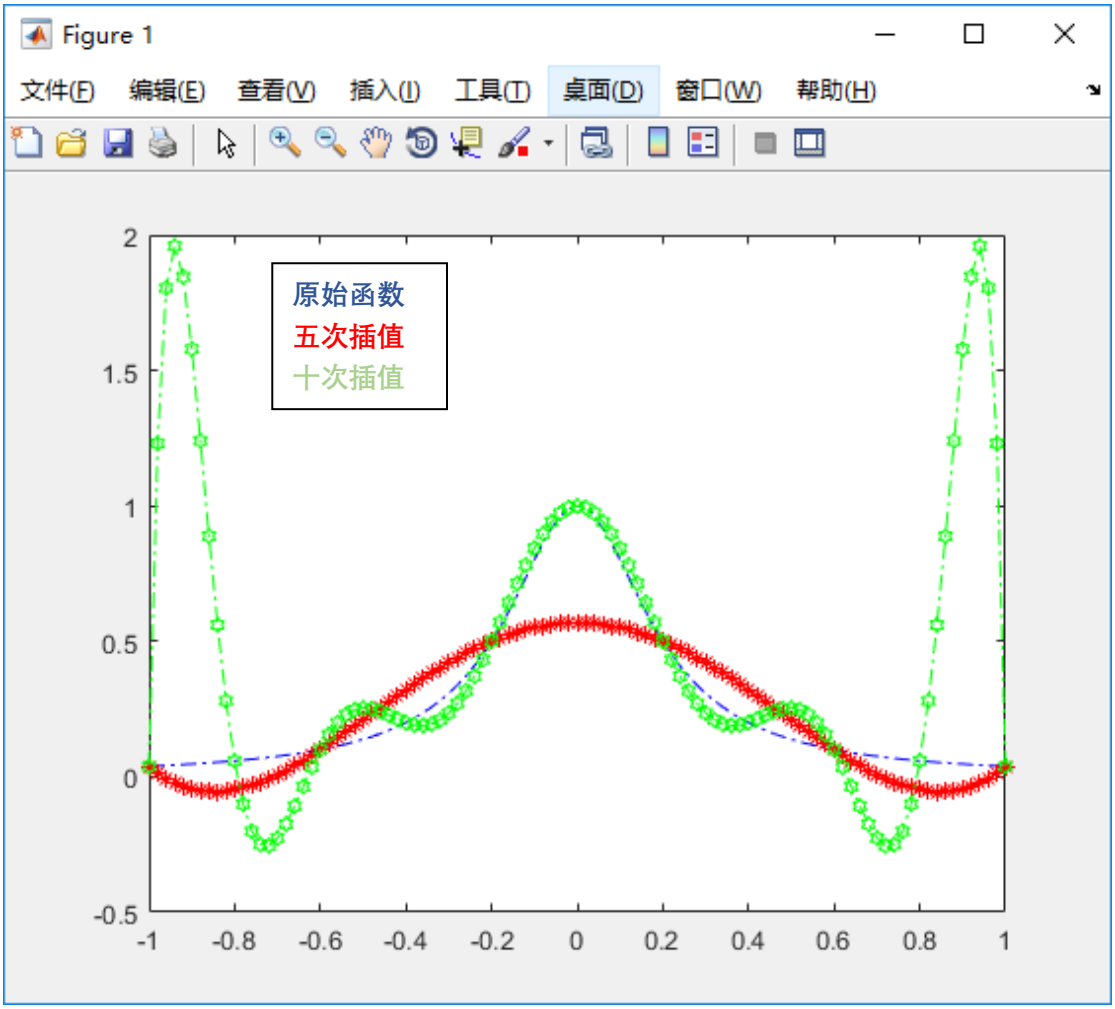
(125*((5*x)/2 + 3/2)*(x - 1)*(x - 1/5)*(x + 1/5)*(x - 3/5))/9984 +

(125*((5*x)/2 + 5/2)*(x - 1)*(x - 1/5)*(x + 1/5)*(x - 3/5))/768 -

(625*((5*x)/4 + 5/4)*(x - 1)*(x - 1/5)*(x - 3/5)*(x + 3/5))/192 + (625*((5*x)/6 + 5/6)*(x - 1)*(x + 1/5)*(x - 3/5)*(x + 3/5))/128 - (125*((5*x)/8 + 5/8)*(x - 1)*(x - 1/5)*(x + 1/5)*(x + 3/5))/192

十次插值多项式拟合公式：

$f_{10}(x)$=(78125*x*(5*x + 4)*(x - 1)*(x - 1/5)*(x + 1/5)*(x - 2/5)*(x + 2/5)*(x - 3/5)*(x + 3/5)*(x - 4/5))/3773952 - (390625*x*(5*x + 5)*(x - 1)*(x - 1/5)*(x + 1/5)*(x - 2/5)*(x + 2/5)*(x - 3/5)*(x + 3/5)*(x - 4/5))/1233792 + (390625*x*(x/2 + 1/2)*(x - 1/5)*(x + 1/5)*(x - 2/5)*(x + 2/5)*(x - 3/5)*(x + 3/5)*(x - 4/5)*(x + 4/5))/1886976 + (78125*x*((5*x)/2 + 5/2)*(x - 1)*(x - 1/5)*(x + 1/5)*(x - 2/5)*(x + 2/5)*(x - 3/5)*(x - 4/5)*(x + 4/5))/16128 - (78125*x*((5*x)/3 + 5/3)*(x - 1)*(x - 1/5)*(x + 1/5)*(x - 2/5)*(x - 3/5)*(x + 3/5)*(x - 4/5)*(x + 4/5))/2016 + (390625*x*((5*x)/4 + 5/4)*(x - 1)*(x - 1/5)*(x - 2/5)*(x + 2/5)*(x - 3/5)*(x + 3/5)*(x - 4/5)*(x + 4/5))/1728 + (390625*x*((5*x)/6 + 5/6)*(x - 1)*(x + 1/5)*(x - 2/5)*(x + 2/5)*(x - 3/5)*(x + 3/5)*(x - 4/5)*(x + 4/5))/1152 - (78125*x*((5*x)/7 + 5/7)*(x - 1)*(x - 1/5)*(x + 1/5)*(x + 2/5)*(x - 3/5)*(x + 3/5)*(x - 4/5)*(x + 4/5))/864 + (78125*x*((5*x)/8 + 5/8)*(x - 1)*(x - 1/5)*(x + 1/5)*(x - 2/5)*(x + 2/5)*(x + 3/5)*(x - 4/5)*(x + 4/5))/4032 - (390625*x*((5*x)/9 + 5/9)*(x - 1)*(x - 1/5)*(x + 1/5)*(x - 2/5)*(x + 2/5)*(x - 3/5)*(x + 3/5)*(x + 4/5))/137088 - (390625*(x - 1)*(x + 1)*(x - 1/5)*(x + 1/5)*(x - 2/5)*(x + 2/5)*(x - 3/5)*(x + 3/5)*(x - 4/5)*(x + 4/5))/576



(2) 再取 Chebyshev 节点$x_k = -\cos\frac{k\pi}{n}$, k=0, 1, ···, 10 进行 lagrange 插值，把 f(x)和 Chebyshev 节点的 10 次插值多项式的曲线画在同一张图上

程序截图：

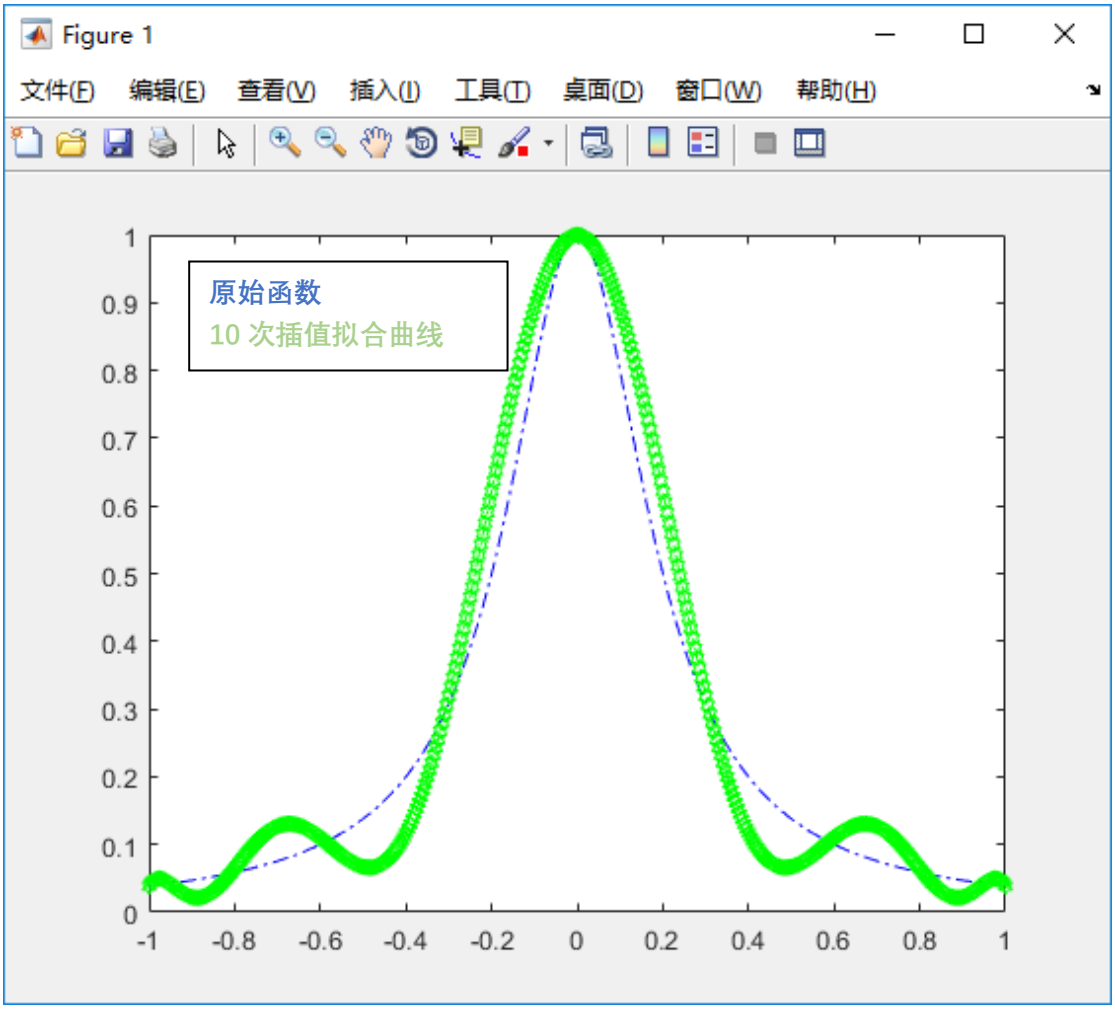
```
function ac()
syms x;
f = 1/(1 + 25*x^2);

for i = 1 : 11
    b(i) = -cos((i-1)*pi/10);
    yb(i) = subs(f,x,b(i));
end

fb = 0;
for i = 1 : 11
    fb1 = 1;
    for k = 1 : 11
        if(i ~= k)
            fb1 = fb1*(x - b(k))/(b(i) - b(k));
        end
    end
    fb = fb + yb(i)*fb1;
end

u = -1 : 1/200 : 1;
for i = 1 : length(u)
    y(i) = subs(f,x,u(i));
    y10(i) = subs(fb,x,u(i));
end
plot(u,y,'-.b',u,y10,'-.gh');
```

程序运行截图：



9-1

实验目的：熟悉数值积分公式，掌握数值计算定积分的方法

实验内容：采用不同方法数值计算积分

$$\int_0^1 \frac{In(1+x)}{x} dx$$

（1）编写复化梯形公式和复化 Simpson 公式通用子程序，分别采用 4，8，16，32，64 等分区间计算。

（2）使用 Romberg 求积公式。

（3）使用高斯-勒让德求积公式(n=2，4，8)。

（1）复化梯形公式：

$$\int_a^b f(x) = \frac{(b-a)}{2}\left(f(a)+f(b)\right)$$

程序截图：

```
function at(n)
syms x;
f = log(x + 1)/x;
a = 0.001 : 1/n : 1;
a(n+1) = 1;
sum = 0;
for i = 1 : n
    sum = sum + (1/n)*(subs(f,x,a(i))+subs(f,x,a(i+1)))/2;
end
fprintf('当积分区间被%i等分时，积分结果为：\r\n\t',[n]);
disp(vpa(sum));
```

程序运行截图：

```
>> at(4)
当积分区间被4等分时，积分结果为：

    0.82377279926042567716932264295925

>> at(8)
当积分区间被8等分时，积分结果为：

    0.82257080880412003955570974221188

>> at(16)
当积分区间被16等分时，积分结果为：

    0.82226605393169759449176657660948

>> at(32)
当积分区间被32等分时，积分结果为：

    0.82218827767234910941554412524353

>> at(64)
当积分区间被64等分时，积分结果为：

    0.82216807442202779924084420750857
```

复化 Simpson 公式：

$$\int_a^b f(x) = \frac{(b-a)}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

程序截图：

```
function as(n)
syms x;
f = log(x + 1)/x;
a = 0.001 : 1/n : 1;
a(n+1) = 1;
sum = 0;
for i = 1 : n
    sum = sum + (1/n)*(subs(f,x,a(i)) + 4*subs(f,x,(a(i) + a(i+1))/2)+subs(f,x,a(i+1)))/6;
end
fprintf('当积分区间被%i等分时，积分结果为：\r\n\t',[n]);
disp(vpa(sum));
```

程序运行截图：

命令行窗口

```
>> as(4)
当积分区间被4等分时，积分结果为：

    0.82218776176367958200552855963772

>> as(8)
当积分区间被8等分时，积分结果为：

    0.82217288050941972402665744676071

>> as(16)
当积分区间被16等分时，积分结果为：

    0.82216646432158827613539862748363

>> as(32)
当积分区间被32等分时，积分结果为：

    0.82216337325819790809965156803444

>> as(64)
当积分区间被64等分时，积分结果为：

    0.82216184778951134618278616221965
```

（2）Romberg 求积：

$$T_{1,k} = T_{2^k} = \frac{1}{2}\left[T_{1,k-1} + h_{k-1}\sum_{i=1}^{2^{k-1}} f\left(a + h_k(2i-1)\right)\right], h_k = \frac{b-a}{2^k}$$

$$T_{m+1,k} = T_{m,k+1} + \frac{T_{m,k+1} - T_{m,k}}{4^m - 1}$$

程序截图：

```
function r()
syms x;
f = log(x + 1)/x;
a = 0.001 : 1/8 : 1;
a(9) = 1;
for i = 1 : 9
    y(i) = subs(f,x,a(i));
end
T1(1) = (1/2)*(y(1) + y(9));
T1(2) = (1/2)*(T1(1) + y(5));
T1(3) = (1/2)*(T1(2) + 0.5*(y(3) + y(7)));
T1(4) = (1/2)*(T1(3) + 0.25*(y(2) + y(4) + y(6) + y(8)));
for i = 1 : length(T1) - 1
    T2(i) = T1(i+1) + (T1(i+1)-T1(i))/3;
end
for i = 1 : length(T2) - 1
    T3(i) = T2(i+1) + (T2(i+1) - T2(i))/15;
end
T4 = T3(2) + (T3(2) - T3(1))/63;
fprintf('根据Romberg求积公式计算得: T4 = %f',[T4]);
```

程序运行截图：

```
>> r()
根据Romberg求积公式计算得: T4 = 0.822168>>
```

（3）高斯-勒让德求积公式(n=2，4，8)

$$\int_0^1 \frac{In(1+x)}{x}dx = \int_{-1}^1 \frac{In\left(\frac{t+3}{2}\right)}{t+1}dt, \left(x = \frac{t+1}{2}\right)$$

$$\mathrm{f}(t) = \frac{In\left(\frac{t+3}{2}\right)}{t+1}$$

程序截图：

```
function gauss()
x2 = [-0.5773502692,0.5773502692];
A2 = [1,1];
x4 = [-0.8611363116,-0.3399810436,0.3399810436,0.8611363116];
A4 = [0.3478548451,0.6521451549,0.6521451549,0.3478548451];
x8 = [-0.9602898565,-0.7966664774,-0.5255324099,-0.1834346425,0.1834346425,0.5255324099,0.7966664774,0.9602898565];
A8 = [0.1012285363,0.2223810345,0.3137066459,0.3626837834,0.3626837834,0.3137066459,0.2223810345,0.1012285363];

I2 = 0;
I4 = 0;
I8 = 0;

syms t;
f = log((t + 3)/2)/(t + 1);

for i = 1 : 2
    I2 = I2 + A2(i)*subs(f,t,x2(i));
end
fprintf('当n=2时，根据高斯-勒让德求积公式得I2 = %f\r\n',[I2]);

for  i = 1 : 4
    I4 = I4 + A4(i)*subs(f,t,x4(i));
end
fprintf('当n=4时，根据高斯-勒让德求积公式得I2 = %f\r\n',[I4]);

for i = 1 : 8
    I8 = I8 + A8(i)*subs(f,t,x8(i));
end
fprintf('当n=8时，根据高斯-勒让德求积公式得I2 = %f\r\n',[I8]);
```

程序运行截图：

```
>> gauss()
当n=2时，根据高斯-勒让德求积公式得I2 = 0.822242

当n=4时，根据高斯-勒让德求积公式得I2 = 0.822467

当n=8时，根据高斯-勒让德求积公式得I2 = 0.822467
```