**P20/17上机题（用C++编写）**

1. 编写计算从大到小的程序

**程序代码：**

#include<iostream>

using namespace std;

int main()

{

float sn=0;

float n;

while(cin >> n)

{for(float i=2;i<=n;i++)

{

sn=sn+1/(i\*i-1);

}

cout << "当N="<<n<<",从大到小的值为"<<sn<< endl;sn=0;}

return 0;

}

1. 编写计算从小到大的程序

**程序代码：**

#include<iostream>

using namespace std;

int main()

{

float sn=0;

float n;

while(cin >> n)

{for(float i=n;i>=2;i--)

{

sn=sn+1/(i\*i-1);

}

cout << "当N="<<n<<",从小到大的值为"<<sn<< endl;sn=0;}

return 0;

}

**合并成总程序：**

#include <cstdio>

using namespace std;

int main()

{

float sn,snreal,sn1=0;

float n;

while(scanf("%f",&n)!=EOF)

{for(float i=n;i>=2;i--)

{

sn+=1/(i\*i-1);

}

for(float i=2;i<=n;i++)

{

sn1=sn1+1/(i\*i-1);

}

snreal+=0.5\*(1.5-(1/n)-1/(n+1));

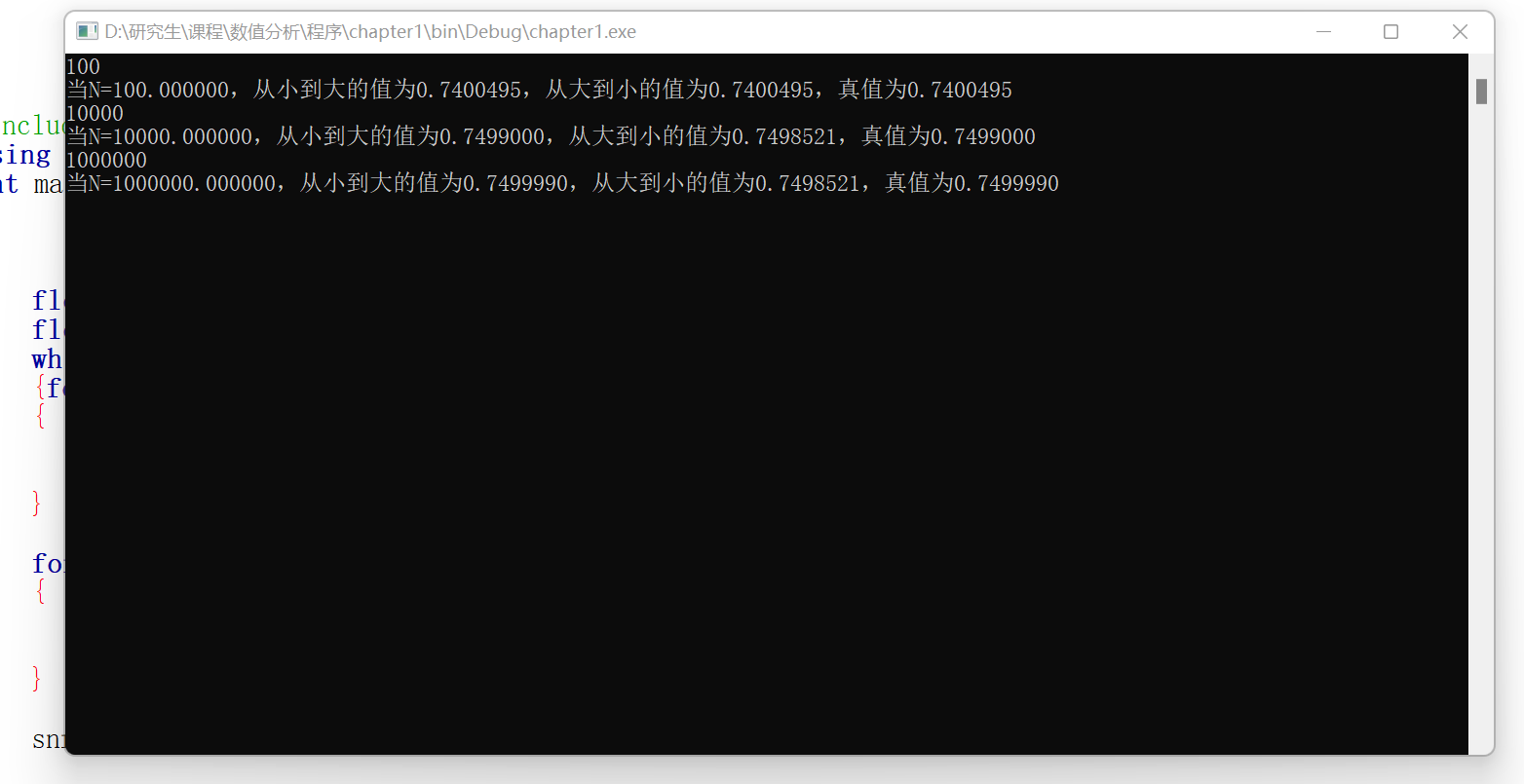
printf("当N=%f，从小到大的值为%.7f，从大到小的值为%.7f，真值为%.7f",n,sn,sn1,snreal);

sn=0;snreal=0;sn1=0;

}

}

1. 算N=100,N=10000,N=1000000时的值，指出有效位数。



**从大到小：有效位数分别是7、3、3。**

**从小到大：有效位数分别是7、7、7。**

注：C++中，采用单精度，**最大有效数字只能是7**，虽然可以存储大于7位数的有效数字，但已经丧失精度。

4．

可见，采用从大到小的计算方式，随着N的增大，误差增大。（N趋向于无穷会更明显）。这样的原因是由于“大数吃小数”的问题。由于存储是以单精度存储，也就意味着，如果一个大数加上一个小数时，很可能会忽略小数，而这种误差累计起来，将会很大！反过来，当先从小数加起，积累成一个较大的数，再与大数相加，就极有可能不损失任何的有效数了。

**P56/20上机题 牛顿迭代法（用C++编写）**

1. 编写Newton法通用根程序

**程序代码：**

#include <iostream>

#include <math.h>

using namespace std;

double f(double x)

{

double fx=pow(x,5)-2\*x\*x+6\*x+3;

return fx;

}

double df(double x)

{

double df=5\*pow(x,4)-4\*x+6;

return df;

}

int main()

{

double x,x0,error,gap;

printf("输入初值：");

scanf("%lf",&x);

printf("输入误差：");

scanf("%lf",&error);

x0=x;

gap=10;

while(gap>=error)

{

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

gap=abs(x-x0);

x0=x;

}

printf("牛顿法迭代值为%lf",x);

return 0;

}

以为例，求得的*Newton*迭代值为-0.434492



1. 确定尽可能大的，使得（）间的初值均收敛在。

**程序代码：**

#include <iostream>

#include <math.h>

using namespace std;

double fx(double x)

{

double fx=pow(x,3)/3-x; //表达式

return fx;

}

double dfx(double x)

{

double df=x\*x-1;

return df;

}

int main()

{

double mm=1e-6; //为了探测到最大值，每次增长的速度

double x,x0;

x0=0;

x=0; //初值

int i=0;

int p=0;

for(i=0;;i++)

{

x0=i\*mm;

x=x0-(fx(x0)/dfx(x0));

if(x0>=abs(x))

{

p=0; //若趋近于0，那么证明收敛，继续寻找！

}

else{p=1;break;}

}

printf("寻找到的最大范围为%lf",x0);

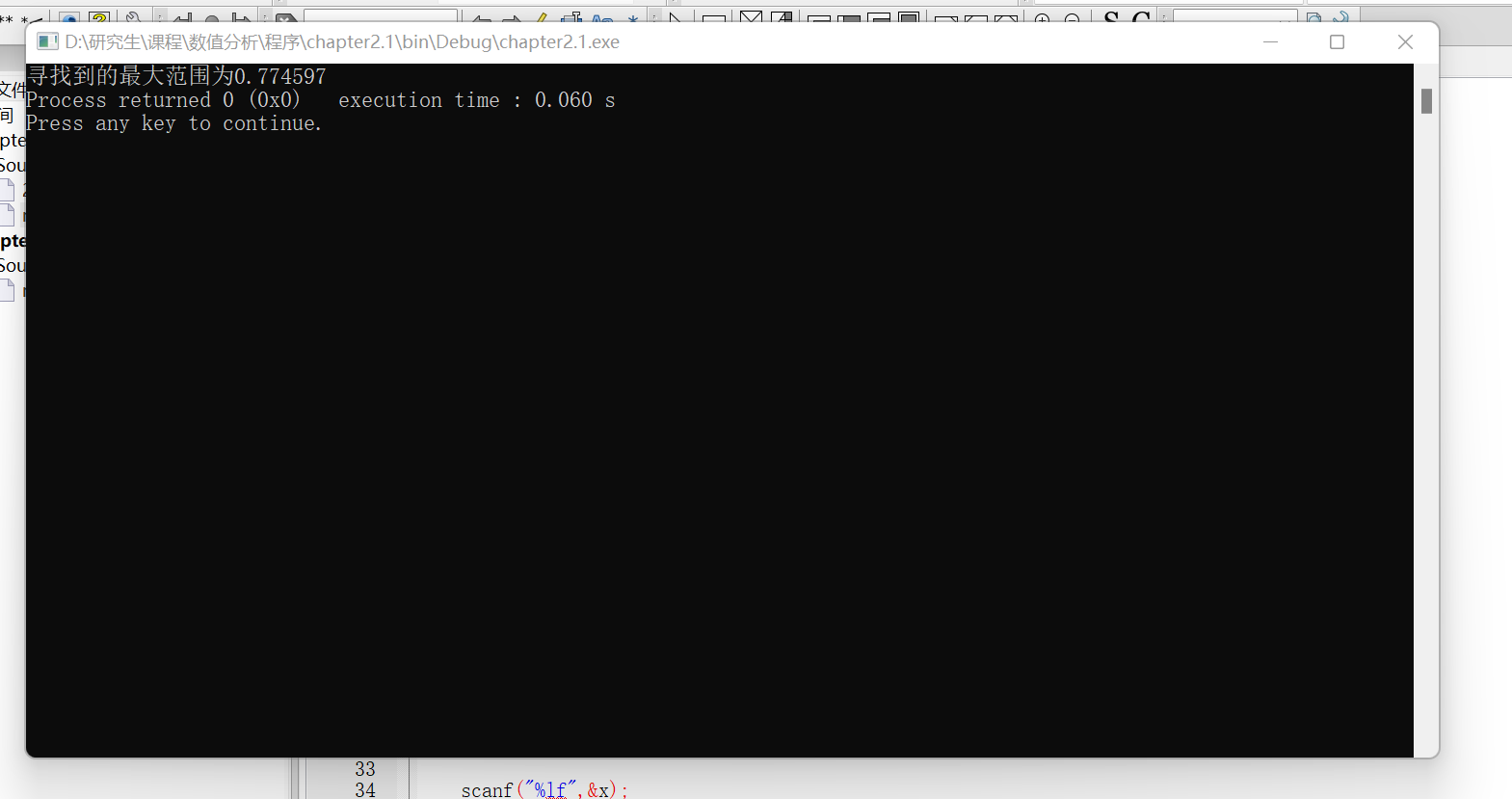
return 0;

}

程序的思路是：由于原函数是奇函数，**不妨令的选择满足以下条件：**



运行结果：



由于函数是奇函数，所以这个区间的最小值为-0.774597

故（）为（-0.774597，0.774597）。

3、判断函数初值选取在不同区间是否收敛以及收敛到哪个值

（1）****在（上

收敛到-1.732

（2）****在（上

收敛到1.732

（3）****在（上

收敛到0

（4）****在（上

收敛到-1.732

（5）****在（上

收敛到1.732

得出的结论：Newton迭代法对于初值的选择需要慎重（必须加以对图像的理解，知道根的大概区间）。上面的函数为什么会出现初值和迭代值相差过大的情况，是因为初值的切线会与远端的线相交。

**P125/40上机题（用MATLAB编写）**

1. 编写解*n*阶线性方程组*Ax=b*的列主元*Gauss*消去法的通用程序。

程序如下：

%% 定义初始矩阵，值可以随意更换

A=[2,-4,6;4,-9,2;1,-1,3]

b=[3;5;4]

T=[A,b]

%% 开始列主元,每次选定n-i列最大值，进行行调换，并且消去该列其他非零值。

n=length(b);

i=n-1;

k=i;

while(i ~= 0)

[maxnum,maxline]= max(T(:,n-i));

T([n-i maxline],:) = T([maxline n-i],:);

while(k ~= 0 )

T(n-k+1,:)=T(n-k+1,:)-T(n-i,:).\*( T(n-k+1,n-i) / T(n-i,n-i));

k=k-1;

end

i=i-1;

k=i;

T

end

T

%% 开始回代

xi=n;

j=xi;

while(xi ~= 0)

X(xi)=T(xi,n+1);

j=xi;

while((n-j) ~= 0)

X(xi)=X(xi)-X(n-j+xi)\*T(xi,n-j+xi);

j=j+1;

end

X(xi)=X(xi)/T(xi,xi);

xi=xi-1;

end

X

1. 用程序解*RI=V，*打印出解向量，保留五位有效数字。

%% 定义初始矩阵

A=[31,-13, 0,0,0,-10,0,0,0;

-13,35,-9,0,-11,0,0,0,0;

0,-9,31,-10,0,0,0,0,0;

0,0,-10,79,-30,0,0,0,-9;

0,0,0,-30,57,-7,0,-5,0;

0,0,0,0,-7,47,-30,0,0;

0,0,0,0,0,-30,41,0,0;

0,0,0,0,-5,0,0,27,-2;

0,0,0,-9,0,0,0,-2,29

]

b=[-15;27;-23;0;-20;12;-7;7;10]

T=[A,b]

%% 开始列主元,每次选定n-i列最大值，进行行调换，并且消去该列其他非零值。

n=length(b);

i=n-1;

k=i;

while(i ~= 0)

[maxnum,maxline]= max(T(:,n-i));

T([n-i maxline],:) = T([maxline n-i],:);

while(k ~= 0 )

T(n-k+1,:)=T(n-k+1,:)-T(n-i,:).\*( T(n-k+1,n-i) / T(n-i,n-i));

k=k-1;

end

i=i-1;

k=i;

T

end

T

%% 开始回代

xi=n;

j=xi;

while(xi ~= 0)

X(xi)=T(xi,n+1);

j=xi;

while((n-j) ~= 0)

X(xi)=X(xi)-X(n-j+xi)\*T(xi,n-j+xi);

j=j+1;

end

X(xi)=X(xi)/T(xi,xi);

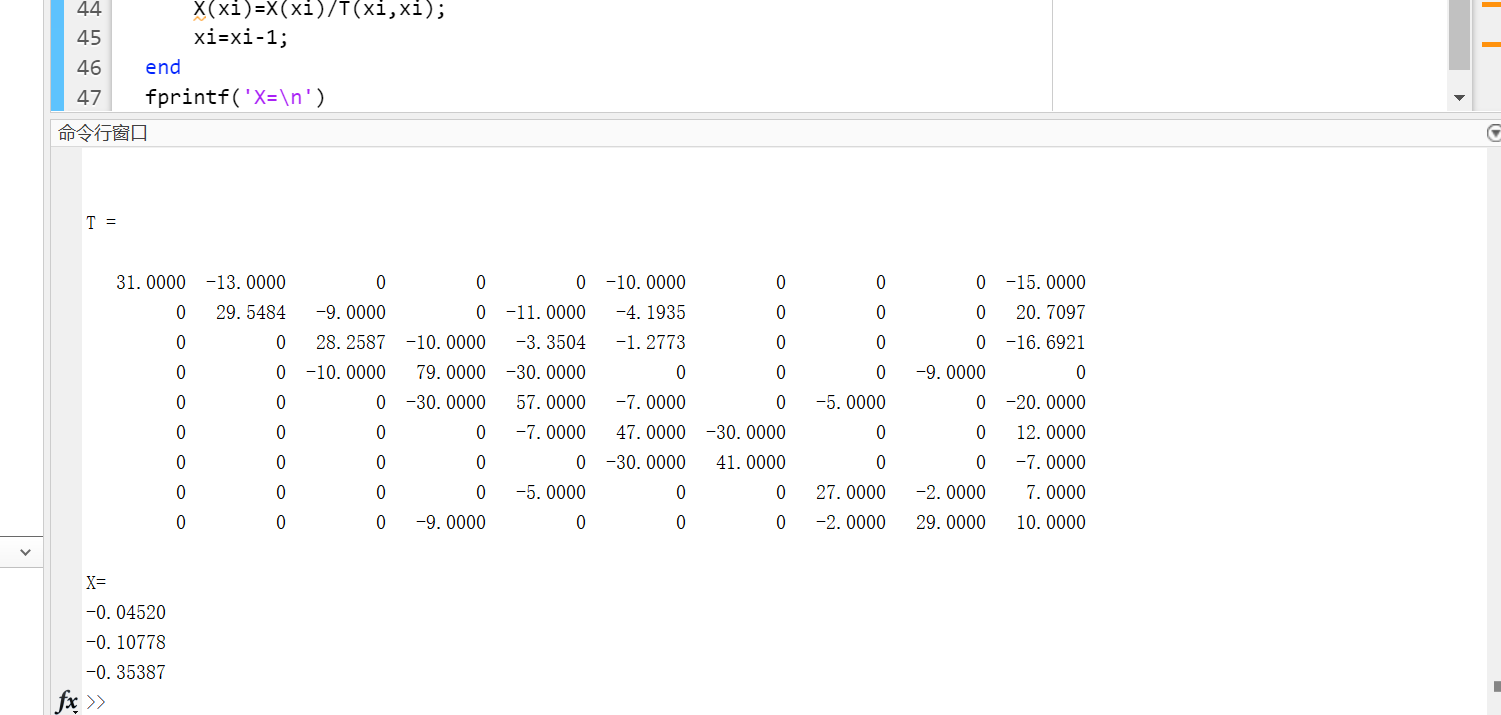
xi=xi-1;

end

fprintf('X=\n')

fprintf('%.5f\r',X)

结果：



1. 通过本次编程，我更加认识到了矩阵消去方法的强逻辑性，并且列主元方法可以适当减少不必要的误差。利用Matlab，处理矩阵更加便捷。

**P125/41上机题（用MATLAB编写）**

1. 编写SOR方法通用程序

程序如下：

A = [2,-1,1;2,-2,2;1,-1,3]

b = [3;5;4]

X = [9;9;2] %初值

w = 1; % 松弛因子

eps= 0.001;

error = 1;

U = triu(A,1)

L = tril(A,-1)

D = A-U-L

% 进行SOR迭代

while(error>eps)

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

if (vrho(sw)>=1) % 计算谱半径

fprintf("SOR无法收敛！")

break

end

fw=w \* (D+w\*L)\b;

xnew = sw\*X+fw;

error=max(abs(xnew-X));

X=xnew;

end

X

1. 计算上一题的线性方程组，取松弛因子容许误差,打印松弛因子、迭代次数、最佳松弛因子、解向量。

代码如下：

A=[31,-13, 0,0,0,-10,0,0,0;

-13,35,-9,0,-11,0,0,0,0;

0,-9,31,-10,0,0,0,0,0;

0,0,-10,79,-30,0,0,0,-9;

0,0,0,-30,57,-7,0,-5,0;

0,0,0,0,-7,47,-30,0,0;

0,0,0,0,0,-30,41,0,0;

0,0,0,0,-5,0,0,27,-2;

0,0,0,-9,0,0,0,-2,29

];

b=[-15;27;-23;0;-20;12;-7;7;10];

X = [0;0;0;0;0;0;0;0;0]; %初值

w = 1/50; % 松弛因子

eps= 0.5e-5;

error = 1;

U = triu(A,1);

L = tril(A,-1);

D = A-U-L;

i=1;

% 进行SOR迭代

while(i<=99)

diedainum=0;

w=i/50;

while(error>eps)

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

if (vrho(sw)>=1) % 计算谱半径

fprintf("SOR无法收敛！")

break

end

fw=w \* (D+w\*L)\b;

xnew = sw\*X+fw;

error=max(abs(xnew-X));

X=xnew;

diedainum=diedainum+1;

end

answer(i,1)=w;

answer(i,2)=diedainum;

i=i+1;

error=1;

end

X

|  |  |
| --- | --- |
| 松弛因子 | 迭代次数*N* |
| 0.0200000000000000 | 2181 |
| 0.0400000000000000 | 1115 |
| 0.0600000000000000 | 663 |
| 0.0800000000000000 | 460 |
| 0.100000000000000 | 346 |
| 0.120000000000000 | 274 |
| 0.140000000000000 | 225 |
| 0.160000000000000 | 189 |
| 0.180000000000000 | 162 |
| 0.200000000000000 | 141 |
| 0.220000000000000 | 124 |
| 0.240000000000000 | 110 |
| 0.260000000000000 | 99 |
| 0.280000000000000 | 89 |
| 0.300000000000000 | 81 |
| 0.320000000000000 | 74 |
| 0.340000000000000 | 68 |
| 0.360000000000000 | 62 |
| 0.380000000000000 | 58 |
| 0.400000000000000 | 54 |
| 0.420000000000000 | 50 |
| 0.440000000000000 | 47 |
| 0.460000000000000 | 44 |
| 0.480000000000000 | 41 |
| 0.500000000000000 | 38 |
| 0.520000000000000 | 36 |
| 0.540000000000000 | 34 |
| 0.560000000000000 | 32 |
| 0.580000000000000 | 30 |
| 0.600000000000000 | 29 |
| 0.620000000000000 | 27 |
| 0.640000000000000 | 26 |
| 0.660000000000000 | 25 |
| 0.680000000000000 | 23 |
| 0.700000000000000 | 22 |
| 0.720000000000000 | 21 |
| 0.740000000000000 | 20 |
| 0.760000000000000 | 19 |
| 0.780000000000000 | 18 |
| 0.800000000000000 | 18 |
| 0.820000000000000 | 17 |
| 0.840000000000000 | 16 |
| 0.860000000000000 | 15 |
| 0.880000000000000 | 15 |
| 0.900000000000000 | 14 |
| 0.920000000000000 | 13 |
| 0.940000000000000 | 13 |
| 0.960000000000000 | 12 |
| 0.980000000000000 | 12 |
| 1 | 11 |
| 1.02000000000000 | 11 |
| 1.04000000000000 | 10 |
| 1.06000000000000 | 10 |
| 1.08000000000000 | 9 |
| 1.10000000000000 | 9 |
| 1.12000000000000 | 9 |
| 1.14000000000000 | 9 |
| 1.16000000000000 | 8 |
| 1.18000000000000 | 8 |
| 1.20000000000000 | 8 |
| 1.22000000000000 | 8 |
| 1.24000000000000 | 9 |
| 1.26000000000000 | 9 |
| 1.28000000000000 | 9 |
| 1.30000000000000 | 9 |
| 1.32000000000000 | 9 |
| 1.34000000000000 | 10 |
| 1.36000000000000 | 11 |
| 1.38000000000000 | 11 |
| 1.40000000000000 | 11 |
| 1.42000000000000 | 12 |
| 1.44000000000000 | 12 |
| 1.46000000000000 | 12 |
| 1.48000000000000 | 14 |
| 1.50000000000000 | 14 |
| 1.52000000000000 | 15 |
| 1.54000000000000 | 15 |
| 1.56000000000000 | 17 |
| 1.58000000000000 | 18 |
| 1.60000000000000 | 18 |
| 1.62000000000000 | 20 |
| 1.64000000000000 | 21 |
| 1.66000000000000 | 23 |
| 1.68000000000000 | 26 |
| 1.70000000000000 | 27 |
| 1.72000000000000 | 31 |
| 1.74000000000000 | 34 |
| 1.76000000000000 | 39 |
| 1.78000000000000 | 45 |
| 1.80000000000000 | 52 |
| 1.82000000000000 | 63 |
| 1.84000000000000 | 84 |
| 1.86000000000000 | 118 |
| 1.88000000000000 | 204 |
| 1.90000000000000 | 795 |
| 1.92000000000000 | 无法收敛！ |
| 1.94000000000000 |
| 1.96000000000000 |
| 1.98000000000000 |

最佳的松弛因子在[1.16-1.22]间，当再降低误差允许范围时，可以发现是1.18.

解向量为：

**P155/39上机题（用MATLAB编写）**

1. 求第一型三次样条插值函数的通用程序

程序如下：

x=[1,2,4,5];

lex=length(x);

y=[1,3,4,2];

h = [0];

miu = [0];

dy = [1,-1]; % 第一型条件，分别为区间两侧的一阶导数值

chashang2 = [0]; %1阶差商

chashang3 = [0]; %2阶差商

% 求参数lamda,miu

for i=1:lex-1

h(i)=x(i+1)-x(i);

end

for i=1:length(h)-1

miu(i)=h(i)/(h(i)+h(i+1));

end

lamda = 1 -miu;

% 求1阶差商

for i=1:lex-1

chashang2(i)=(y(i+1)-y(i))/(x(i+1)-x(i));

end

% 求2阶差商

chashang3(1)=(chashang2(1)-dy(1)) / (x(2)-x(1)) ;

chashang3(lex)=(dy(2)-chashang2(lex-1)) / (x(lex)-x(lex-1)) ;

for i=2:lex-1

chashang3(i)=(chashang2(i)-chashang2(i-1)) / (x(i+1)-x(i-1));

end

d=6\*chashang3; % D矩阵求得！

% 定义A矩阵

A(1,1)=2;

A(1,2)=1;

A(lex,lex)=2;

A(lex,lex-1)=1;

for i=2:lex-1

A(i,i-1)=miu(i-1);

A(i,i)=2;

A(i,i+1)=lamda(i-1);

end

% A矩阵构建完毕 AM=d,求M即可！

M=A\(d');

k=input('请输入要求的点x=：\n');

% 确定输入值的上下界

for i=1:lex

if x(i)>k

index1=i-1;

break;

end

end

sx=y(index1)+( chashang2(index1)- ( (1/3)\*M(index1)+(1/6)\*M(index1+1) )\* h(index1) )\*(k-x(index1));

sx=sx+(1/2)\*(M(index1)\*(k-x(index1)))^2;

sx=sx+(1/(6\*h(index1))) \*(M(index1+1)-M(index1))\*(k-x(index1))^3;

sx

1. 求汽车门曲线型值点的三次样条插值函数，并打印出

**程序如下：**

x=[0,1,2,3,4,5,6,7,8,9,10];

lex=length(x);

y=[2.51,3.30,4.04,4.70,5.22,5.54,5.78,5.40,5.57,5.70,5.80];

h = [0];

miu = [0];

dy = [0.8,0.2]; % 第一型条件，分别为区间两侧的一阶导数值

chashang2 = [0]; %1阶差商

chashang3 = [0]; %2阶差商

% 求参数lamda,miu

for i=1:lex-1

h(i)=x(i+1)-x(i);

end

for i=1:length(h)-1

miu(i)=h(i)/(h(i)+h(i+1));

end

lamda = 1 -miu;

% 求1阶差商

for i=1:lex-1

chashang2(i)=(y(i+1)-y(i))/(x(i+1)-x(i));

end

% 求2阶差商

chashang3(1)=(chashang2(1)-dy(1)) / (x(2)-x(1)) ;

chashang3(lex)=(dy(2)-chashang2(lex-1)) / (x(lex)-x(lex-1)) ;

for i=2:lex-1

chashang3(i)=(chashang2(i)-chashang2(i-1)) / (x(i+1)-x(i-1));

end

d=6\*chashang3; % D矩阵求得！

% 定义A矩阵

A(1,1)=2;

A(1,2)=1;

A(lex,lex)=2;

A(lex,lex-1)=1;

for i=2:lex-1

A(i,i-1)=miu(i-1);

A(i,i)=2;

A(i,i+1)=lamda(i-1);

end

% A矩阵构建完毕 AM=d,求M即可！

M=A\(d');

p=1;

% 确定输入值的上下界

for r=0.5:9.5

for i=1:lex

if x(i)>r

index1=i-1;

break;

end

end

sx=y(index1)+( chashang2(index1)- ( (1/3)\*M(index1)+(1/6)\*M(index1+1) )\* h(index1) )\*(r-x(index1));

sx=sx+(1/2)\*(M(index1)\*(r-x(index1)))^2;

sx=sx+(1/(6\*h(index1))) \*(M(index1+1)-M(index1))\*(r-x(index1))^3;

sxall(p)=sx;

p=p+1;

end

sxall

汇总如下：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *S(0.5)* | *S(1.5)* | *S(2.5)* | *S(3.5)* | *S(4.5)* | *S(5.5)* | *S(6.5)* | *S(7.5)* | *S(8.5)* | *S(9.5)* |
| 2.9089 | 3.6855 | 4.3923 | 5.0023 | 5.4419 | 5.6976 | 5.9749 | 5.4679 | 5.7215 | 5.7366 |

**第五章上机题（用MATLAB编写）**

用*Romberg*积分算法估计，

程序如下：

i=1; %循环次数

n=1; %等分区间 2^(i-1)=n

h=1-(-1);

eps=0.5e-7;

while i>0

T(i)=( (h/2^(i-1))/2 ) \* (fx(-1)+ fx(1));

hh=2/2^(i-1);

n=2^(i-1) ;

for k=-1+hh:hh:1-hh

T(i)=T(i)+ ( (h/2^(i-1))/2 ) \* (2\*fx(k));

end

if(i>1)

S(i-1)=(4/3)\*T(i)-(1/3)\*T(i-1);

end

if(i>2)

C(i-2)=(16/15)\*S(i-1)-(1/15)\*S(i-2);

end

if(i>3)

R(i-3)=(64/63)\*C(i-2)-(1/63)\*C(i-3);

end

if(i>4)

if abs( R(i-3)-R(i-4) )/255 <eps

break;

end

end

i=i+1;

end

fprintf('romberg估计值为%6.7f\n',R(i-3))

得到如下结果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **n** | ***T(f)*** | ***S(f)*** | ***C(f)*** | ***R(f)*** |
| **1** | 0.01980198 | 1.33993399 | 0.32444782 | 0.27711803 |
| **2** | 1.0099009 | 0.387915714 | 0.27785756 | 0.28111937 |
| **4** | 0.54341203 | 0.284736200 | 0.28106840 | 0.29385002 |
| **8** | 0.349405158 | 0.281297643 | 0.29365031 | 0.29431613 |
| **16** | 0.298324521 | 0.292878268 | 0.29430573 | 0.29422487 |
| **32** | 0.294239831 | 0.294216516 | 0.29422613 | 0.29422552 |
| **64** | 0.294222344 | 0.294225534 | 0.29422553 |  |
| **128** | 0.294224737 | 0.294225534 |  |  |
| **256** | 0.294225335 |  |  |  |

故。

**第六章上机题（用MATLAB编写）**

程序如下：

函数：

function [output] = fx(x,y)

output=-x^2\*y^2;

end

RK4：

% #RK4

xmin=0; %定义x的取值范围

xmax=1.5;

h=0.1; %区间大小

y(1)=3;%给定y初值

yi=y(1);%设定xi,yi初值

xi=xmin;

i=0;

while(xi<=xmax)

k1=fx(xi,yi);

k2=fx(xi+0.5\*h,yi+0.5\*h\*k1);

k3=fx(xi+0.5\*h,yi+0.5\*h\*k2);

k4=fx(xi+h,yi+h\*k3);

y(i+1)=yi+(h/6)\*(k1+2\*k2+2\*k3+k4);

yi=y(i+1);

i=i+1;

xi=xi+h;

end

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **x** | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 |
| **结果** | 2.9970028 | 2.9761900 | 2.9211287 | 2.8195472 | 2.6666634 | 2.4671002 | 2.2337991 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0.8 | 0.9 | 1.0 | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 |
| 1.9841228 | 1.7351071 | 1.5000058 | 1.2870125 | 1.0997221 | 0.93839745 | 0.80130042 | 0.68573208 |

AB4(Adams显式)

% #AB4

xmin=0; %定义x的取值范围

xmax=1.5;

h=0.1; %区间大小

y(1)=3;%给定y初值

yi=y(1);%设定xi,yi初值

xi=xmin;

i=0;

while(i<=3)

k1=fx(xi,yi);

k2=fx(xi+0.5\*h,yi+0.5\*h\*k1);

k3=fx(xi+0.5\*h,yi+0.5\*h\*k2);

k4=fx(xi+h,yi+h\*k3);

y(i+1)=yi+(h/6)\*(k1+2\*k2+2\*k3+k4);

yi=y(i+1);

i=i+1;

xi=xi+h;

end

while(i>=4 && xi<=xmax)

k1=fx(xi,y(i));

k2=fx(xi-h,y(i-1));

k3=fx(xi-2\*h,y(i-2));

k4=fx(xi-3\*h,y(i-3));

y(i+1)=yi+(h/24)\*(55\*k1-59\*k2+37\*k3+-9\*k4);

yi=y(i+1);

i=i+1;

xi=xi+h;

end

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **x** | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 |
| **结果** | 2.9970028 | 2.9761900 | 2.9211287 | 2.8195472 | 2.6655910 | 2.4660975 | 2.2337496 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0.8 | 0.9 | 1.0 | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 |
| 1.9855258 | 1.7374454 | 1.5025299 | 1.2889758 | 1.1009128 | 0.93881198 | 0.80124536 | 0.68537731 |

AB4-AM4(Adams预测校正)

% #AB4-AM4

xmin=0; %定义x的取值范围

xmax=1.5;

h=0.1; %区间大小

y(1)=3;%给定y初值

yi=y(1);%设定xi,yi初值

xi=xmin;

i=0;

while(i<=3)

k1=fx(xi,yi);

k2=fx(xi+0.5\*h,yi+0.5\*h\*k1);

k3=fx(xi+0.5\*h,yi+0.5\*h\*k2);

k4=fx(xi+h,yi+h\*k3);

y(i+1)=yi+(h/6)\*(k1+2\*k2+2\*k3+k4);

yi=y(i+1);

i=i+1;

xi=xi+h;

end

while(i>=4 && xi<=xmax)

k1=fx(xi,y(i));

k2=fx(xi-h,y(i-1));

k3=fx(xi-2\*h,y(i-2));

k4=fx(xi-3\*h,y(i-3));

y(i+1)=yi+(h/24)\*(55\*k1-59\*k2+37\*k3+-9\*k4);

k5=fx(xi+h,y(i+1));

y(i+1)=yi+(h/24)\*(9\*k5+19\*k1-5\*k2+k3); %预测-校正

yi=y(i+1);

i=i+1;

xi=xi+h;

end

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **x** | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 |
| **结果** | 2.9970028 | 2.9761900 | 2.9211287 | 2.8195472 | 2.6667593 | 2.4671523 | 2.2336497 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0.8 | 0.9 | 1.0 | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 |
| 1.9837222 | 1.7345581 | 1.4994791 | 1.2866300 | 1.0995133 | 0.93832808 | 0.80131683 | 0.68578839 |

改进AB4-AM4(使用Richardson外推改进Adams预测校正)

% #AB4-AM4++

xmin=0; %定义x的取值范围

xmax=1.5;

h=0.1; %区间大小

y(1)=3;%给定y初值

yi=y(1);%设定xi,yi初值

xi=xmin;

i=0;

while(i<=3)

k1=fx(xi,yi);

k2=fx(xi+0.5\*h,yi+0.5\*h\*k1);

k3=fx(xi+0.5\*h,yi+0.5\*h\*k2);

k4=fx(xi+h,yi+h\*k3);

y(i+1)=yi+(h/6)\*(k1+2\*k2+2\*k3+k4);

yi=y(i+1);

i=i+1;

xi=xi+h;

end

while(i>=4 && xi<=xmax)

k1=fx(xi,y(i));

k2=fx(xi-h,y(i-1));

k3=fx(xi-2\*h,y(i-2));

k4=fx(xi-3\*h,y(i-3));

yp=yi+(h/24)\*(55\*k1-59\*k2+37\*k3+-9\*k4);

k5=fx(xi+h,yp);

y(i+1)=yi+(h/24)\*(9\*k5+19\*k1-5\*k2+k3);

y(i+1)=(251/270)\*y(i+1)+(19/270)\*yp; %改进

yi=y(i+1);

i=i+1;

xi=xi+h;

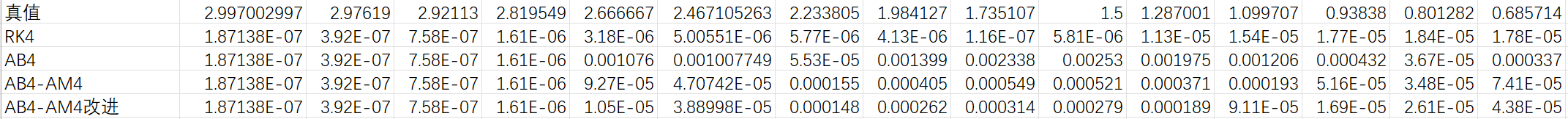
end

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **x** | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 |
| **结果** | 2.9970028 | 2.9761900 | 2.9211287 | 2.8195472 | 2.6666771 | 2.4670663 | 2.2336573 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0.8 | 0.9 | 1.0 | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 |
| 1.9838648 | 1.7347928 | 1.4997205 | 1.2868123 | 1.0996156 | 0.93836287 | 0.80130811 | 0.68575807 |

对比各方法：

各方法的局部截断误差如下：



可见，对AB4进行改进后，误差得到了显著减少，AB4-AM4结合了显式方法的方便性和隐式方法的精度更高的特点，而Richardson外推使得精度更高了一阶。RK4方法的精度最高。