《数值分析》实验报告一

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 专业 | 数学与应用数学 | 班级 | 000 | 姓名 | 挖掘机大王子 | 学号 | 2015040201 |

**实验项目**

累计求和的练习。

**实验内容**

用复合梯形公式、复合辛普森公式算定积分



**算法设计分析**

1.



**= *Tn***



2.

**= *Sn***

**实验测试结果及结果分析**

## 1.函数functionT1

>> format long%16为有效数字

>> f=@(x)x^2%定义被积函数函数

>> functionT1(f,1,2,100)

ans =2.333350000000001

>> functionT1(f,1,2,1000)

ans =2.333333499999998

>> functionT1(f,1,2,10000)

ans =2.333333334999989

>> functionT1(f,1,2,100000)

ans =2.333333333350027

>> functionT1(f,1,2,1000000)

ans =2.333333333333480

>> functionT1(f,1,2,10000000)

ans =2.333333333333582

## 2.函数functionT2

>> functionT2(f,1,2,100)

ans =2.333333333333334

>> functionT2(f,1,2,1000)

ans =2.333333333333333

>> functionT2(f,1,2,10000)

ans =2.333333333333326

>> functionT2(f,1,2,100000)

ans =2.333333333333343

>> functionT2(f,1,2,1000000)

ans =2.333333333333328

>> functionT2(f,1,2,10000000)

ans =2.333333333333257

## 运行结果分析

理论真实值为



|  |  |  |
| --- | --- | --- |
| **n值** | **functionT1(梯形公式)** | **functionT2（辛普森公式）** |
| 1000 | 2.333350000000001 | 2.333333333333334 |
| 10000 | 2.333333499999998 | 2.333333333333333 |
| 100000 | 2.333333334999989 | 2.333333333333326 |
| 1000000 | 2.333333333350027 | 2.333333333333343 |
| 10000000 | 2.333333333333480 | 2.333333333333328 |
| 100000000 | 2.333333333333582 | 2.333333333333257 |

通过观察表格可知，在n值相同的情况下辛普森公式比较接近理论真实值。

**实验总结**

1. matlab中的代码最好写成文件代码的形式，方便修改
2. matlab中后缀为m的文件存储类型有函数或者是命令
3. 在matlab函数的定义可以用格式:变量名=@(输入参数列表)运算表达式,例如：f=@(x)x^2
4. 通过本次实验深入了解辛普森求公式梯形面积的极限思想

**附录 实验程序代码（加注释）**

## 1.函数functionT1代码

%定义复合梯形函数functionT1,传入参数f为被积函数,a、b为积分上下限，n为区间分割量

%函数开始

function [ T ] = functionT1( f,a,b,n )

%h的表达式

h=(b-a)/n;

%复合梯形公式累加部分开始

E=0;

for k=1:(n-1)

xk=a+k\*h;

E=E+f(xk);

end

%复合梯形公式累加部分结束

%复合梯形总算法公式

T=h/2\*(f(a)+2\*E+f(b))

%函数结束

end

## 2.函数functionT2代码

%定义辛普森符合梯形函数functionT2,传入参数f为被积函数,a、b为积分上下限，n为区间分割量

%函数开始

function [ T ] = functionT2( f,a,b,n )

%h的表达式

h=(b-a)/n;

%辛普森复合梯形公式累加部分1开始

E1=0;

for k=0:(n-1)

xk=a+k\*h;

E1=E1+f(xk+h/2);

end

%辛普森复合梯形公式累加部分1结束

%辛普森复合梯形公式累加部分2开始

E2=0;

for k=1:(n-1)

xk=a+k\*h;

E2=E2+f(xk);

end

%辛普森复合梯形公式累加部分2结束

%辛普森复合梯形总算法公式

T=h/6\*(f(a)+4\*E1+2\*E2+f(b));

%函数结束

end

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 项目 | 得分 | 总分 |
| 1 | 实验报告排版(5分) |  |  |
| 2 | 算法思想分析(6分) |  |
| 3 | 源代码(7分) |  |
| 4 | 实验结果及分析(7分) |  |

《数值分析》实验报告二

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 专业 | 数学与应用数学 | 班级 | 000 | 姓名 | 挖掘机大王子 | 学号 | 20150402201 |

**实验项目**

迭代法的练习。

**实验内容**

1 用迭代公式，计算3的平方根，使误差不超过10-6

2 解3次代数方程，（1.5为初值）用不动点迭代法，牛顿迭代法，使误差不超过10-6

**算法设计分析**

**实验测试结果及结果分析**

### 1.迭代函数Root

>> format long%16位有效数字

>> [ak2,k]=Root(3,10^(-1))

ak2 =1.732142857142857

k =3

>> [ak2,k]=Root(3,10^(-2))

ak2 =1.732050810014727

k =4

>> [ak2,k]=Root(3,10^(-3))

ak2 =1.732050810014727

k =4

>> [ak2,k]=Root(3,10^(-4))

ak2 =1.732050810014727

k =4

>> [ak2,k]=Root(3,10^(-5))

ak2 =1.732050807568877

k =5

>> [ak2,k]=Root(3,10^(-6))

ak2 =1.732050807568877

k =5

>> [ak2,k]=Root(3,10^(-7))

ak2 =1.732050807568877

k =5

>> [ak2,k]=Root(3,10^(-100))

ak2 =1.732050807568877

k =6

## 运行结果分析

理论真实值为：>> I=sqrt(3)= 1.732050807568877

|  |  |  |
| --- | --- | --- |
| 误差值e | 迭代次数k | 近似值ak2 |
| 10^(-1) | 3 | 1.732142857142857 |
| 10^(-2) | 4 | 1.732050810014727 |
| 10^(-3) | 4 | 1.732050810014727 |
| 10^(-4) | 4 | 1.732050807568877 |
| 10^(-5) | 5 | 1.732050807568877 |
| 10^(-6) | 5 | 1.732050807568877 |
| 10^(-7) | 5 | 1.732050807568877 |
| 10^(-1000) | 6 | 1.732050807568877 |

从表中可以看出误差为10^(-6)的迭代次数k=5开始接近真实真实值I

## 2.（1）不动点函数RootB

>> [a,k]=RootB(1.5,10^(-1))

a =1.330860958801428

k =2

>> [a,k]=RootB(1.5,10^(-2))

a =1.325883774232348

k =3

>> [a,k]=RootB(1.5,10^(-3))

a =1.324939363401885

k =4

>> [a,k]=RootB(1.5,10^(-4))

a =1.324725945226887

k =6

>> [a,k]=RootB(1.5,10^(-5))

a =1.324719474534364

k =7

>> [a,k]=RootB(1.5,10^(-6))

a =1.324718011988197

k =9

>> [a,k]=RootB(1.5,10^(-7))

a =1.324717967643087

k =10

>> [a,k]=RootB(1.5,10^(-100))

a =1.324717957244746

k =22

## （2）牛顿迭代法函数RootN

>> f=@(x)x^3-x-1;%原函数

>> g=@(x)3\*x^2-1;%原函数f的导函数

>> [x,k]=RootN(f,g,10^(-1))

x =1.325200398950907

k =2

>> [x,k]=RootN(f,g,10^(-2))

x =1.324718173999054

k =3

>> [x,k]=RootN(f,g,10^(-3))

x =1.324718173999054

k =3

>> [x,k]=RootN(f,g,10^(-4))

x =1.324717957244790

k =4

>> [x,k]=RootN(f,g,10^(-5))

x =1.324717957244790

k =4

>> [x,k]=RootN(f,g,10^(-6))

x =1.324717957244790

k =4

>> [x,k]=RootN(f,g,10^(-7))

x =1.324717957244746

k =5

>> [x,k]=RootN(f,g,10^(-1000))

x =1.324717957244746

k =6

## 运行结果分析

|  |  |  |
| --- | --- | --- |
| 误差值e | 不动点迭代法(k) | 牛顿迭代法(k) |
| 10^(-1) | 2 | 2 |
| 10^(-2) | 3 | 3 |
| 10^(-3) | 4 | 3 |
| 10^(-4) | 6 | 4 |
| 10^(-5) | 7 | 4 |
| 10^(-6) | 9 | 4 |
| 10^(-7) | 10 | 5 |
| 10^(-1000) | 22 | 6 |

当误差为10^(-6)时，可以出牛顿迭代法更精准，而且纵观整个迭代过程，当误差值更加小的时候，牛顿迭代法趋近真实值的程度要更快一些。

**实验总结**

1.通过实验测试三种迭代算法通过比较之后得出他们的优点和缺点

2.实验分析过程中需要仔细观察每个结果的细微变动。

3.通过实验深刻了解各种迭代算法的原理。

**附录 实验程序代码（加注释）**

## 1.迭代函数Root

%定义迭代函数Root求开根，ak1为开方数、e为误差值

%返回值ak2为近似开根数，k为迭代次数

%函数开始

function [ ak2,k ] =Root(ak1,e)

%初始化ak2的值

ak2=(ak1+3/ak1)/2

%初始化k值

k=1;

%迭代算法开始

while abs(ak1-ak2)>e

ak1=ak2;

ak2=(ak1+3/ak1)/2;

k=k+1;

end

%迭代算法结束

end

%函数结束

2.（1）不动点函数RootB

%定义不动点迭代函数RootB三次开根，a0为初始值、e为误差值

%返回值a为近似三次求根数，k为迭代次数

%函数开始

function [ a,k ] =RootB(a0,e)

%根据方程变形，初始化a的值

a=(a0+1)^(1/3);

%初始化k值

k=1;

%不动点迭代算法开始

while abs(a0-a)>e

a0=a;

a=(a0+1)^(1/3);

k=k+1;

end

%不动点迭代算法结束

end

%函数结束

## 2.（2）牛顿迭代法函数RootN

%定义牛顿迭代法迭代函数RootN求三次开根，f为原函数、g为原函数f的导函数,e为误差值

%返回值x为近似三次求根数，k为迭代次数

%函数开始

function [ x,k ] =RootN(f,g,e)

%初始化x0的值

x0=1.5;

%根据牛顿迭代法初始化x的值

x=x0-f(x0)/g(x0);

%初始化k值

k=1;

%牛顿迭代算法开始

while abs(x-x0)>e

x0=x;

x=x-f(x0)/g(x0);

k=k+1;

end

%牛顿迭代算法结束

end

%函数结束

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 项目 | 得分 | 总分 |
| 1 | 实验报告排版(5分) |  |  |
| 2 | 算法思想分析(6分) |  |
| 3 | 源代码(7分) |  |
| 4 | 实验结果及分析(7分) |  |

《数值分析》实验报告三

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 专业 | 数学与应用数学 | 班级 | 000 | 姓名 | 挖掘机大王子 | 学号 | 20150402201 |

**实验项目**

迭代法的练习。

**实验内容**

用雅克比迭代法，高斯—塞德尔迭代法，SOR方法（松弛系数取不同值）解下列方程组，并计较迭代次数，要求误差不超过10-4



**算法设计分析**

**其中**

**其中**

**其中**

**实验测试结果及结果分析**（需要注意松弛系数的不同对结果的影响。）

## 1.雅可比迭代法函数yakb

>>A=[5,2,1;-1,4,2;2,-3,10];%初始化方程组的系数矩阵A

>>b=[12;20;3];%初始化方程组的常数矩阵b

>>x0=[0;0;0];%初始化初始向量x0

>>[x,k]=yakb(A,b,x0,10^(-4))

x =

0.363604836730476

4.328063811408591

1.525717088294118

k =

17

>> A\*x

ans =

11.999868894763679

20.000084585492122

3.000189122176356

## 2.高斯-塞德尔迭代法函数gssdr

>>A=[5,2,1;-1,4,2;2,-3,10];%初始化方程组的系数矩阵A

>>b=[12;20;3];%初始化方程组的常数矩阵b

>>x0=[0;0;0];%初始化初始向量x0

>> [x,k]=gssdr(A,b,x0,10^(-4))

x =

0.363640952793824

4.328062994796051

1.525690707880051

k =

9

>> A\*x

ans =

12.000021461441269

19.999992442150482

3.000000000000002

## 超松驰迭代法函数chaosc

>>A=[5,2,1;-1,4,2;2,-3,10];%初始化方程组的系数矩阵A

>>b=[12;20;3];%初始化方程组的常数矩阵b

>>x0=[0;0;0];%初始化初始向量x0

>> [x,k]=chaosc(A,b,x0,1,10^(-4))

x =

0.363640952793824

4.328062994796051

1.525690707880051

k =

9

>> A\*x

ans =

12.000021461441269

19.999992442150482

3.000000000000002

>> [x,k]=chaosc(A,b,x0,1.1,10^(-4))

x =

0.363638830102960

4.328060364254755

1.525689511906311

k =

12

>> A\*x

ans =

12.000004390930622

19.999981650728682

2.999991686504766

>> [x,k]=chaosc(A,b,x0,1.2,10^(-4))

x =

0.363609874452559

4.328051490440860

1.525694661051442

k =

17

>> A\*x

ans =

11.999847014195957

19.999985409413764

3.000011888096959

>> [x,k]=chaosc(A,b,x0,1.3,10^(-4))

x =

0.363610216439020

4.328044133439322

1.525689993647623

k =

27

>> A\*x

ans =

11.999829342721364

19.999946304613513

2.999987969036303

>> [x,k]=chaosc(A,b,x0,1.4,10^(-4))

x =

0.363669043353591

4.328089130775695

1.525694886651656

k =

91

>> A\*x

ans =

12.000218364970999

20.000077253052503

3.000019560896658

## 运行结果分析

|  |  |  |  |
| --- | --- | --- | --- |
| 误差值e | 雅可比迭代法(k) | 高-赛迭代法(k) | 超松驰迭代法(k) |
| 10^(-4) | 17 | 9 | 9、12、17、27 |

通过比较，可以看出，在误差允许等效的情况下，高斯塞德尔和超松驰（w=1.0）的迭代效果比较好，。显然当w=1时，超松驰迭代法的k值=高斯-塞德尔迭代法k值，说明他们效果一样(A\*x相等)，而雅可比通过A\*x验证后,误差效果大，但是对于超松驰迭代次数越大不一定误差效果好，当选择一个合适的松弛因子之后，误差效果会比较好。

|  |  |
| --- | --- |
| 松弛因子 | 误差e=10^(-4) |
| 1.0 | 9 |
| 1.1 | 12 |
| 1.2 | 17 |
| 1.3 | 27 |
| 1.4 | 91 |

通过上表可以看出松弛因子选择得好，会使迭代收敛次数K大大加速（k=1.0最小）

**实验总结**

1. 函数的实现首先分析原算法的迭代原理
2. 迭代算法函数的实现过程中先理清楚输入参数和希望返回值
3. 各种迭代算法中的叠加部分的算法实现为赋值表达式+通项

**附录 实验程序代码（加注释）**

## 1.雅可比迭代法函数yakb

%定义yakb雅可比迭代函数,A为系数矩阵,b为常数矩阵,x0为初始向量，e为误差值

%返回值x为方程组的近似解，k为迭代次数

%函数开始

function [ x,k ] = yakb( A,b,x0,e )

D=diag(diag(A));%初始化矩阵A对角矩阵为D

D\_ni=inv(D);%初始化矩阵D的逆矩阵为D\_ni

L=-tril(A,-1);%初始化矩阵A的下三角矩阵为L

U=-triu(A,1);%初始化矩阵A的上三角矩阵为U

B=D\_ni\*(L+U);

f=D\_ni\*b;

x=B\*x0+f;

%初始化k

k=1;

%雅可比迭代算法开始

while norm(x-x0,inf)>e

x0=x;

x=B\*x+f;

k=k+1;

end

%雅可比迭代算法结束

end

%函数结束

## 2.高斯-塞德尔迭代法函数gssdr

%定义gssdr高斯-塞德尔迭代函数,A为系数矩阵,b为常数矩阵,x0为初始向量，e为误差值

%返回值x为方程组的近似解，k为迭代次数

%函数开始

function [ x,k ] = gssdr( A,b,x0,e )

D=diag(diag(A));%初始化矩阵A对角矩阵为D

L=-tril(A,-1);%初始化矩阵A的下三角矩阵为L

U=-triu(A,1);%初始化矩阵A的上三角矩阵为U

B=inv(D-L)\*U;

f=inv(D-L)\*b;

x=B\*x0+f;

%初始化k

k=1;

%高斯-塞德尔迭代算法开始

while norm(x-x0,inf)>e

x0=x;

x=B\*x+f;

k=k+1;

end

%高斯-塞德尔迭代算法结束

end

%函数结束

## 3..高斯-塞德尔迭代法函数chaosc

%定义chaosc超松驰尔迭代函数,A为系数矩阵,b为常数矩阵,x0为初始向量，w为超松驰因子(0<w<2)，e为误差值

%返回值x为方程组的近似解，k为迭代次数

%函数开始

function [ x,k ] = chaosc( A,b,x0,w,e )

D=diag(diag(A));%初始化矩阵A对角矩阵为D

L=-tril(A,-1);%初始化矩阵A的下三角矩阵为L

U=-triu(A,1);%初始化矩阵A的上三角矩阵为U

Lw=inv(D-w\*L)\*((1-w)\*D+w\*U);

f=w\*inv(D-w\*L)\*b;

x=Lw\*x0+f;

%初始化k

k=1;

%超松驰迭代算法开始

while norm(x-x0)>e

x0=x;

x=Lw\*x0+f;

k=k+1;

end

%超松驰迭代算法结束

end

%函数结束

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 项目 | 得分 | 总分 |
| 1 | 实验报告排版(5分) |  |  |
| 2 | 算法思想分析(6分) |  |
| 3 | 源代码(7分) |  |
| 4 | 实验结果及分析(7分) |  |

《数值分析》实验报告四

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 专业 | 数学与应用数学 | 班级 | 000 | 姓名 | 挖掘机大王子 | 学号 | 20150402201 |

**实验内容**

用**分量形式**的雅克比迭代法，高斯—塞德尔迭代法，SOR方法（松弛系数取不同值）解下列方程组，并计较迭代次数，要求误差不超过10-4

迭代法的练习。



**算法设计分析**

**表示迭代次数**

****

**为松弛因子**

**实验测试结果及结果分析**

## 1.雅可比迭代法函数yakb1

format long%设定16位有效数字

A=[5,2,1;-1,4,2;2,-3,10];%初始化方程组的系数矩阵A

b=[12;20;3];%初始化方程组的常数矩阵b

x0=[0;0;0];%初始化初始向量x0

>> [x,k]=yakb1(A,b,x0,10^(-4))

x =

0.363643298770479

4.328058676406194

1.525686587955120

k =

19

>> A\*x

ans =

12.000020434619904

19.999964582764534

2.999976447873578

## 2.高斯-塞德尔迭代法函数gssdr1

>> [x,k]=gssdr1(A,b,x0,10^(-4))

x =

0.363640952793823

4.328062994796051

1.525690707880051

k =

9

>> A\*x

ans =

12.000021461441269

19.999992442150482

3.000000000000000

## 超松驰迭代法函数chaosc1

>> [x,k]=chaosc1(A,b,x0,10^(-4),1.0)

x =

0.363640952793823

4.328062994796051

1.525690707880051

k =

9

>> A\*x

ans =

12.000021461441269

19.999992442150482

3.000000000000000

>> [x,k]=chaosc1(A,b,x0,10^(-4),1.1)

x =

0.363638830102960

4.328060364254754

1.525689511906311

k =

12

>>

>> A\*x

ans =

12.000004390930622

19.999981650728678

2.999991686504767

>> [x,k]=chaosc1(A,b,x0,10^(-4),1.2)

x =

0.363646591045529

4.328066882594621

1.525689963672848

k =

18

>> A\*x

ans =

12.000056684089733

20.000000866678651

2.999992171035675

>> [x,k]=chaosc1(A,b,x0,10^(-4),1.3)

x =

0.363624063914123

4.328054926035977

1.525691429193842

k =

29

>> A\*x

ans =

11.999921600836410

19.999978498617466

2.999997641658734

## 运行结果分析

迭代次数分析

|  |  |  |  |
| --- | --- | --- | --- |
| 允许误差e | 雅可比k | 高斯塞德尔k | 超松驰k |
| 10^(-4) | 19 | 9 | 9,12,18,29 |

上表说明了高斯塞德尔和超松驰迭代次数有优势比雅可比有优势，通过超松弛因子测试说明了要选择合适的超松驰因子才能达到比较接近理论真实值理想结果。

近似值解分析A\*x分析，以下误差均为10^(-4)

|  |  |  |  |
| --- | --- | --- | --- |
| A\*x | 雅可比 | 高斯塞德尔 | 超松驰w=1.0 |
| b2 | 12.000020434619904 | 12.000021461441269 | 12.000021461441269 |
| b3 | 19.999964582764534 | 19.999992442150482 | 19.999992442150482 |
| b4 | 2.999976447873578 | 3.000000000000000 | 3.000000000000000 |

通过与线性方程组的真实值12，20，3比较之后，可以得出，高斯塞德尔和超松驰比较接近理论真实值，再仔细观察，当w=1.0时，偶然发现超松驰的近似解等于高斯塞德尔的近似解。

**实验总结**

1. 通过本实验，掌握了解线性方程组的分量形式迭代算法的实现
2. 实验过程中的累加符号的算法实现特别注意x同上标形势的叠加。
3. 实验结果分析要跟理论真实值进行比较才能得出靠谱结论

**附录 实验程序代码（加注释）**

## 1.雅可比迭代法函数yakb1

%定义yakb1雅可比迭代函数,A为系数矩阵,b为常数矩阵,x0为初始向量，e为误差值

%返回值x为方程组的近似解，k为迭代次数

%函数开始

function [ x,k ] = yakb1( A,b,x0,e )

%计算向量b的长度

n=length(b);

%初始化r

r=A\*x0-b;

%声明x是一个列向量,保证迭代过程中维数一致。

x=zeros(n,1);

%初始化迭代次数k

k=0;

%循环开始

while norm(r,inf)>e

%开始计算迭代次数K

k=k+1;

%雅可比分量迭代开始

for i=1:n

%开始分量公式右边s1

s1=0;

for j=1:i-1

s1=s1+A(i,j)\*x0(j);

end

%完成分量公式右边s1

%开始分量公式右边s2

s2=0;

for j=i+1:n

s2=s2+A(i,j)\*x0(j);

end

%完成分量公式右边s2

%开始计算xi

x(i)=1/A(i,i)\*(-s1-s2+b(i));

end

%重新计算r

r=A\*x-b;

%替换x0

x0=x;

end

%循环结束

end

%函数结束

## 高斯-塞德尔迭代法函数gssdr1

%定义gssdr1高斯-赛德尔迭代函数,A为系数矩阵,b为常数矩阵,x0为初始向量，e为误差值

%返回值x为方程组的近似解，k为迭代次数

%函数开始

function [ x,k ] = gssdr1( A,b,x0,e )

%计算向量b的长度

n=length(b);

%初始化r

r=A\*x0-b;

%声明x是一个列向量,保证迭代过程中维数一致。

x=zeros(n,1);

%初始化迭代次数k

k=0;

%循环开始

while norm(r,inf)>e

%开始计算迭代次数K

k=k+1;

%雅可比分量迭代开始

for i=1:n

%开始分量公式右边s1

s1=0;

for j=1:i-1

s1=s1+A(i,j)\*x(j);

end

%完成分量公式右边s1

%开始分量公式右边s2

s2=0;

for j=i+1:n

s2=s2+A(i,j)\*x0(j);

end

%完成分量公式右边s2

%开始计算xi

x(i)=1/A(i,i)\*(-s1-s2+b(i));

end

%重新计算r

r=A\*x-b;

%替换x0

x0=x;

end

%循环结束

end

%函数结束

## 超松驰迭代法函数chaosc1

%定义chaosc1超松弛迭代函数,A为系数矩阵,b为常数矩阵,x0为初始向量，e为误差值,w松弛系数

%返回值x为方程组的近似解，k为迭代次数

%函数开始

function [ x,k ] = chaosc1( A,b,x0,e,w )

%计算向量b的长度

n=length(b);

%初始化r

r=A\*x0-b;

%初始化迭代次数k

k=0;

%循环开始

while norm(r,inf)>e

%开始计算迭代次数K

k=k+1;

%雅可比分量迭代开始

for i=1:n

%开始分量公式右边s1

s1=0;

for j=1:i-1

s1=s1+A(i,j)\*x(j);

end

%完成分量公式右边s1

%开始分量公式右边s2

s2=0;

for j=i+1:n

s2=s2+A(i,j)\*x0(j);

end

%完成分量公式右边s2

%开始计算xi

x(i,1)=(1-w)\*x0(i)+w./A(i,i)\*(b(i)-s1-s2);

end

%重新计算r

r=A\*x-b;

%替换x0

x0=x;

end

%循环结束

end

%函数结束

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 项目 | 得分 | 总分 |
| 1 | 实验报告排版(5分) |  |  |
| 2 | 算法思想分析(6分) |  |
| 3 | 源代码(7分) |  |
| 4 | 实验结果及分析(7分) |  |

《数值分析》实验报告五

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 专业 | 数学与应用数学 | 班级 | 000 | 姓名 | 挖掘机大王子 | 学号 | 20150402201 |

**实验项目**

递推公式编写程序

**实验内容**

1 运用幂法和反幂法求解任一矩阵的最大模特征值和最小模特征值

2练习MATLAB中插值与拟合的命令

**算法设计分析**

**实验测试结果及结果分析**

## 1.幂法\*(1)

>> A=[1,1,1/2;1,1,1/4;1/2,1/4,2];

>> v0=[1;1;1];

>> [maxvk,uk,k] = FunctionPower( A,v0,10^(-4))

k =

3

uk =

0.7990

0.7030

1.0000

k =

4

uk =

0.7774

0.6803

1.0000

k =

5

uk =

0.7651

0.6674

1.0000

k =

6

uk =

0.7580

0.6600

1.0000

k =

7

uk =

0.7539

0.6557

1.0000

k =

8

uk =

0.7515

0.6532

1.0000

k =

9

uk =

0.7502

0.6517

1.0000

k =

10

uk =

0.7494

0.6508

1.0000

k =

11

uk =

0.7489

0.6504

1.0000

k =

12

uk =

0.7486

0.6501

1.0000

k =

13

uk =

0.7484

0.6499

1.0000

k =

14

uk =

0.7484

0.6498

1.0000

k =

15

uk =

0.7483

0.6497

1.0000

maxvk =

2.5366

uk =

0.7483

0.6497

1.0000

k =

15

## 2.反幂法

>> [maxvk,uk,k] = FunctionPowerr( A,v0,10^(-4))

k =

3

maxvk =

0.0166

uk =

1.0000

-0.9516

-0.1300

k =

4

maxvk =

0.0166

uk =

-1.0000

0.9517

0.1300

k =

5

maxvk =

0.0166

uk =

1.0000

-0.9517

-0.1300

k =

6

maxvk =

0.0166

uk =

-1.0000

0.9517

0.1300

maxvk =

0.0166

uk =

-1.0000

0.9517

0.1300

k =

6

## 3.拟合和插值

>>%下面开始练习插值与拟合的命令

>>%定义数据x和y,x-->y

>>x=1:1:5;

>>y=[4 4.5 6 8 8.5];

>>%开始拟合

>>A=polyfit(x,y,2)

A =

0.0357 1.0357 2.7000

>>%开始插值

>> b=polyval(A,5.3)

b =

9.1925

>>[b,r]=polyfit(x,y,2)

b =

0.0357 1.0357 2.7000

r =

struct with fields:

R: [3×3 double]

df: 2

normr: 0.8106

## 运行结果分析

## 1.幂法

以下结果在允许误差e=10^(-4)范围内

|  |  |  |
| --- | --- | --- |
| K（迭代次数） | uk（特征向量） | maxvk（特征值） |
| 3 | (0.7990,0.7030,1.0000) | 2.6047 |
| 4 | (2.5753,0.6497,0.6497) | 2.5753 |
| 7 | (0.7539,0.6557,1.0000) | 2.5440 |
| 10 | (0.7494,0.6508,1.0000) | 2.5380 |
| 12 | (0.7486,0.6501,1.0000) | 2.5370 |
| 14 | (0.7484,0.6498,1.0000) | 2.5367 |
| 15 | (0.7483,0.6497,1.0000) | 2.5366 |

从上表可以看出随着迭代次数K的增多，特征向量亦或是特征值趋近一个稳定的数值范围波动，直到满足误差允许的范围内，迭代才停止，这时的特征值是最接近真实特征解。

## 2.反幂法

以下结果在允许误差e=10^(-4)范围内

|  |  |  |
| --- | --- | --- |
| K（迭代次数） | uk（特征向量） | maxvk（特征值） |
| 3 | (1.0000,-0.9516,-0.1300) | 0.0166 |
| 4 | (-1.0000,0.9517,0.1300) | 0.0166 |
| 5 | (1.0000,-0.9517,-0.1300) | 0.0166 |
| 6 | (-1.0000,0.9517,0.1300) | 0.0166 |

从上表可以看出，随着迭代次数的增多，特征值在一个范围内波动小。

## 3.拟合和插值

b =

0.0357 1.0357 2.7000

r =

struct with fields:

R: [3×3 double]

df: 2

normr: 0.8106

b的三个数字分别对应二次项，一次项和常数项，通过返回值r而且可以看出精确度以及范数。

**实验总结**

1. 幂法迭代法的迭代过程（循环里面）中注意递推公式中的相邻两项的变换。
2. 要数值分析，最好是把数据可视化后进行对比。

**附录 实验程序代码（加注释）**

**1.（1）幂法迭代法**

%定义幂法函数FunctionPower,传入参数A矩阵,v0为初始向量，e为误差数

%返回值,maxvk为矩阵的近似特征值，uk为规范化向量，K为迭代次数

%函数开始

function [maxvk,uk,k] = FunctionPower( A,v0,e)

%计算v1

v1=A\*v0;

miu1=norm(v1,inf);

%计算u1

u1=v1/miu1;

%计算v2

v2=A\*u1;

miu2=norm(v2,inf);

%计算u2

u2=v2/miu2;

%前面已经迭代2次，故初始化K的值=2

k=2;

while abs(miu1-miu2)>e

v1=v2;

u1=u2;

miu1=miu2;

v2=A\*u1;

miu2=norm(v2,inf);

%迭代次数累加

k=k+1;

u2=v2/miu2;

%便于观察终端显示K，maxvk,uk

k

maxvk=miu2

uk=u2

end

end

%函数结束

**1（2）.反幂法迭代法**

%定义反幂法函数FunctionPowerr,传入参数A矩阵,v0为初始向量，e为误差数

%返回值,maxvk为矩阵的近似特征值，uk为规范化向量，K为迭代次数

%函数开始

function [maxvk,uk,k] = FunctionPowerr( A,v0,e)

%计算v1

A\_ni=inv(A);

v1=A\_ni\*v0;

miu1=norm(v1,inf);

%计算u1

u1=v1/miu1;

%计算v2

v2=A\_ni\*u1;

miu2=norm(v2,inf);

%计算u2

u2=v2/miu2;

%前面已经迭代2次，故初始化K的值=2

k=2;

while abs(miu1-miu2)>e

v1=v2;

u1=u2;

miu1=miu2;

v2=A\_ni\*u1;

miu2=norm(v2,inf);

%迭代次数累加

k=k+1;

u2=v2/miu2;

%便于观察终端显示K，maxvk,uk

k

maxvk=1/miu2

uk=u2

end

End

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 项目 | 得分 | 总分 |
| 1 | 实验报告排版(5分) |  |  |
| 2 | 算法思想分析(6分) |  |
| 3 | 源代码(7分) |  |
| 4 | 实验结果及分析(7分) |  |

《数值分析》实验报告六

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 专业 | 数学与应用数学 | 班级 | 000 | 姓名 | 挖掘机大王子 | 学号 | 20150402201 |

**实验项目**

递推公式编写程序，作图

**实验内容**

用欧拉方法和ode45方法求解常微分方程

**算法设计分析**

**实验测试结果及结果分析**

## 1.欧拉法

>> f=@(x,y)y-2\*x/y;

>> y0=1;

>> a=0;

>> b=1;

>> n=10;

>> [yy,xx] = FunctionOla( f,y0,a,b,n)

>> [yy,xx] = FunctionOla( f,y0,a,b,n)

xi =

0

y =

1.1000

xi =

0.1000

y =

1.1918

xi =

0.2000

y =

1.2774

xi =

0.3000

y =

1.3582

xi =

0.4000

y =

1.4351

xi =

0.5000

y =

1.5090

xi =

0.6000

y =

1.5803

xi =

0.7000

y =

1.6498

xi =

0.8000

y =

1.7178

xi =

0.9000

y =

1.7848

yy =

1.1000 1.1918 1.2774 1.3582 1.4351 1.5090 1.5803 1.6498 1.7178 1.7848

xx =

0 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000 0.7000 0.8000 0.9000

%下面开始画解的对比图

>> plot(xx,yy,'\*')

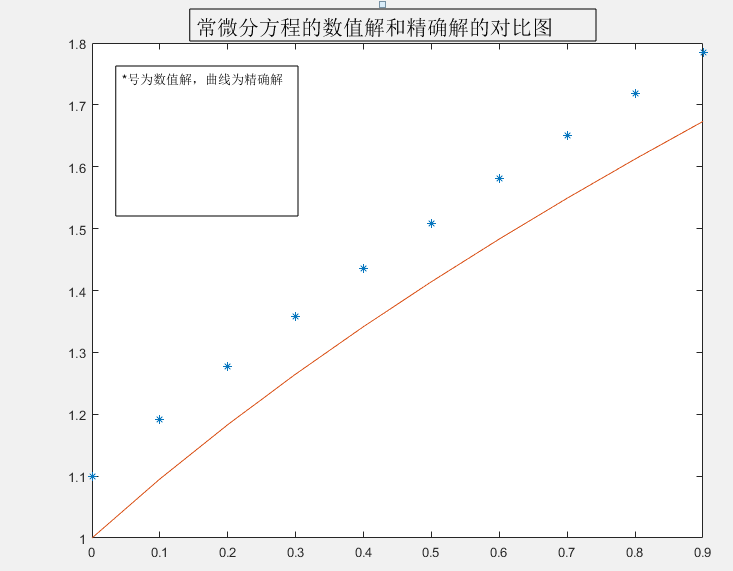
>> hold on

>> x1=0:0.1:0.9;

>> y1=(1+2\*x1).^(1/2);

>> plot(x1,y1)

>> hold off



## 2.ode45

>> odefun=@(x,y)y-2\*x/y;

>> tspan=[0,1];

>> y0=1;

>> [t,y]=ode45(odefun,tspan,y0)

t =

0

0.0250

0.0500

0.0750

0.1000

0.1250

0.1500

0.1750

0.2000

0.2250

0.2500

0.2750

0.3000

0.3250

0.3500

0.3750

0.4000

0.4250

0.4500

0.4750

0.5000

0.5250

0.5500

0.5750

0.6000

0.6250

0.6500

0.6750

0.7000

0.7250

0.7500

0.7750

0.8000

0.8250

0.8500

0.8750

0.9000

0.9250

0.9500

0.9750

1.0000

y =

1.0000

1.0247

1.0488

1.0724

1.0954

1.1180

1.1402

1.1619

1.1832

1.2042

1.2247

1.2450

1.2649

1.2845

1.3038

1.3229

1.3416

1.3601

1.3784

1.3964

1.4142

1.4318

1.4491

1.4663

1.4832

1.5000

1.5166

1.5330

1.5492

1.5652

1.5811

1.5969

1.6125

1.6279

1.6432

1.6583

1.6733

1.6882

1.7029

1.7176

1.7321

%下面开始画oder45和精确解的对比图。

>> plot(t,y,'\*')

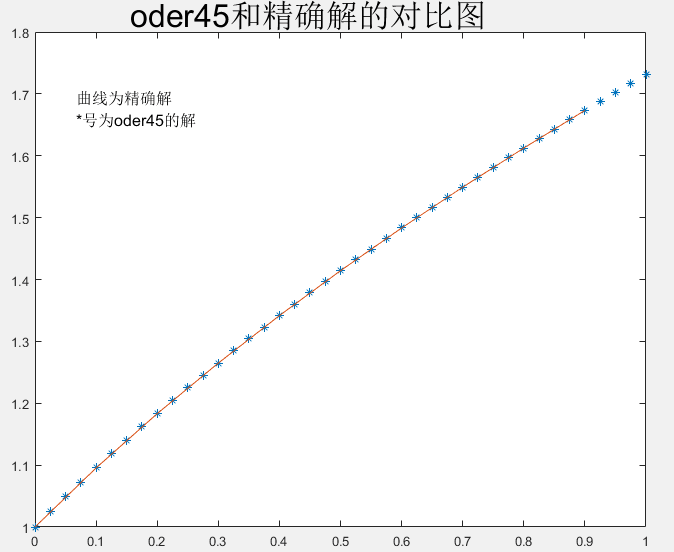
>> hold on

>> x1=0:0.1:0.9;

>> y1=(1+2\*x1).^(1/2);

>> plot(x1,y1)

>> hold off



## 运行结果分析

## 1.欧拉法

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| xi | y | 准确值 | xi | y | 准确值 |
| 0.1000 | 1.1918 | 1.0954 | 0.6000 | 1.5803 | 1.4832 |
| 0.2000 | 1.2774 | 1.1832 | 0.7000 | 1.6498 | 1.5492 |
| 0.3000 | 1.3582 | 1.2649 | 0.8000 | 1.7178 |  |
| 0.4000 | 1.4351 | 1.3416 | 0.9000 | 1.7848 |  |
| 0.5000 | 1.5090 | 1.4142 |  |  |  |

根据解析式算出准确值

p=(1+2\*xx).^(1/2)

p =

1.0000 1.0954 1.1832 1.2649 1.3416 1.4142 1.4832 1.5492 1.6125 1.6733

两者相比比较可以看出欧拉方法的精确度很差。而且通过图像可视化观察，欧拉方法的解是偏离精确解的。

## oder45法

y =

1.0000

1.0247

1.0488

1.0724

1.0954

1.1180

1.1402

1.1619

1.1832

1.2042

1.2247

1.2450

1.2649

1.2845

1.3038

1.3229

1.3416

1.3601

1.3784

1.3964

1.4142

1.4318

1.4491

1.4663

1.4832

1.5000

1.5166

1.5330

1.5492

1.5652

1.5811

1.5969

1.6125

1.6279

1.6432

1.6583

1.6733

1.6882

1.7029

1.7176

1.7321

跟精准真实值相比差距不大。明显比上一个欧拉法改善了精度。而且通过图像对比oder45的解基本落在了精确解的图像上。

**实验总结**

1.欧拉法编写函数时候，要注意观察结果（有必要时候，显示迭代的过程）

2.如何使用oder45?终端输入helep oder45寻求帮助。

**附录 实验程序代码（加注释）**

**1..欧拉**

%定义欧拉法函数FunctionOla,传入参数f为微积分函数,y0为初始值，a和b分别是区间上下限

%返回值,yy为所有解的向量集合

%函数开始

function [yy,xx] = FunctionOla( f,y0,a,b,n)

y=y0;

h=(b-a)/n;

%定义一个空向量，存储每次的y值，xx储存xi的值，为了方便画图

yy=[];

xx=[];

for i=0:n-1

%以下为了实验结果观察，故不加分号。

xi=a+i\*h

xx=[xx,xi];

y=y+h\*f(xi,y)

%将y追加到向量yy

yy=[yy,y];

end

%函数结束

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 项目 | 得分 | 总分 |
| 1 | 实验报告排版(5分) |  |  |
| 2 | 算法思想分析(6分) |  |
| 3 | 源代码(7分) |  |
| 4 | 实验结果及分析(7分) |  |

《数值分析》实验报告七

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 专业 | 数学与应用数学 | 班级 | 000 | 姓名 | 挖掘机大王子 | 学号 | 20150402201 |

**实验项目**

高次插值的龙格现象

**实验内容**

用拉格朗日插值分别编写程序，以函数在区间中所有整数点为插值节点。

（1）画出原函数与插值节点的图像，

（2）画出插值多项式的图像，并且观察龙格现象。

**算法设计分析**

**实验测试结果及结果分析**

1. 画出原函数与插值节点的图像

以下命令在脚本内运行

xx =-5:0.01:5;

yy = 1./(1+xx.^2);

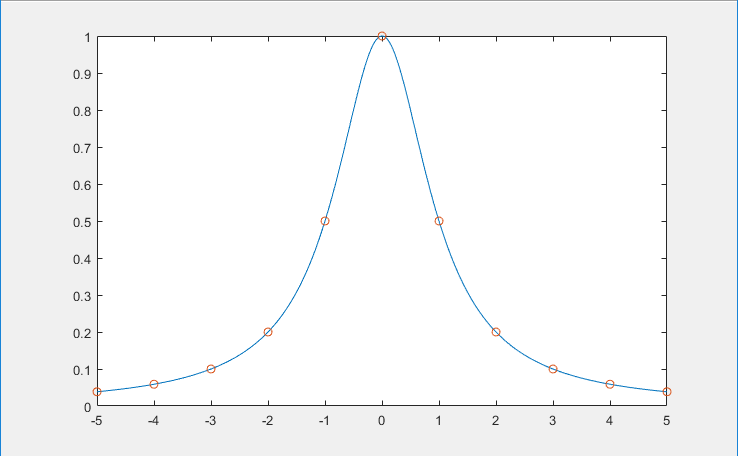
plot(xx,yy)

hold on

xi= -5:5;

yi=1./(1+xi.^2);

plot(xi,yi,'o')

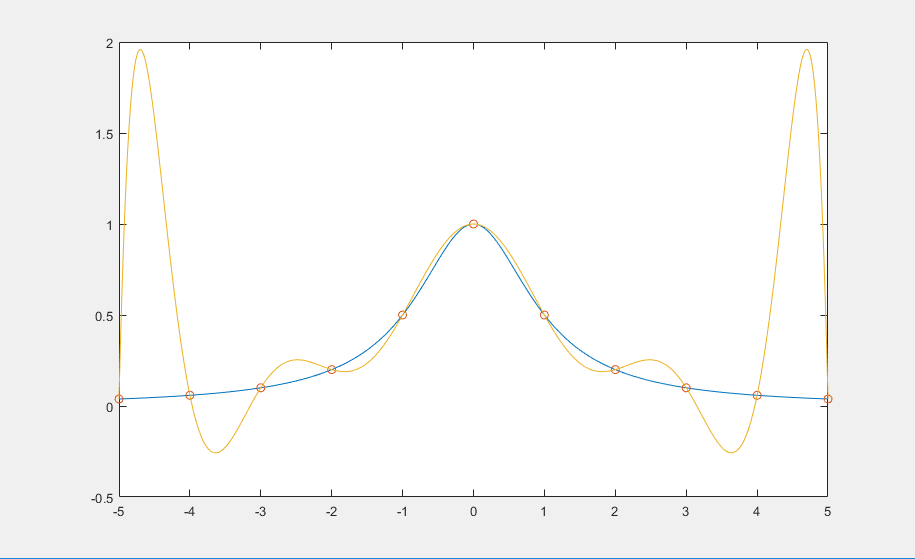


# 画出插值多项式的图像，并且观察龙格现象。

往（1）内添加以下内容

y=lagrang(xi,yi,x);

plot(x,y)



## 运行结果分析

(1)原函数与插值节点的图像

从图中可以看出，每个整数解都落在了原函数图像上。

# 插值多项式的图像

通过图像观察，首先，两个图像交点部分，一共有十一个交点。从整体上看，两图象的交点维度是往中间靠拢的，两个图形都是对称的，绿色的是原函数，但是从-2到2部分之后，插值函数越来越偏离原函数。

**实验总结**

1. 通过本实验掌握了画图的基本操作
2. 拉格朗日插值函数，要实现多次插值可以在外层再加一个循环，并且记录每次插值的结果追加到一个空向量。
3. 通过作图可以观察到一些图像能直观的基本规律的过程。

**附录 实验程序代码（加注释）**

1. **脚本部分**

xx =-5:0.01:5;

yy = 1./(1+xx.^2);

plot(xx,yy)

hold on

xi= -5:5;

yi=1./(1+xi.^2);

plot(xi,yi,'o')

y=lagrang(xi,yi,x);

plot(x,y)

**(2)拉格朗日函数**

%定义拉格朗日插值函数langrange,传入参数xi,和yi分别对应函数区间和函数，x为插值可以一个，或者多个，多个用向量表示。

%返回值,y是每次插值x返回的对应的值。多个返回多个，以向量的形式返回。

%函数开始

function [yy]=lagrange(xi,yi,x)

%取向量xi的个数

n=length(xi);

%取向量x的个数,目的是计算多次插值

m=length(x);

%插值累加算法开始

yy=[];

for i=1:m

%取i次插值xi，以便于多次插值。

z=x(i);

%定义并初始化s

s=0;

for k=1:n

%连乘号部分开始,先初始化ji

ji=1.0;

for j=1:n

%注意判断j是否等于k,如果不相等则进入循环,相等的则不进入连乘，

%如果j=k,那么p始终等于0

if j~=k

%连乘号公式部分

ji=ji\*(x(i)-xi(j))/(xi(k)-xi(j));

end

end

s=ji\*yi(k)+s;

end

%二层循环结束后把插值得到的结果追加到向量yy

yy=[yy,s];

end

%插值累加算法结束

end

%函数结束

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 项目 | 得分 | 总分 |
| 1 | 实验报告排版(5分) |  |  |
| 2 | 算法思想分析(6分) |  |
| 3 | 源代码(7分) |  |
| 4 | 实验结果及分析(7分) |  |

《数值分析》实验报告八

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 专业 | 数学与应用数学 | 班级 | 000 | 姓名 | 挖掘机大王子 | 学号 | 20150402201 |

**实验项目**

高次插值的龙格现象

**实验内容**

用牛顿插值编写程序，以函数在区间中所有整数点为插值节点。

（1）画出原函数与插值节点的图像，

（2）画出插值多项式的图像，并且观察龙格现象。

**算法设计分析**

**实验测试结果及结果分析**

（1）画出原函数与插值节点的图像

以下命令在脚本内运行

%第一小题开始

xx =-5:0.01:5;

yy = 1./(1+xx.^2);

plot(xx,yy)

hold on

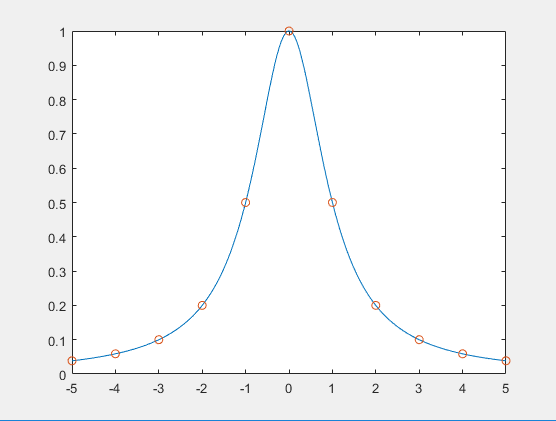
xi= -5:5;

yi=1./(1+xi.^2);

%开始画图

plot(xi,yi,'o')

%第一小题结束



# （2）画出插值多项式的图像，并且观察龙格现象。

往（1）内添加以下内容

%第二小题开始

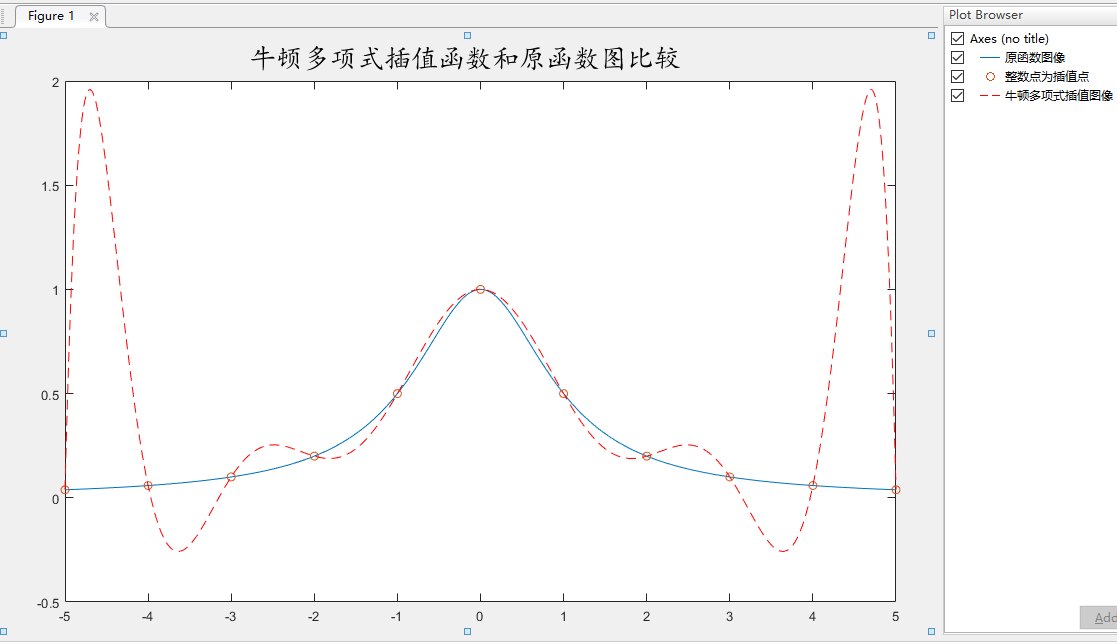
%多次插值初始化

x =-5:0.01:5;

yis=Newton\_in(xi,yi,x);

%开始画图

plot(x,yis)

%第二小题结束

## 运行结果分析

(1)原函数与插值节点的图像

从图中可以看出，每个整数解都落在了原函数图像上。

# 插值多项式的图像

通过图像观察，得到的图像跟拉格朗日插值得到的图像一样。首先，两个图像交点部分，一共有十一个交点。从整体上看，两图象的交点维度是往中间靠拢的，两个图形都是对称的，绿色的是原函数，但是从-2到2部分之后开始，插值函数越来越偏离原函数，所谓的龙格现象。

**实验总结**

1. 拉格朗日只要增加一个节点，每一项都需要重新计算，但是牛顿插值只需要重新计算最后一项就可以了，大大减少了计算量。
2. 通过两个实验可以知道，高次插值，会导致龙格现象。
3. 编写牛顿插值函数的时候，跟拉格朗日插值一样，如果要算多个插值的话，最好是把函数写成能插入多个的，也就是插入一个向量。得到一个结果也是向量。

**附录 实验程序代码（加注释）**

%定义牛顿插值函数Newton\_in,传入参数x,和y分别对应插值节点和差值节点处的函数值，xi为插值，可以是一个，也可以是多个，多个用行向量表示

%返回值,yis为xi出的函数估计值,

%函数开始

function yis=Newton\_in(x,y,xi)

%取迭代次数n

n=length(x);

%取插值的个数m。便于多次插值。

m=length(xi);

%根据插商表的规律开始计算插商Y

%初始化Y为一个零矩阵

Y=zeros(n);

Y(:,1)=y';

%初始化yis为一个空矩阵。

yis=[];

for k=1:n-1

for i=1:n-k

Y(i,k+1)=(Y(i+1,k)-Y(i,k))/(x(i+k)-x(i));

end

end

%计算牛顿插值公式

for q=1:m

%取i次插值xi，以便于多次插值。

z=xi(q);

yi=0;

for i=1:n

%连乘号部分开始,先初始化z

z=1;

for k=1:i-1

z=z\*(xi(q)-x(k));

end

yi=yi+Y(1,i)\*z;

end

%把次插值的结果存入yis,便于画图

yis = [yis,yi];

end

%函数结束

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 项目 | 得分 | 总分 |
| 1 | 实验报告排版(5分) |  |  |
| 2 | 算法思想分析(6分) |  |
| 3 | 源代码(7分) |  |
| 4 | 实验结果及分析(7分) |  |