计 算 方 法

实验指导与实验报告

姓名 卢兑玧

学号 L170300901

院系 计算机科学与技术学院

专业 计算机科学与技术

哈尔滨工业大学

实验报告一

题目： Rung现象产生和克服

摘要：由于高次多项式插值不收敛，会产生Runge现象，本实验在给出具体的实例后，采用分段线性插值和三次样条插值的方法有效的克服了这一现象，而且还取的很好的插值效果。

前言：（目的和意义）

1. 深刻认识多项式插值的缺点。
2. 明确插值的不收敛性怎样克服。
3. 明确精度与节点和插值方法的关系。

数学原理：

在给定*n+*1个节点和相应的函数值以后构造*n*次的Lagrange插值多项式，实验结果表明（见后面的图）这种多项式并不是随着次数的升高对函数的逼近越来越好，这种现象就是Rung现象。

解决Rung现象的方法通常有分段线性插值、三次样条插值等方法。

**分段线性插值：**

设在区间[*a, b*]上，给定n+1个插值节点

*a=x0<x1<…<xn=b*

和相应的函数值*y0*，*y1*，*…*，*yn*，，求作一个插值函数，具有如下性质：

* 1. ，*j=*0，1，…，*n*。
  2. 在每个区间[*xi, xj*]上是线性连续函数。则插值函数称为区间[*a, b*]上对应*n*个数据点的分段线性插值函数。

**三次样条插值：**

给定区间[*a, b*]一个分划

⊿：*a=x0<x1<…<xN=b*

若函数*S(x)*满足下列条件：

1. *S(x)*在每个区间[*xi, xj*]上是不高于3次的多项式。
2. *S(x)*及其2阶导数在[*a, b*]上连续。则称*S(x)*使关于分划⊿的三次样条函数。

程序设计：

本实验采用*Matlab*的*M*文件编写。其中待插值的方程写成*function*的方式，如下

*function y=f(x);*

*y=1/（1+25\*x\*x）;*

写成如上形式即可，下面给出主程序

**Lagrange插值源程序：**

*n=input('将区间分为的等份数输入：\n');*

*s=[-1+2/n\*[0:n]];*%%%给定的定点，Rf为给定的函数

*x=-1:0.01:1;*

*f=0;*

*for q=1:n+1;*

*l=1;*%求插值基函数

*for k=1:n+1;*

*if k~=q;*

*l=l.\*(x-s(k))./(s(q)-s(k));*

*else*

*l=l;*

*end*

*end*

*f=f+Rf(s(q))\*l;*%求插值函数

*end*

*plot(x,f,'r')*%作出插值函数曲线

*grid on*

*hold on*

**分段线性插值源程序**

*clear*

*n=input('将区间分为的等份数输入：\n');*

*s=[-1+2/n\*[0:n]];*%%%给定的定点，Rf为给定的函数

*m=0;*

*hh=0.001;*

*for x=-1:hh:1;*

*ff=0;*

*for k=1:n+1;*%%%求插值基函数

*switch k*

*case 1*

*if x<=s(2);*

*l=(x-s(2))./(s(1)-s(2));*

*else*

*l=0;*

*end*

*case n+1*

*if x>s(n);*

*l=(x-s(n))./(s(n+1)-s(n));*

*else*

*l=0;*

*end*

*otherwise*

*if x>=s(k-1)&x<=s(k);*

*l=(x-s(k-1))./(s(k)-s(k-1));*

*else if x>=s(k)&x<=s(k+1);*

*l=(x-s(k+1))./(s(k)-s(k+1));*

*else*

*l=0;*

*end*

*end*

*end*

*ff=ff+Rf(s(k))\*l;*%%求插值函数值

*end*

*m=m+1;*

*f(m)=ff;*

*end*

%%%作出曲线

*x=-1:hh:1;*

*plot(x,f,'r');*

*grid on*

*hold on*

**三次样条插值源程序：**（采用第一边界条件）

*clear*

*n=input('将区间分为的等份数输入：\n');*

%%%插值区间

*a=-1;*

*b=1;*

*hh=0.001*;%画图的步长

*s=[a+(b-a)/n\*[0:n]];*%%%给定的定点，Rf为给定的函数

%%%%第一边界条件Rf"(-1),Rf"(1)

*v=5000\*1/(1+25\*a\*a)^3-50/(1+25\*a\*a)^4;*

*for k=1:n*;%取出节点间距

*h(k)=s(k+1)-s(k);*

*end*

*for k=1:n-1;*%求出系数向量lamuda,miu

*la(k)=h(k+1)/(h(k+1)+h(k));*

*miu(k)=1-la(k);*

*end*

%%%%赋值系数矩阵A

*for k=1:n-1;*

*for p=1:n-1;*

*switch p*

*case k*

*A(k,p)=2;*

*case k-1*

*A(k,p)=miu(p+1);*

*case k+1*

*A(k,p)=la(p-1);*

*otherwise*

*A(k,p)=0;*

*end*

*end*

*end*

%%%%求出d阵

*for k=1:n-1;*

*switch k*

*case 1*

*d(k)=6\*f2c([s(k) s(k+1) s(k+2)])-miu(k)\*v;*

*case n-1*

*d(k)=6\*f2c([s(k) s(k+1) s(k+2)])-la(k)\*v;*

*otherwise*

*d(k)=6\*f2c([s(k) s(k+1) s(k+2)]);*

*end*

*end*

%%%%求解M阵

*M=A\d';*

*M=[v;M;v];*

%%%%

*m=0;*

*f=0;*

*for x=a:hh:b;*

*if x==a;*

*p=1;*

*else*

*p=ceil((x-s(1))/((b-a)/n));*

*end*

*ff1=0;*

*ff2=0;*

*ff3=0;*

*ff4=0;*

*m=m+1;*

*ff1=1/h(p)\*(s(p+1)-x)^3\*M(p)/6;*

*ff2=1/h(p)\*(x-s(p))^3\*M(p+1)/6;*

*ff3=((Rf(s(p+1))-Rf(s(p)))/h(p)-h(p)\*(M(p+1)-M(p))/6)\*(x-s(p));*

*ff4=Rf(s(p))-M(p)\*h(p)\*h(p)/6;*

*f(m)=ff1+ff2+ff3+ff4 ;*

*end*

%%%作出插值图形

*x=a:hh:b;*

*plot(x,f,'k')*

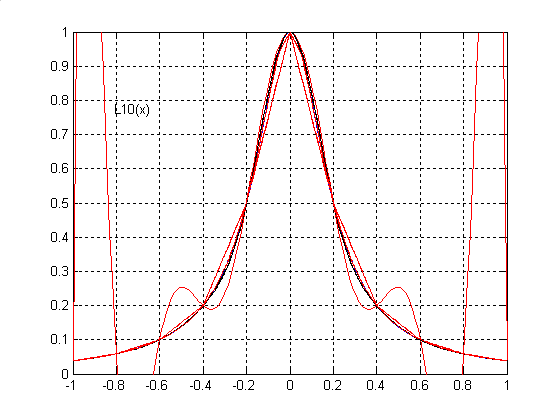
*hold on*

*grid on*

实验结果、结论与讨论

本实验采用函数进行数值插值，插值区间为[-1，1]，给定节点为

*xj=*-1*+jh*,*h=*0.1，*j*=0,…，*n*。下面分别给出Lagrang*e*插值，三次样条插值，线性插值的函数曲线和数据表。图中只标出Lagrang*e*插值的十次多项式的曲线，其它曲线没有标出，从数据表中可以看出具体的误差。



表中，*L10(x)*为Lagrang*e*插值的10次多项式，*S10(x)*，*S40(x)*分别代表*n*=10，40的三次样条插值函数，*X10(x)*，*X40(x)*分别代表*n*=10，40的线性分段插值函数。

*x* *f(x)* *L10(x)* *S10(x) S40(x) X10(x) X40(x)*

-1.00000000000000 0.03846153846154 0.03846153846154 0.03846153846154 0.03846153846154 0.03846153846154 0.03846153846154

-0.95000000000000 0.04244031830239 1.92363114971920 0.04240833151040 0.04244031830239 0.04355203619910 0.04244031830239

-0.90000000000000 0.04705882352941 1.57872099034926 0.04709697585458 0.04705882352941 0.04864253393665 0.04705882352941

-0.85000000000000 0.05245901639344 0.71945912837982 0.05255839923979 0.05245901639344 0.05373303167421 0.05245901639344

-0.80000000000000 0.05882352941176 0.05882352941176 0.05882352941176 0.05882352941176 0.05882352941176 0.05882352941176

-0.75000000000000 0.06639004149378 -0.23146174989674 0.06603986172744 0.06639004149378 0.06911764705882 0.06639004149378

-0.70000000000000 0.07547169811321 -0.22619628906250 0.07482116198866 0.07547169811321 0.07941176470588 0.07547169811321

-0.65000000000000 0.08648648648649 -0.07260420322418 0.08589776360849 0.08648648648649 0.08970588235294 0.08648648648649

-0.60000000000000 0.10000000000000 0.10000000000000 0.10000000000000 0.10000000000000 0.10000000000000 0.10000000000000

-0.55000000000000 0.11678832116788 0.21559187891257 0.11783833017713 0.11678832116788 0.12500000000000 0.11678832116788

-0.50000000000000 0.13793103448276 0.25375545726103 0.14004371555730 0.13793103448276 0.15000000000000 0.13793103448276

-0.45000000000000 0.16494845360825 0.23496854305267 0.16722724315883 0.16494845360825 0.17500000000000 0.16494845360825

-0.40000000000000 0.20000000000000 0.20000000000000 0.20000000000000 0.20000000000000 0.20000000000000 0.20000000000000

-0.35000000000000 0.24615384615385 0.19058046675376 0.24054799403464 0.24615384615385 0.27500000000000 0.24615384615385

-0.30000000000000 0.30769230769231 0.23534659131080 0.29735691695860 0.30769230769231 0.35000000000000 0.30769230769231

-0.25000000000000 0.39024390243902 0.34264123439789 0.38048738140327 0.39024390243902 0.42500000000000 0.39024390243902

-0.20000000000000 0.50000000000000 0.50000000000000 0.50000000000000 0.50000000000000 0.50000000000000 0.50000000000000

-0.15000000000000 0.64000000000000 0.67898957729340 0.65746969368431 0.64000000000000 0.62500000000000 0.64000000000000

-0.10000000000000 0.80000000000000 0.84340742982890 0.82052861660828 0.80000000000000 0.75000000000000 0.80000000000000

-0.05000000000000 0.94117647058824 0.95862704866073 0.94832323122810 0.94117647058824 0.87500000000000 0.94117647058824

0 1.00000000000000 1.00000000000000 1.00000000000000 1.00000000000000 1.00000000000000 1.00000000000000

0.05000000000000 0.94117647058824 0.95862704866073 0.94832323122810 0.94117647058824 0.87500000000000 0.94117647058824

0.10000000000000 0.80000000000000 0.84340742982890 0.82052861660828 0.80000000000000 0.75000000000000 0.80000000000000

0.15000000000000 0.64000000000000 0.67898957729340 0.65746969368431 0.64000000000000 0.62500000000000 0.64000000000000

0.20000000000000 0.50000000000000 0.50000000000000 0.50000000000000 0.50000000000000 0.50000000000000 0.50000000000000

0.25000000000000 0.39024390243902 0.34264123439789 0.38048738140327 0.39024390243902 0.42500000000000 0.39024390243902

0.30000000000000 0.30769230769231 0.23534659131080 0.29735691695860 0.30769230769231 0.35000000000000 0.30769230769231

0.35000000000000 0.24615384615385 0.19058046675376 0.24054799403464 0.24615384615385 0.27500000000000 0.24615384615385

0.40000000000000 0.20000000000000 0.20000000000000 0.20000000000000 0.20000000000000 0.20000000000000 0.20000000000000

0.45000000000000 0.16494845360825 0.23496854305267 0.16722724315883 0.16494845360825 0.17500000000000 0.16494845360825

0.50000000000000 0.13793103448276 0.25375545726103 0.14004371555730 0.13793103448276 0.15000000000000 0.13793103448276

0.55000000000000 0.11678832116788 0.21559187891257 0.11783833017713 0.11678832116788 0.12500000000000 0.11678832116788

0.60000000000000 0.10000000000000 0.10000000000000 0.10000000000000 0.10000000000000 0.10000000000000 0.10000000000000

0.65000000000000 0.08648648648649 -0.07260420322418 0.08589776360849 0.08648648648649 0.08970588235294 0.08648648648649

0.70000000000000 0.07547169811321 -0.22619628906250 0.07482116198866 0.07547169811321 0.07941176470588 0.07547169811321

0.75000000000000 0.06639004149378 -0.23146174989674 0.06603986172744 0.06639004149378 0.06911764705882 0.06639004149378

0.80000000000000 0.05882352941176 0.05882352941176 0.05882352941176 0.05882352941176 0.05882352941176 0.05882352941176

0.85000000000000 0.05245901639344 0.71945912837982 0.05255839923979 0.05245901639344 0.05373303167421 0.05245901639344

0.90000000000000 0.04705882352941 1.57872099034926 0.04709697585458 0.04705882352941 0.04864253393665 0.04705882352941

0.95000000000000 0.04244031830239 1.92363114971920 0.04240833151040 0.04244031830239 0.04355203619910 0.04244031830239

1.00000000000000 0.03846153846154 0.03846153846154 0.03846153846154 0.03846153846154 0.03846153846154 0.03846153846154

从以上结果可以看到，用三次样条插值和线性分段插值，不会出现多项式插值是出现的Runge现象，插值效果明显提高。进一步说，为了提高插值精度，用三次样条插值和线性分段插值是可以增加插值节点的办法来满足要求，而用多项式插值函数时，节点数的增加必然会使多项式的次数增加，这样会引起数值不稳定，所以说这两种插值要比多项式插值好的多。而且在给定节点数的条件下，三次样条插值的精度要优于线性分段插值，曲线的光滑性也要好一些。

1. 拉格朗日插值多项式的次数越大越好吗？

解问题时用的程序为如下；

function TestLag(fx, a, b, n, xi)

% TestLag 实验题目1 1,2

%

% Synopsis: TestLag(fun, a, b, n, xi)

%

% Input: fx = 用来验证插值计算准确率的函数

% a,b = 节点选取上下限

% n = 多项式次数，固定区间[-a,b]分段数

% xi = 要计算插值的点

x = linspace(a, b, n);

y = feval(fx, x);

yi = LagInterp(x, y, xi);

yFact = feval(fx, xi);

err = yFact - yi;

fprintf('将区间[%d,%d]分为了%d段\n', a, b, n);

fprintf('计算插值的点 xi =\n');

disp(xi);

fprintf('计算出的插值 yi =\n');

disp(yi);

fprintf('插值点处函数值 yFact =\n');

disp(yFact);

fprintf('计算误差 err =\n');

disp(err);

代入数据时，可以导出来问题1-（1），1-（2）的答案

(1)

N = 5时，程序运行如下：

TestLag(inline('1./(1+x.^2)'), -5, 5, 5, 0.75:4.75);

将区间[-5,5]分为了5段

计算插值的点 xi =

0.7500 1.7500 2.7500 3.7500 4.7500

计算出的插值 yi =

0.9054 0.5258 0.0096 -0.3568 -0.1595

插值点处函数值 yFact =

0.6400 0.2462 0.1168 0.0664 0.0424

计算误差 err =

-0.2654 -0.2796 0.1072 0.4232 0.2020

N = 10时，程序运行如下：

TestLag(inline('1./(1+x.^2)'), -5, 5, 10, 0.75:4.75);

将区间[-5,5]分为了10段

计算插值的点 xi =

0.7500 1.7500 2.7500 3.7500 4.7500

计算出的插值 yi =

0.6907 0.2330 0.1122 0.1084 -0.2360

插值点处函数值 yFact =

0.6400 0.2462 0.1168 0.0664 0.0424

计算误差 err =

-0.0507 0.0132 0.0045 -0.0420 0.2785

N = 20时，程序运行如下：

TestLag(inline('1./(1+x.^2)'), -5, 5, 20, 0.75:4.75);

将区间[-5,5]分为了20段

计算插值的点 xi =

0.7500 1.7500 2.7500 3.7500 4.7500

计算出的插值 yi =

0.6413 0.2491 0.1282 0.1903 6.4150

插值点处函数值 yFact =

0.6400 0.2462 0.1168 0.0664 0.0424

计算误差 err =

-0.0013 -0.0029 -0.0114 -0.1239 -6.3726

(2)

N = 5时，程序运行如下：

TestLag(inline('exp(x)'), -1, 1, 5, [-0.95 -0.05 0.05 0.95]);

将区间[-1,1]分为了5段

计算插值的点 xi =

-0.9500 -0.0500 0.0500 0.9500

计算出的插值 yi =

0.3863 0.9513 1.0512 2.5863

插值点处函数值 yFact =

0.3867 0.9512 1.0513 2.5857

计算误差 err =

1.0e-003 \*

0.4471 -0.1051 0.1069 -0.6129

N = 10时，程序运行如下：

TestLag(inline('exp(x)'), -1, 1, 10, [-0.95 -0.05 0.05 0.95]);

将区间[-1,1]分为了10段

计算插值的点 xi =

-0.9500 -0.0500 0.0500 0.9500

计算出的插值 yi =

0.3867 0.9512 1.0513 2.5857

插值点处函数值 yFact =

0.3867 0.9512 1.0513 2.5857

计算误差 err =

1.0e-008 \*

-0.3126 -0.0055 -0.0055 -0.3714

N = 20时，程序运行如下：

TestLag(inline('exp(x)'), -1, 1, 20, [-0.95 -0.05 0.05 0.95]);

将区间[-1,1]分为了20段

计算插值的点 xi =

-0.9500 -0.0500 0.0500 0.9500

计算出的插值 yi =

0.3867 0.9512 1.0513 2.5857

插值点处函数值 yFact =

0.3867 0.9512 1.0513 2.5857

计算误差 err =

1.0e-012 \*

0.7339 0 -0.0002 -0.5671

1. 插值区间越小越好吗？

解本问题时用的程序函数是跟问题一用的程序一样

实验结果为如下;

(1)

N = 5时，程序运行如下：

TestLag(inline('1./(1+x.^2)'), -1, 1, 5, [-0.95 -0.05 0.05 0.95]);

将区间[-1,1]分为了5段

计算插值的点 xi =

-0.9500 -0.0500 0.0500 0.9500

计算出的插值 yi =

0.5136 0.9978 0.9978 0.5136

插值点处函数值 yFact =

0.5256 0.9975 0.9975 0.5256

计算误差 err =

0.0121 -0.0002 -0.0002 0.0121

N = 10时，程序运行如下：

TestLag(inline('1./(1+x.^2)'), -1, 1, 10, [-0.95 -0.05 0.05 0.95]);

将区间[-1,1]分为了10段

计算插值的点 xi =

-0.9500 -0.0500 0.0500 0.9500

计算出的插值 yi =

0.5243 0.9975 0.9975 0.5243

插值点处函数值 yFact =

0.5256 0.9975 0.9975 0.5256

计算误差 err =

0.0014 0.0000 0.0000 0.0014

N = 20时，程序运行如下：

TestLag(inline('1./(1+x.^2)'), -1, 1, 20, [-0.95 -0.05 0.05 0.95]);

将区间[-1,1]分为了20段

计算插值的点 xi =

-0.9500 -0.0500 0.0500 0.9500

计算出的插值 yi =

0.5256 0.9975 0.9975 0.5256

插值点处函数值 yFact =

0.5256 0.9975 0.9975 0.5256

计算误差 err =

1.0e-005 \*

-0.7023 0.0000 0.0000 -0.7023

(2)

N = 5时，程序运行如下：

TestLag(inline('exp(x)'), -5, 5, 5, [-4.75 -0.25 0.25 4.75]);

将区间[-5,5]分为了5段

计算插值的点 xi =

-4.7500 -0.2500 0.2500 4.7500

计算出的插值 yi =

-1.9321 1.4275 0.5882 123.7146

插值点处函数值 yFact =

0.0087 0.7788 1.2840 115.5843

计算误差 err =

1.9408 -0.6487 0.6958 -8.1303

N = 10时，程序运行如下：

TestLag(inline('exp(x)'), -5, 5, 10, [-4.75 -0.25 0.25 4.75]);

将区间[-5,5]分为了10段

计算插值的点 xi =

-4.7500 -0.2500 0.2500 4.7500

计算出的插值 yi =

0.0425 0.7796 1.2848 115.6630

插值点处函数值 yFact =

0.0087 0.7788 1.2840 115.5843

计算误差 err =

-0.0339 -0.0008 -0.0008 -0.0788

N = 20时，程序运行如下：

TestLag(inline('exp(x)'), -5, 5, 20, [-4.75 -0.25 0.25 4.75]);

将区间[-5,5]分为了20段

计算插值的点 xi =

-4.7500 -0.2500 0.2500 4.7500

计算出的插值 yi =

0.0087 0.7788 1.2840 115.5843

插值点处函数值 yFact =

0.0087 0.7788 1.2840 115.5843

计算误差 err =

1.0e-007 \*

-0.0914 0.0000 0.0000 -0.1434

1. 在区间[1,1]考虑拉格朗日插值问题，为了使得插值误差较小，应如何选取插值节点？

解问题时用的程序为如下；

function TestLag2(fx, a, b, n, xi)

% TestLag2 实验题目1 3

%

% Synopsis: TestLag2(fun, a, b, n, xi)

%

% Input: fx = 用来验证插值计算准确率的函数

% a,b = 节点选取上下限

% n = 多项式次数，固定区间[-a,b]分段数

% xi = 要计算插值的点

x = zeros(1,n);

for k = 1:n

x(k) = cos( (2\*k-1)\*pi/(2\*n) ); %构造非等距节点

end

y = feval(fx, x);

yi = LagInterp(x, y, xi);

yFact = feval(fx, xi);

err = yFact - yi;

fprintf('将区间[%d,%d]分为了%d段\n', a, b, n);

fprintf('计算插值的点 xi =\n');

disp(xi);

fprintf('计算出的插值 yi =\n');

disp(yi);

fprintf('插值点处函数值 yFact =\n');

disp(yFact);

fprintf('计算误差 err =\n');

disp(err);

代入数据时，可以得到问题3的答案；

(1)

N =5时，程序运行如下：

TestLag2(inline('1./(1+x.^2)'), -1, 1, 5, [-0.95 -0.05 0.05 0.95]);

将区间[-1,1]分为了5段

计算插值的点 xi =

-0.9500 -0.0500 0.0500 0.9500

计算出的插值 yi =

0.5254 0.9978 0.9978 0.5254

插值点处函数值 yFact =

0.5256 0.9975 0.9975 0.5256

计算误差 err =

1.0e-003 \*

0.2071 -0.3011 -0.3011 0.2071

N =10时，程序运行如下：

TestLag2(inline('1./(1+x.^2)'), -1, 1, 10, [-0.95 -0.05 0.05 0.95]);

将区间[-1,1]分为了10段

计算插值的点 xi =

-0.9500 -0.0500 0.0500 0.9500

计算出的插值 yi =

0.5255 0.9972 0.9972 0.5255

插值点处函数值 yFact =

0.5256 0.9975 0.9975 0.5256

计算误差 err =

1.0e-003 \*

0.1562 0.2603 0.2603 0.1562

N =20时，程序运行如下：

TestLag2(inline('1./(1+x.^2)'), -1, 1, 20, [-0.95 -0.05 0.05 0.95]);

将区间[-1,1]分为了20段

计算插值的点 xi =

-0.9500 -0.0500 0.0500 0.9500

计算出的插值 yi =

0.5256 0.9975 0.9975 0.5256

插值点处函数值 yFact =

0.5256 0.9975 0.9975 0.5256

计算误差 err =

1.0e-007 \*

0.2318 0.2381 0.2381 0.2318

(2)

N =5时，程序运行如下：

TestLag2(inline('exp(x)'), -1, 1, 5, [-0.95 -0.05 0.05 0.95]);

将区间[-1,1]分为了5段

计算插值的点 xi =

-0.9500 -0.0500 0.0500 0.9500

计算出的插值 yi =

0.3867 0.9514 1.0511 2.5857

插值点处函数值 yFact =

0.3867 0.9512 1.0513 2.5857

计算误差 err =

1.0e-003 \*

0.0079 -0.1317 0.1339 -0.0108

N =10时，程序运行如下：

TestLag2(inline('exp(x)'), -1, 1, 10, [-0.95 -0.05 0.05 0.95]);

将区间[-1,1]分为了10段

计算插值的点 xi =

-0.9500 -0.0500 0.0500 0.9500

计算出的插值 yi =

0.3867 0.9512 1.0513 2.5857

插值点处函数值 yFact =

0.3867 0.9512 1.0513 2.5857

计算误差 err =

1.0e-009 \*

-0.5045 -0.4791 -0.4835 -0.5994

N =20时，程序运行如下：

TestLag2(inline('exp(x)'), -1, 1, 20, [-0.95 -0.05 0.05 0.95]);

将区间[-1,1]分为了20段

计算插值的点 xi =

-0.9500 -0.0500 0.0500 0.9500

计算出的插值 yi =

0.3867 0.9512 1.0513 2.5857

插值点处函数值 yFact =

0.3867 0.9512 1.0513 2.5857

计算误差 err =

1.0e-015 \*

0.1665 0.3331 -0.4441 -0.8882

1. 考虑拉格朗日插值问题，内插比外推更可靠吗？

解问题时用的程序为如下；

function TestLag3(x, xi)

% TestLag3 实验题目1 4

%

% Synopsis: TestLag3(fun, a, b, n, xi)

%

% Input: x = 构造Lagrange插值的节点

% xi = 要计算插值的点

fx = inline('sqrt(x)');

y = feval(fx, x);

yi = LagInterp(x, y, xi);

yFact = feval(fx, xi);

err = yFact - yi;

fprintf('计算插值的点 xi =\n');

disp(xi);

fprintf('计算出的插值 yi =\n');

disp(yi);

fprintf('插值点处函数值 yFact =\n');

disp(yFact);

fprintf('计算误差 err =\n');

disp(err);

实验结果为如下;

1. 程序运行如下：

TestLag3([1 4 9], [5 50 115 185])

计算插值的点 xi =

5 50 115 185

计算出的插值 yi =

2.2667 -20.2333 -171.9000 -492.7333

插值点处函数值 yFact =

2.2361 7.0711 10.7238 13.6015

计算误差 err =

-0.0306 27.3044 182.6238 506.3348

1. 程序运行如下：

TestLag3([36 49 64], [5 50 115 185])

计算插值的点 xi =

5 50 115 185

计算出的插值 yi =

3.1158 7.0718 10.1670 10.0388

插值点处函数值 yFact =

2.2361 7.0711 10.7238 13.6015

计算误差 err =

-0.8797 -0.0007 0.5568 3.5626

1. 程序运行如下：

TestLag3([100 121 144], [5 50 115 185])

计算插值的点 xi =

5 50 115 185

计算出的插值 yi =

4.4391 7.2850 10.7228 13.5357

插值点处函数值 yFact =

2.2361 7.0711 10.7238 13.6015

计算误差 err =

-2.2030 -0.2139 0.0010 0.0658

1. 程序运行如下：

TestLag3([169 196 225], [5 50 115 185])

计算插值的点 xi =

5 50 115 185

计算出的插值 yi =

5.4972 7.8001 10.8005 13.6006

插值点处函数值 yFact =

2.2361 7.0711 10.7238 13.6015

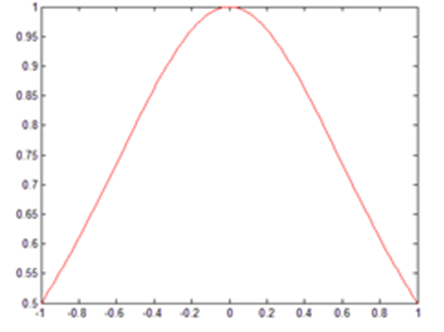
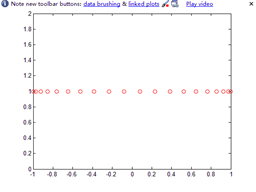
计算误差 err =

-3.2611 -0.7291 -0.0767 0.0009

思考题：

1. 拉格朗日插值多项式的次数并不是越大越好，根据定义，插值式可以在节点处与实际函数匹配，但不能保证在节点之间很好的逼近实际函数。这个现象就是多项式摆动—Runge现象，有时多项式摆动可以通过谨慎选择基础函数的取样点来减小。通常采用分段插值可以很好的消除多项式摆动现象。
2. 在分段段数相同的情况下，插值区间越大，误差越大。原因是大部分情况下，相对于比较大的区间，函数在比较小的区间上的函数值变化较缓和，因此即使出现摆动也不会偏离原函数太大。
3. 第一问中已经提到多项式摆动可以通过谨慎选择基础函数的取样点来减小。我们来看下

的分布和的图像：



可以看出，的分布是两端密集中间稀疏，的趋势是两边陡峭中间平缓，函数变化陡峭时节点增多正好可以增加插值的准确性。

1. 一般来说，内插时插值收敛于实际函数，一旦超出内插的范围，插值函数会发散，且离插值区间越远外推误差越大。使用不用的插值方法在同一点外推的值也会相差很多，这说明外推本身就存在很大的不确定性。

实验报告二

题目： *Romberg*积分法

摘要：对于实际的工程积分问题，很难应用*Newton-Leibnitz*公式去求解。因此应用数值方法进行求解积分问题已经有着很广泛的应用，本文基于*Romberg*积分法来解决一类积分问题。

前言：（目的和意义）

1. 理解和掌握*Romberg*积分法的原理；
2. 学会使用*Romberg*积分法；
3. 明确*Romberg*积分法的收敛速度及应用时容易出现的问题。

数学原理：

考虑积分，欲求其近似值，通常有复化的梯形公式、*Simpsion*公式和*Cotes*公式。但是给定一个精度，这些公式达到要求的速度很缓慢。如何提高收敛速度，自然是人们极为关心的课题。为此，记*T1,k*为将区间[*a,b*]进行*2k*等分的复化的梯形公式计算结果，记*T2,k*为将区间[*a,b*]进行*2k*等分的复化的*Simpsion*公式计算结果，记*T3,k*为将区间[*a,b*]进行*2k*等分的复化的*Cotes*公式计算结果。根据Richardson外推加速方法，可以得到收敛速度较快的*Romberg*积分法。其具体的计算公式为：

1. 准备初值，计算



1. 按梯形公式的递推关系，计算



1. 按*Romberg*积分公式计算加速值

 *m=2,…，k*

1. 精度控制。对给定的精度，若



则终止计算，并取为所求结果；否则返回2重复计算，直至满足要求的精度为止。

程序设计：

本实验采用*Matlab*的*M*文件编写。其中待积分的函数写成*function*的方式，例如如下

*function yy=f(x,y);*

*yy=x.^3;*

写成如上形式即可，下面给出主程序

***Romberg*积分法源程序**

%%% *Romberg*积分法

*clear*

%%%积分区间

*b=3;*

*a=1;*

%%%精度要求

*R=1e-5;*

%%%应用梯形公式准备初值

*T(1,1)=(b-a)\*(f(b)+f(a))/2;*

*T(1,2)=T(1,1)/2+(b-a)/2\*f((b+a)/2);*

*T(2,1)=(4\*T(1,2)-T(1,1))/(4-1);*

*j=2;*

*m=2;*

%%%主程序体%%%

*while(abs(T(m,1)-T(m-1,1))>R);*%%%精度控制

*j=j+1;*

*s=0;*

*for p=1:2^(j-2);*

*s=s+f(a+(2\*p-1)\*h/(2^(j-1)));*

*end*

*T(1,j)=T(1,j-1)/2+h\*s/(2^(j-1));* %%%梯形公式应用

*for m=2:j;*

*k=(j-m+1);*

*T(m,k)=((4^(m-1))\*T(m-1,k+1)-T(m-1,k))/(4^(m-1)-1);*

*end*

*end*

%%%给出 *Romberg*积分法的函数表

*I=T(m,1)*

实验结果、结论与讨论

进行具体的积分时，精度取*R=1e-8*。

1. 求积分。精确解*I= 24999676*。

运行程序得*Romberg*积分法的函数表为

1.0e+007 \*

4.70101520000000 3.05022950000000 2.63753307500000

2.49996760000000 2.49996760000000 0

2.49996760000000 0 0

由函数表知*Romberg*积分给出的结果为2.4999676\*10^7，与精确没有误差，

精度很高。

1. 求积分。精确解*I=ln3= 1.09861228866811*。

运行程序得*Romberg*积分法的函数表为

1.33333333333333 1.16666666666667 1.11666666666667 1.10321067821068 1.09976770156303 1.09890151516846 1.09868461878559

1.11111111111111 1.10000000000000 1.09872534872535 1.09862004268048 1.09861278637027 1.09861231999130 0

1.09925925925926 1.09864037197371 1.09861302227749 1.09861230261625 1.09861228889937 0 0

1.09863054836600 1.09861258815533 1.09861229119306 1.09861228868164 0 0 0

1.09861251772313 1.09861229002850 1.09861228867179 0 0 0 0

1.09861228980593 1.09861228867046 0 0 0 0 0

1.09861228867019 0 0 0 0 0 0

从积分表中可以看出程序运行的结果为1.09861228867019，取8位有效数字，满足要求。

1. 求积分。

直接按前面方法进行积分，会发现系统报错，出现了0为除数的现象。出现这种情况的原因就是当x=0时，被积函数分母出现了0，如果用一个适当的小数（最好不要小于程序给定的最小误差值，但是不能小于机器的最大精度）来代替，可以避免这个问题。本实验取，可得函数表为：

0.92073548319659 0.93979327500190 0.94451351171417 0.94569085359489 0.94598501993743

0.94614587227034 0.94608692395160 0.94608330088846 0.94608307538495 0

0.94608299406368 0.94608305935092 0.94608306035138 0 0

0.94608306038722 0.94608306036726 0 0 0

0.94608306036718 0 0 0 0

故该函数的积分为0.94608306036718，取8位有效数字。

1. 求积分

本题的解析解很难给出，但运用*Romberg*积分可以很容易给出近似解，函数表为：

0.42073549240395 0.33406972582924 0.31597536075922 0.31168023948094 0.31062036680949 0.31035626065456

0.30518113697100 0.30994390573588 0.31024853238818 0.31026707591900 0.31026822526959 0

0.31026142365354 0.31026884083167 0.31026831215439 0.31026830189296 0 0

0.31026895856465 0.31026830376269 0.31026830173008 0 0 0

0.31026830119484 0.31026830172211 0 0 0 0

0.31026830172262 0 0 0 0 0

故该函数的积分为0.31026830172262，取8位有效数字。

1. 利用龙贝格(Romberg)积分法计算积分

(1)程序运行如下：

I = Romberginterg(inline('x.^2.\*exp(x)'),0,1,25,1e-6)

I = 0.7183

(2)程序运行如下：

I = Romberginterg(inline('exp(x).\*sin(x)'),1,3,25,1e-6)

I = 10.9502

(3)程序运行如下：

I = Romberginterg(inline('4./(1+x.^2)'),0,1,25,1e-6)

I = 3.1416

(4)程序运行如下：

I = Romberginterg(inline('1./(1+x)'),0,1,25,1e-6)

I = 0.6931

2. 被积函数无界，如何处理？

(1) 程序运行如下：

I = Romberginterg(inline('cos(x)./sqrt(1-x)'),0,1,25,1e-6)

I = NaN

因为函数在x=0处出现了0/0的情况，极限为1，所以Matlab的结果显示为NaN非数，解决方法是把下限0改为一个小数eps。

I = Romberginterg(inline('sin(x)./x'),eps,1,25,1e-6)

I = 0.9461

(2) 程序运行如下：

I = Romberginterg(inline('cos(x)./sqrt(1-x)'),0,1,25,1e-6)

I = NaN

与（1）的原因相同，函数在x=1处出现了0/0的情况，结果为无穷，此时应选择变换，将积分变为，再进行变换，将积分变为

，变换后输入命令：

I = RombergInterg(inline('2\*sin(x).\*cos((sin(x)).^2)'),0,pi/2,25,1e-6)

I = 1.4996

(3) 程序运行如下：

I = Romberginterg(inline('cos(x)./sqrt(x)'),0,1,25,1e-6)

I = NaN

函数在x=0处出现了1/0的情况，结果为无穷。先将积分

，再做变换，

I = 2\*Romberginterg(inline('cos(x.^2)'),0,1,25,1e-6)

I = 1.8090

(4)程序运行如下：

I = Romberginterg(inline('x.\*sin(x)./(1-x.^2)'),-1,1,25,1e-6)

I = NaN

函数在x=1,-1处出现了sin1/0的情况，结果为无穷。被积函数为偶函数，

变换，积分变为

I = 2\*Romberginterg(inline('sin(x).\*sin(sin(x))'),0,pi/2,25,1e-6)

I = 1.3825

或者使用Gauss-Chebyshev求积公式：

I = GaussInterg(inline('x.\*sin(x)'),'Chebyshev',-1,1,1e-6)

I = 1.3825

由于Gauss-Chebyshev求积公式求的是在的积分因此

处的为。

3. 积分区间无限，如何处理？

(1) 程序运行如下：

I = Romberginterg(inline('exp(-x.^2)'),-10,10,25,1e-6)

I = 1.7725

由于积分区间无限，而函数在[-10,10]外很小，可以用函数在[-10,10

上的积分近似这个无穷积分。或者使用Gauss-Hermite求积公式：

I = GaussInterg(inline('x+1-x'),'Hermite',0,0,1e-6)

I = 1.7725

由于Gauss-Hermite求积公式求的是在的积分因此此处

为1。

(2) 程序运行如下：

先做变换，则原积分变为

I = 2\*Romberginterg(inline('1./(1+x.^2)'),0,1,25,1e-6)

I = 1.5708

(3) 程序运行如下：

使用Gauss-Hermite求积公式：

I = GaussInterg(inline('cos(x).^3'),'Hermite',0,0,1e-6)

I = 1.0820

由于Gauss-Hermite求积公式求的是在的积分因此此处的为。

(4)程序运行如下:

使用Gauss-Laguerre求积公式：

I = GaussInterg(inline('sin(x).^2'),'Laguerre',0,0,1e-6)

I = 0.4000

由于Gauss-Laguerre求积公式求的是在[0,]的积分因此此处的为 。

思考题：

1. 输入的参数N越大，在有限区间上分段越小，由复化梯形公式的误差，计算精度越高。
2. 二分次数越多，计算精度越高。因为每一次二分，N变为原来的二倍，由复化梯形公式的误差，二分后误差约为二分前的四分之一。
3. 有普遍性，被积函数在积分区间内存在奇点。若在奇点上极限存在，则可以利用小数—机器Epsilon将奇点从积分区间去除，如问题2（1）。若奇点上极限不存在，则需要根据具体情况进行积分变换使被积函数在积分区间上不存在奇点。若求的是在的积分，则可以使用Gauss-Chebyshev求积公式。
4. 有普遍性，积分区间均为无限区间。若求的是在的积分可用Gauss-Hermite求积公式，若求的是在的积分可用Gauss-Laguerre求积公式，其他形式根据具体情况进行积分变换使积分区间变为有限区间。

**结论：**

*Romberg*积分通常要求被积函数在积分区间上没有奇点。如有奇点，且奇点为第一间断点，那么采用例3的方法，还是能够求出来的，否则，必须采用其它的积分方法。当然，*Romberg*积分的收敛速度还是比较快的。

实验报告三

题目： 非线性方程求解

摘要：非线性方程的解析解通常很难给出，因此线性方程的数值解法就尤为重要。本实验采用两种常见的求解方法二分法和Newton法及改进的Newton法。

前言：（目的和意义）

掌握二分法与Newton法的基本原理和应用。

数学原理：

对于一个非线性方程的数值解法很多。在此介绍两种最常见的方法：二分法和Newton法。

对于二分法，其数学实质就是说对于给定的待求解的方程*f(x)*，其在[*a,b*]上连续，*f(a)f(b)<0*，且f(x)在[*a,b*]内仅有一个实根*x\**，取区间中点*c*，若，则*c*恰为其根，否则根据*f(a)f(c)<0*是否成立判断根在区间[*a,c*]和[*c,b*]中的哪一个，从而得出新区间，仍称为[*a,b*]。重复运行计算，直至满足精度为止。这就是二分法的计算思想。

Newton法通常预先要给出一个猜测初值x0，然后根据其迭代公式



产生逼近解*x\**的迭代数列{*xk*}，这就是Newton法的思想。当*x0*接近*x\**时收敛很快，但是当*x0*选择不好时，可能会发散，因此初值的选取很重要。另外，若将该迭代公式改进为



其中*r*为要求的方程的根的重数，这就是改进的Newton法，当求解已知重数的方程的根时，在同种条件下其收敛速度要比Newton法快的多。

程序设计：

本实验采用*Matlab*的*M*文件编写。其中待求解的方程写成*function*的方式，如下

*function y=f(x);*

*y=-x\*x-sin(x);*

写成如上形式即可，下面给出主程序。

**二分法源程序：**

*clear*

%%%给定求解区间

*b=1.5;*

*a=0;*

%%%误差

*R=1;*

*k=0;*%迭代次数初值

*while (R>5e-6) ;*

*c=(a+b)/2;*

*if f12(a)\*f12(c)>0;*

*a=c;*

*else*

*b=c;*

*end*

*R=b-a;%求出误差*

*k=k+1;*

*end*

*x=c*%给出解

**Newton法及改进的Newton法源程序：**

*clear*

%%%% 输入函数

*f=input('请输入需要求解函数>>','s')*

%%%求解f(x)的导数

*df=diff(f);*

%%%改进常数或重根数

*miu=2;*

%%%初始值x0

*x0=input('input initial value x0>>');*

*k=0*;%迭代次数

*max=100*;%最大迭代次数

*R=eval(subs(f,'x0','x'));*%求解f(x0)，以确定初值x0时否就是解

*while (abs(R)>1e-8)*

*x1=x0-miu\*eval(subs(f,'x0','x'))/eval(subs(df,'x0','x'));*

*R=x1-x0;*

*x0=x1;*

*k=k+1;*

*if (eval(subs(f,'x0','x'))<1e-10);*

*break*

*end*

*if k>max*;%如果迭代次数大于给定值，认为迭代不收敛，重新输入初值

*ss=input('maybe result is error,choose a new x0,y/n?>>','s');*

*if strcmp(ss,'y')*

*x0=input('input initial value x0>>');*

*k=0;*

*else*

*break*

*end*

*end*

*end*

*k;*%给出迭代次数

*x=x0；*%给出解

实验结果、结论与讨论：

1. 用二分法计算方程在[1，2]内的根。(,下同)

计算结果为

x= 1.40441513061523；

f(x)= -3.797205105904311e-007；

k=18；

由f(x)知结果满足要求，但迭代次数比较多，方法收敛速度比较慢。

1. 用二分法计算方程在[1，1.5]内的根。

计算结果为

x= 1.32471847534180；

f(x)= 2.209494846194815e-006；

k=17；

由f(x)知结果满足要求，但迭代次数还是比较多。

1. 用Newton法求解下列方程
2.  x0=0.5；

计算结果为

x= 0.56714329040978；

f(x)= 2.220446049250313e-016；

k=4；

由f(x)知结果满足要求，而且又迭代次数只有4次看出收敛速度很快。

1.  x0=1；
2.  x0=0.45, x0=0.65；

当x0=0.45时，计算结果为

x= 0.49999999999983；

f(x)= -8.362754932994584e-014；

k=4；

由f(x)知结果满足要求，而且又迭代次数只有4次看出收敛速度很快，实际上该方程确实有真解x=0.5。

当x0=0.65时，计算结果为

x= 0.50000000000000；

f(x)=0；

k=9；

由f(x)知结果满足要求，实际上该方程确实有真解x=0.5，但迭代次数增多，实际上当取x0〉0.68时，x≈1，就变成了方程的另一个解，这说明Newton法收敛与初值很有关系，有的时候甚至可能不收敛。

1. 用改进的Newton法求解，有2重根，取

 x0=0.55；并与3.中的c)比较结果。

当x0=0.55时，程序死循环，无法计算，也就是说不收敛。改时，结果收敛为

x=0.50000087704286；

f(x)=4.385198907621127e-007；

k=16；

显然这个结果不是很好，而且也不是收敛至方程的2重根上。

当x0=0.85时，结果收敛为

x= 1.00000000000489；

f(x)= 2.394337647718737e-023；

k=4；

这次达到了预期的结果，这说明初值的选取很重要，直接关系到方法的收敛性，实际上直接用Newton法，在给定同样的条件和精度要求下，可得其迭代次数k=15，这说明改进后的Newton法法速度确实比较快。

问题：

1.

1. 程序运行如下：

r = NewtSolveOne('fun1\_1',pi/4,1e-6,1e-4,10)

r = 0.7391

1. 程序运行如下：

r = NewtSolveOne('fun1\_2',0.6,1e-6,1e-4,10)

r = 0.5885

2,

1. 程序运行如下：

r = NewtSolveOne('fun2\_1',0.5,1e-6,1e-4,10)

r = 0.5671

1. 程序运行如下：

r = NewtSolveOne('fun2\_2',0.5,1e-6,1e-4,20)

r = 0.5669

3.

1. 程序运行如下：

① p = LegendreIter(2)

p = 1.0000 0 -0.3333

p = LegendreIter(3)

p = 1.0000 0 -0.6000 0

p = LegendreIter(4)

p = 1.0000 0 -0.8571 0 0.0857

p = LegendreIter(5)

p = 1.0000 0 -1.1111 0 0.2381 0

② p = LegendreIter(6)

p = 1.0000 0 -1.3636 0 0.4545 0 -0.0216

r = root(p)'

r = -0.932469514203150 -0.661209386466265 0.932469514203153

0.661209386466264 -0.238619186083197 0.238619186083197

用二分法求根为：

r = BinSolve('LegendreP6',-1,1,1e-6)

r = -0.932470204878826 -0.661212531887755 -0.238620057397959

0.238600127551020 0.661192602040816 0.932467713647959

1. 程序运行如下：

① p = ChebyshevIter(2)

p = 1.0000 0 -0.5000

p = ChebyshevIter(3)

p = 1.0000 0 -0.7500 0

p = ChebyshevIter(4)

p = 1.0000 0 -1.0000 0 0.1250

p = ChebyshevIter(5)

p = 1.0000 0 -1.2500 0 0.3125 0

② p = ChebyshevIter(6)

p = 1.0000 0 -1.5000 0 0.5625 0 -0.0313

r = roots(p)'

r = -0.965925826289067 -0.707106781186548 0.965925826289068

0.707106781186547 -0.258819045102521 0.258819045102521

用二分法求根为：

r = BinSolve('ChebyshevT6',-1,1,1e-6)

r = -0.965929926658163 -0.707110969387755 -0.258828922193878

0.258818957270408 0.707105986926020 0.965924944196429

与下列代码结果基本一致，只是元素顺序稍有不同：

j = 0:5;

x = cos((2\*j+1)\*pi/2/(5+1))

x = 0.965925826289068 0.707106781186548 0.258819045102521

-0.258819045102521 -0.707106781186547 -0.965925826289068

1. 程序运行如下：

① p = LaguerreIter(2)

p = 1 -4 2

p = LaguerreIter(3)

p = 1 -9 18 -6

p = LaguerreIter(4)

p = 1 -16 72 -96 24

p = LaguerreIter(5)

p =1.0000 -25.0000 200.0000 -600.0000 600.0000 -120.000

② p = LaguerreIter(5)

p =1.0000 -25.0000 200.0000 -600.0000 600.0000 -120.000

r = roots(p)'

r = 12.640800844275732 7.085810005858891 3.596425771040711

1.413403059106520 0.263560319718141

用二分法求根为：

r = BinSolve('LaguerreL5',0,13,1e-6)

r = 0.263560314567722 1.413403056105789 3.596425765631150

7.085810005360720 12.640800843813590

1. 程序运行如下：

① p = HermiteIter(2)

p = 1.0000 0 -0.5000

p = HermiteIter(3)

p = 1.0000 0 -1.5000 0

p = HermiteIter(4)

p = 1.0000 0 -3.0000 0 0.7500

p = HermiteIter(5)

p = 1.0000 0 -5.0000 0 3.7500 0

② p = HermiteIter(6)

p = 1.0000 0 -7.5000 0 11.2500 0 -1.8750

r = roots(p)'

r = -2.350604973674487 2.350604973674488 -1.335849074013696

1.335849074013698 -0.436077411927617 0.436077411927616

用二分法求根为：

r = BinSolve('HermiteH6',-3,3,1e-6)

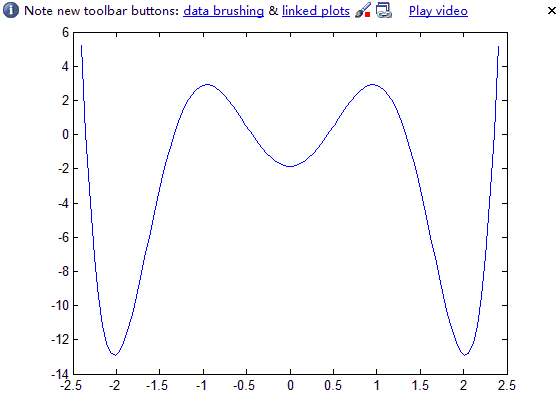
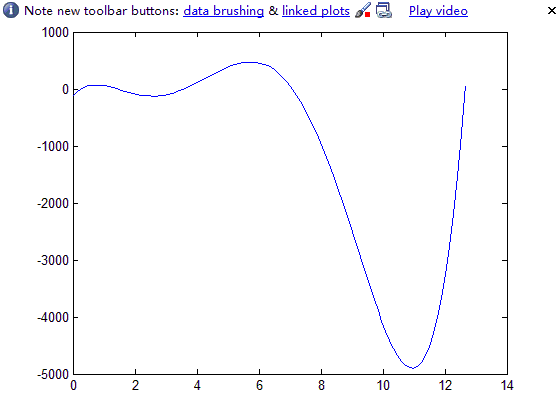
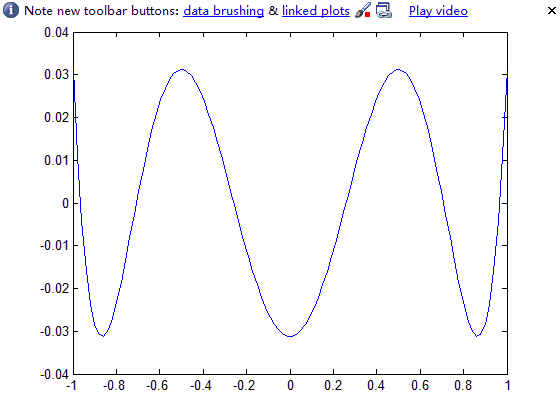
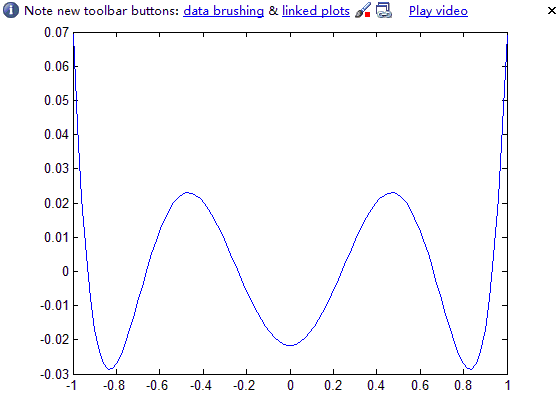
r = -2.350604981792216 -1.335849100229691 -0.436077818578603

0.436077351472816 1.335848983453244 2.350604952598104

思考题：

1. 由于Newton法具有局部收敛性，所以在实际问题中，当实际问题本身能提供接近于根的初始近似值时，就可保证迭代序列收敛，但当初值难以确定时，迭代序列就不一定收敛。实际计算时应先用比较稳定的算法，如二分法，计算根的近似值，再将该近似值作为牛顿法的初值，以保证迭代序列的收敛性。
2. 实验2中两个方程根其实相同，只是第二个方程为重根，通过比较迭代次数，第一个方程迭代了3次得出结果，第二个方程迭代了8次得出结果，且第二个方程的结果不如第一个准确，这是由于第二个方程在根处导数为0，在根的领域内导数很小使Newton法收敛速度变慢，精度变低。

3. Legendre P6 Chebyshev T6



**结论：**

对于二分法，只要能够保证在给定的区间内有根，使能够收敛的，当时收敛的速度和给定的区间有关，二且总体上来说速度比较慢。Newton法，收敛速度要比二分法快，但是最终其收敛的结果与初值的选取有关，初值不同，收敛的结果也可能不一样，也就是结果可能不时预期需要得结果。改进的Newton法求解重根问题时，如果初值不当，可能会不收敛，这一点非常重要，当然初值合适，相同情况下其速度要比Newton法快得多。

实验报告四

题目： Gauss列主元消去法

摘要：求解线性方程组的方法很多，主要分为直接法和间接法。本实验运用直接法的Guass消去法,并采用选主元的方法对方程组进行求解。

前言：（目的和意义）

1. 学习Gauss消去法的原理。
2. 了解列主元的意义。
3. 确定什么时候系数阵要选主元

数学原理：

由于一般线性方程在使用Gauss消去法求解时，从求解的过程中可以看到，若=0，则必须进行行交换，才能使消去过程进行下去。有的时候即使0，但是其绝对值非常小，由于机器舍入误差的影响，消去过程也会出现不稳定得现象，导致结果不正确。因此有必要进行列主元技术，以最大可能的消除这种现象。这一技术要寻找行*r*，使得



并将第*r*行和第*k*行的元素进行交换，以使得当前的的数值比0要大的多。这种列主元的消去法的主要步骤如下：

1. 消元过程

对*k*=1,2,…,*n*-1,进行如下步骤。

* 1. 选主元，记



若很小，这说明方程的系数矩阵严重病态，给出警告，提示结果可能不对。

* 1. 交换增广阵A的*r*，*k*两行的元素。

 (*j=k,…,n*+1)

* 1. 计算消元

 (*i=k+*1,…,*n*; *j*=*k*+1,……,*n*+1)

1. 回代过程

对*k*= *n*, *n*-1,…,1,进行如下计算



至此，完成了整个方程组的求解。

程序设计：

本实验采用*Matlab*的*M*文件编写。

**Gauss消去法源程序：**

*clear*

*a=input('输入系数阵：>>\n')*

*b=input('输入列阵b：>>\n')*

*n=length(b);*

*A=[a b]*

*x=zeros(n,1);*

%%%函数主体

*for k=1:n-1;*

%%%是否进行主元选取

*if abs(A(k,k))<yipusilong;*%事先给定的认为有必要选主元的小数

*yzhuyuan=1;*

*else yzhuyuan=0;*

*end*

*if yzhuyuan;*

%%%%选主元

*t=A(k,k);*

*for r=k+1:n;*

*if abs(A(r,k))>abs(t)*

*p=r;*

*else p=k;*

*end*

*end*

%%%交换元素

*if p~=k;*

*for q=k:n+1;*

*s=A(k,q);*

*A(k,q)=A(p,q);*

*A(p,q)=s;*

*end*

*end*

*end*

%%%判断系数矩阵是否奇异或病态非常严重

*if abs(A(k,k))< yipusilong*

*disp(‘矩阵奇异，解可能不正确’)*

*end*

%%%%计算消元,得三角阵

*for r=k+1:n;*

*m=A(r,k)/A(k,k);*

*for q=k:n+1;*

*A(r,q)=A(r,q)-A(k,q)\*m;*

*end*

*end*

*end*

%%%%求解x

*x(n)=A(n,n+1)/A(n,n);*

*for k=n-1:-1:1;*

*s=0;*

*for r=k+1:n;*

*s=s+A(k,r)\*x(r);*

*end*

*t=(A(k,n+1)-s)*

*x(k)=(A(k,n+1)-s)/A(k,k)*

*end*

实验结果、结论与讨论：

例：求解方程。其中为一小数，当时，分别采用列主元和不列主元的Gauss消去法求解，并比较结果。

记*Emax*为求出的解代入方程后的最大误差，按要求，计算结果如下：

当时，不选主元和选主元的计算结果如下，其中前一列为不选主元结果，后一列为选主元结果，下同。

0.99999934768391 0.99999934782651

2.00000217421972 2.00000217391163

2.99999760859451 2.99999760869721

*Emax=* 9.301857062382624e-010，0

此时，由于不是很小，机器误差就不是很大，由*Emax*可以看出不选主元的计算结果精度还可以，因此此时可以考虑不选主元以减少计算量。

当时，不选主元和选主元的计算结果如下

1.00001784630877 0.99999999999348

1.99998009720807 2.00000000002174

3.00000663424731 2.99999999997609

*Emax=* 2.036758973744668e-005，0

此时由*Emax*可以看出不选主元的计算精度就不好了，误差开始增大。

当时，不选主元和选主元的计算结果如下

1.42108547152020 1.00000000000000

1.66666666666666 2.00000000000000

3.11111111111111 300000000000000

*Emax=* 0.70770085900503，0

此时由*Emax*可以看出，不选主元的结果应该可以说是不正确了，这是由机器误差引起的。

当时，不选主元和选主元的计算结果如下

NaN 1

NaN 2

NaN 3

*Emax=*NaN, 0

不选主元时，程序报错：Warning: Divide by zero.。这是因为机器计算的最小精度为10-15，所以此时的就认为是0，故出现了错误现象。而选主元时则没有这种现象，而且由*Emax*可以看出选主元时的结果应该是精确解。

## 问题1

1. 程序运行如下：

x = GaussSysSolve(Mat1\_1,b1\_1)

x = 1.0000 1.0000 1.0000 1.0000

x = GaussExpSysSolve(Mat1\_1,b1\_1)

x = 1.0000 1.0000 1.0000 1.0000

x = GaussIneSysSolve(Mat1\_1,b1\_1)

x = 1.0000 1.0000 1.0000 1.0000

1. 程序运行如下：

x = GaussSysSolve(Mat1\_2,b1\_2)

x = 1.0000 1.0000 1.0000 1.0000

x = GaussExpSysSolve(Mat1\_2,b1\_2)

x = 1.0000 1.0000 1.0000 1.0000

x = GaussIneSysSolve(Mat1\_2,b1\_2)

x = 1.0000 1.0000 1.0000 1.0000

1. 程序运行如下：

x = GaussSysSolve(Mat1\_3,b1\_3)

x = 1.0000 1.0000 1.0000 1.0000

x = GaussExpSysSolve(Mat1\_3,b1\_3)

x = 1.0000 1.0000 1.0000 1.0000

x = GaussIneSysSolve(Mat1\_3,b1\_3)

x = 1.0000 1.0000 1.0000 1.0000

1. 程序运行如下：

x = GaussSysSolve(Mat1\_4,b1\_4)

x = 1.0000 1.0000 1.0000 1.0000

x = GaussExpSysSolve(Mat1\_4,b1\_4)

x = 1.0000 1.0000 1.0000 1.0000

x = GaussIneSysSolve(Mat1\_4,b1\_4)

x = 1.0000 1.0000 1.0000 1.0000

## 问题2

1. 程序运行如下：

= GaussSysSolve(Mat2\_1,b2\_1)

x = 1.0915 0.2832 1.1463 -0.1008

x = GaussExpSysSolve(Mat2\_1,b2\_1)

x = 1.0915 0.2832 1.1463 -0.1008

x = GaussIneSysSolve(Mat2\_1,b2\_1)

x = 1.0915 0.2832 1.1463 -0.1008

1. 程序运行如下：

x = GaussSysSolve(Mat2\_2,b2\_2)

x = 0.5162 0.4152 0.1100 1.0365

x = GaussExpSysSolve(Mat2\_2,b2\_2)

x = 0.5162 0.4152 0.1100 1.0365

x = GaussIneSysSolve(Mat2\_2,b2\_2)

x = 0.5162 0.4152 0.1100 1.0365

1. 程序运行如下：

x = GaussSysSolve(Mat2\_3,b2\_3)

x = 1.0000 1.0000 1.0000

x = GaussExpSysSolve(Mat2\_3,b2\_3)

x = 1 1 1

x = GaussIneSysSolve(Mat2\_3,b2\_3)

x = 1.0000 1.0000 1.0000

1. 程序运行如下：

x = GaussSysSolve(Mat2\_4,b2\_4)

x = 1 1 1

x = GaussExpSysSolve(Mat2\_4,b2\_4)

x = 1.0000 1.0000 1.0000

x = GaussIneSysSolve(Mat2\_4,b2\_4)

x = 1 1 1

## 思考题

1. 在各主元不是非常小的时候，三种方法结果一致
2. 隐式平衡列选主元法最好，应为当主元很小时，普通的Gauss消元法会产生很大误差；显式平衡列选主元法每一行除以其绝对值最大元素时会引入额外的舍入误差。
3. 显式平衡列选主元法实际上是将增广矩阵各行分别除以绝对值最大元素后构成新的增广矩阵后再使用Gauss消去法。

**结论：**

采用Gauss消去法时，如果在消元时对角线上的元素始终较大（假如大于10-5），那么本方法不需要进行列主元计算，计算结果一般就可以达到要求，否则必须进行列主元这一步，以减少机器误差带来的影响，使方法得出的结果正确。