姓名 唐超

班级 智能16

学号 201600181073

数值计算实验报告

Chapter 1 误差理论

Problem 1

构造算法和Matlab程序，以便精确计算所有情况下的二次方程的根，包括 的情况。

Answer

对于一元二次方程 ，当 时，为避免其值过小引起巨量消失，从而带来精度损失。若 ，用下列公式求根：



若 ，则应该用下列公式求根：



因此精确求解所有情况下二次方程的根的代码如下：

Code

function [x1,x2] = NM\_1\_1(a,b,c)

if b\*b-4\*a\*c>=0

if b>=0

x1 = -2\*c/(b+sqrt(b\*b-4\*a\*c));

x2 = (-b-sqrt(b\*b-4\*a\*c))/2\*a;

else

x1 = (-b+sqrt(b\*b-4\*a\*c))/2\*a;

x2 = -2\*c/(b-sqrt(b\*b-4\*a\*c));

end

else

disp('no roots');

end

Problem 2

对下列3个差分方程计算出前10个数值近似值。在每种情况下引入一个小的初始误差，如果没有初始误差，则每个差分方程将生成序列 ，用表和图的形式分别输出结果。

(a) 

(b) 

(c) 

Answer

输出结果如下

表 1.1 序列以及近似值和

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n |  |  |  |  |
| 0 | 1.0000000000 | 0.9940000000 | 1.0000000000 | 1.0000000000 |
| 1 | 0.5000000000 | 0.4970000000 | 0.4970000000 | 0.4970000000 |
| 2 | 0.2500000000 | 0.2485000000 | 0.2455000000 | 0.2425000000 |
| 3 | 0.1250000000 | 0.1242500000 | 0.1197500000 | 0.1092500000 |
| 4 | 0.0625000000 | 0.0621250000 | 0.0568750000 | 0.0276250000 |
| 5 | 0.0312500000 | 0.0310625000 | 0.0254375000 | 0.0506875000 |
| 6 | 0.0156250000 | 0.0155312500 | 0.0097187500 | 0.1835937500 |
| 7 | 0.0078125000 | 0.0077656250 | 0.0018593750 | 0.4844218750 |
| 8 | 0.0039062500 | 0.0038828125 | 0.0020703125 | 1.2207734375 |
| 9 | 0.0019531250 | 0.0019414063 | 0.0040351562 | 3.0537929688 |
| 10 | 0.0009765625 | 0.0009707031 | 0.0050175781 | 7.6324121094 |

表 1.2 误差序列和

|  |  |  |  |
| --- | --- | --- | --- |
| n |  |  |  |
| 0 | 0.0060000000 | 0.0000000000 | 0.0000000000 |
| 1 | 0.0030000000 | 0.0030000000 | 0.0030000000 |
| 2 | 0.0015000000 | 0.0045000000 | 0.0075000000 |
| 3 | 0.0007500000 | 0.0052500000 | 0.0157500000 |
| 4 | 0.0003750000 | 0.0056250000 | 0.0348750000 |
| 5 | 0.0001875000 | 0.0058125000 | 0.0819375000 |
| 6 | 0.0000937500 | 0.0059062500 | 0.1992187500 |
| 7 | 0.0000468750 | 0.0059531250 | 0.4922343750 |
| 8 | 0.0000234375 | 0.0059765625 | 1.2246796875 |
| 9 | 0.0000117188 | 0.0059882812 | 3.0557460938 |
| 10 | 0.0000058594 | 0.0059941406 | 7.6333886719 |



图 1.1 误差序列



图 1.2 误差序列



图 1.3 误差序列

可以看出，的误差是稳定的，且按指数级递减。的误差也是稳定的。的误差是不稳定的，且按指数级增长。

Chapter 2 非线性方程求根

Problem 1

利用牛顿法求解方程



分别取，使得精度不超过，比较初值对计算结果的影响。

牛顿-拉夫森定理

设，且，满足。若，，对任意初始近似值，使得由如下定义的序列收敛到：



Answer

在不同初值下，牛顿迭代结果如下：

表 2.1 不同初值求解结果

|  |  |  |  |
| --- | --- | --- | --- |
| 初值 |  |  |  |
| 零点 | 1.89249 | 1.89279 | -7647350975.56381 |
| 零点函数值 | 0.000006 | 0.000005 | 14620494228936600000 |
| 误差 | 0.003026 | 0.002722 | 2227547899 |
| 迭代次数 | 6 | 10 | 10000 |

可以看出，当初值时，分别经过6次和10次迭代，零点结果已经满足精度要求。当初值，此时初值偏离真实零点较大，造成迭代序列已经不收敛于真实零点，经过最大迭代次数1000次迭代后，零点为-7647350975.56381，显然是错误的。

上面的例子说明了牛顿法对于初值的选择非常敏感，当初值偏离真实零点过远时，会造成迭代序列发散。

**Code：**

a = [pi/2 5\*pi 10\*pi];

P0 = zeros(1,length(a));

Err = zeros(1,length(a));

I = zeros(1,length(a));

Y = zeros(1,length(a));

for k = 1:length(a)

[p0,err,i,y] = NM\_2\_1NEWTON('f','df',a(k),10^-5,10^-5,10000);

P0(k) = p0;

Err(k) = err;

I(k) = i;

Y(k) = y;

End

function [p0,err,i,y] = newton(f,df,p0,delta,epsilon,max\_iteration)

% delta,epsilon are the tolerances of p0 and y respectively.

for i = 1:max\_iteration

p1 = p0 - feval(f,p0)/feval(df,p0);

err = abs(p0-p1);

p0 = p1;

y = feval(f,p0);

if (err<delta)||(abs(y)<epsilon)

break

end

end

end

Problem 2

已知



在上有一个实根，试分别利用二分法、牛顿法、割线法、错位法设计相应的计算格式，并编程求解（精确到4位小数）。

二分法

根据精度要求计算最大迭代次数max1 =1+round((log(b-a)-log(delta))/log(2))=18，取，令，若，令，若，令，重复以上步骤，直至达到迭代次数。

Answer：

经过18次迭代，得到的一个根，，误差.

Code：

function [c,err,yc] = bisect(f,a,b,delta)

%Input - f is the function input as a string 'f'

% - a and b are the left and right end points

% - delta is the tolerance

%Output - c is the zero

% - yc=f(c)

% - err is the error estimate for c

ya=feval(f,a)

yb=feval(f,b)

if ya\*yb<0

max1 = 1+round((log(b-a)-log(delta))/log(2)); %应向上取整。round为四舍五入取整，为避免可能“四舍”向下取整，前面加1

for k = 1:max1

c = (a+b)/2;

yc = feval(f,c);

if yc==0

a = c;

b = c;

elseif yb\*yc>0

b = c;

yb = yc;

else

a = c;

ya = yc;

end

if b-a<delta

break

end

end

c = (a+b)/2;

err = abs((b-a)/2);

yc = feval(f,c);

else

disp('please enter a,b with f(a)\*f(b)<0');

end

end

错位法

错位法与二分法求解方程算法不同点在于c不再是区间的中点，而是和连线与轴交点，以加快迭代的收敛速率。



Answer

经过13次迭代，得到的一个根，，误差。

Code

function [c,err,yc] = regula(f,a,b,delta,max\_iteration)

%Input - f is the function input as a string 'f'

% - a and b are the left and right end points

% - delta is the tolerance

%Output - c is the zero

% - yc=f(c)

% - err is the error estimate for c

ya=feval(f,a);

yb=feval(f,b);

if ya\*yb<0

for k = 1:max\_iteration

c = b - feval(f,b)\*(b-a)/(feval(f,b)-feval(f,a));

yc = feval(f,c);

if yc==0

a = c;

b = c;

elseif yb\*yc>0

b = c;

yb = yc;

else

a = c;

ya = yc;

end

if b-a<delta

break

end

end

c = b - feval(f,b)\*(b-a)/(feval(f,b)-feval(f,a));

err = max(c-a,b-c);

yc = feval(f,c);

else

disp('please enter a,b with f(a)\*f(b)<0');

end

end

牛顿法

Answer

取初值,经过3次迭代，得到,,误差. 代码如Problem 1中所示。

割线法

割线法包含的公式与错位法的公式一样，只是在关于如何定义每个后续项的逻辑判定上不一样。需要两个靠近真实零点的初始点和，定义为经过两个初始点的直线与轴的交点的横坐标，比或更接近。



根据两点迭代公式可得到一般项为



由此得到一个收敛于真实零点的迭代序列.

Answer

初次取，，经过14次迭代，得到 位于外的另一个零点，，误差。

又取，，经过5次迭代，得到在上的一个零点，，误差。

可以看出，在一个函数有多个根的情况下，初始点的选择决定了迭代序列收敛于哪一个根。

Code

function [p1,err,i,y] = secant(f,p0,p1,delta,epsilon,max\_iteration)

for i = 1:max\_iteration

p2 = p1 - feval(f,p1)\*(p1-p0)/(feval(f,p1)-feval(f,p0));

err = abs(p1-p2);

p0 = p1;

p1 = p2;

y = feval(f,p1);

if (err<delta)||(abs(y)<epsilon)

break

end

end

end

Chapter 3 线性方程组求解

Problem 1

求解线性方程组

1. 试用LU分解求解此方程组

Answer

所求解的方程组可写为 ，其中 ， ， .

经过LU分解， ，因此原方程组可写为 ，不妨设 ，则 ，通过简单的forward substitution解得 ，再将 带入 ，通过backward substitution 解得 .

Code

function [L, U] = LU\_factorization(n, A)

%Input - A is the matrix to be factorized.

% - n is the dimension of matrix A.

%Output - L is a lower-triangular matrix

% - U is a upper-triangular matrix

L = zeros(n,n);

U = zeros(n,n);

for i=1:n

L(i,i) = 1; %L对角线元素为1

end

for j=1:n

U(1,j) = A(1,j)/L(1,1); %first row of U

L(j,1) = A(j,1)/U(1,1); %first colum of L

en

for i=2:n-1

U(i,i) = A(i,i);

for k=1:i-1

U(i,i) = U(i,i) - L(i,k)\*U(k,i);

end

if U(i,i)==0

disp('Factorization impossible');

break;

end

for j=i+1:n

U(i,j) = A(i,j);

L(j,i) = A(j,i);

for k=1:i-1

U(i,j) = U(i,j) - L(i,k)\*U(k,j); % ith row of U

L(j,i) = L(j,i) - L(j,k)\*U(k,i); % ith column of L

end

L(j,i) = L(j,i)/U(i,i);

end

end

U(n,n) = A(n,n);

for k=1:n-1

U(n,n) = U(n,n) - L(n,k)\*U(k,n);

end

end

1. 分别用Jacobi, Gauss-Seidel方法求解此方程组

由于系数矩阵A是严格对角占优的，因此此方程组可以用Jacobi, Gauss-Seidel方法求解。

采用Jacobi方法，设 ，初始 ，经过11次迭代，相对误差 ，此时 为满足tol下的方程组的解。

Jacobi迭代的代码如下

function [k,err,relerr,X] = Jacobi(A, b, P, tol, max)

% Input - P is the initial guess.

% - tol is the tolerance for P.

% - max is the maximum number of iterations

% Output - X is the approximation to the solution of AX=B

n = length(b);

X = zeros(n,1);

for k=1:max

for i=1:n

X(i) = (b(i)-A(i,[1:i-1,i+1:n])\*P([1:i-1,i+1:n]))/A(i,i);

end

err = abs(norm(X-P));

relerr = err/(norm(X)+eps);

P = X;

if(err<tol)||(relerr<tol)

break;

end

end

采用Gauss-Seidel方法，仍设 ，初始 ，仅经过7次迭代，相对误差 ，此时 为满足tol下的方程组的解。两种迭代求解方法结果一致，但显然Gauss-Seidel方法的效率更高。

Gauss-Seidel方法的代码如下

function [k,err,relerr,X] = Gauss\_Seidel(A, b, P, tol, max)

% Input - P is the initial guess.

% - tol is the tolerance for P.

% - max is the maximum number of iterations

% Output - X is the approximation to the solution of AX=B

n = length(b);

X = zeros(n,1);

for k=1:max

for i=1:n

if i==1

X(1) = (b(1)-A(1,2:n)\*P(2:n))/A(1,1);

elseif i==n

X(n) = (b(n)-A(n,1:n-1)\*X(1:n-1))/A(n,n);

else

X(i) = (b(i)-A(i,1:i-1)\*X(1:i-1)-A(i,i+1:n)\*P(i+1:n))/A(i,i);

end

end

err = abs(norm(X-P));

relerr = err/(norm(X)+eps);

P = X;

if(err<tol)||(relerr<tol)

break

end

end

end

Problem 2

设有如下三角线性方程组，而且系数矩阵具有严格对角优势：



其中系数矩阵, ，

1. 设计一个算法来求解上述方程组，算法必须有效利用系数矩阵的稀疏性。

Answer

为有效利用系数矩阵的稀疏性，不再将A整体作为高斯-赛德尔迭代法的输入，而是仅输入每个方程的2或3个系数。

**Algorithm**

Input:

the number of equations and unknowns  ;

the entries   of  ;

the entries  of  ;

an initial approximation  ;

tolerance  ;

maximum number of iterations max1.

Output:

the approximate solution  or a message that the number of iterations was exceeded.

Step1: Set k = 1.

Step2: While (), do steps 3~5.

Step3: Set .

For i = 2:N-1, set .

Set .

Step4: If , then output . STOP.

Step5: Set .

Step6: Output (‘Maximum number of iterations exceeded’).

(2) 根据(1)中设计的算法构造一个MATLAB程序，并求解下列三角线性方程组。

(a) , .

**Answer**

将作为参数带入（1）中的算法，初始， ，经过6次迭代，，得到方程组的解

 （“…”代表的值均为0.5000）.

(b) 为奇数，为偶数, ， .

**Answer**

将作为参数带入（1）中的算法，初始， ，经过6次迭代，，得到方程组的解

（“…”代表的值均为0.3334）.

**Code**

max = 100; %the maximum number of iterations

n = 50;

a = ones(n-1,1);

b = ones(n,1);

for i=2:50

b(i)=2;

end

c = ones(n-1,1);

d = 4\*ones(n,1);

P = ones(n,1); %the initial guess

X = zeros(n,1);

max = 1000; %the maximum number of iterations

tol = 10^-3; %tol is the tolerance for P

for k=1:max

for i=1:n

if i==1

X(1) = (b(1)-c(1)\*P(2))/d(1);

elseif i==n

X(n) = (b(n)-a(n-1)\*X(n-1))/d(n);

else

X(i) = (b(i)-a(i-1)\*X(i-1)-c(i)\*P(i+1))/d(i);

end

end

err = abs(norm(X-P));

relerr = err/(norm(X)+eps);

P = X;

if(err<tol)||(relerr<tol)

break

end

end

Problem 3

利用高斯-赛德尔迭代求解带状方程组。

**Answer**

该带状方程组可写为：

其中， ，.

系数矩阵A是具有严格对角优势的，因此可以用高斯-赛德尔迭代法求解该方程组。

**Answer**

采用3.2中的算法求解，只对Step3做如下修改：

Step3: Set .

For i = 3:N-2, set .

Set .

初始， ，经过4次迭代，，得到方程组的解：

（“…”代表的值均为0.5000）.

Chapter 4 插值多项式

**Problem 1**

在区间上，生成11个等距插值节点在相应插值节点上计算函数



的函数值作为观测值，

1. 利用这11个数据点，生成一个10次拉格朗日插值多项式，并做出插值函数与原函数的对比结果图。
2. 利用此多项式近似计算



与解析解比较，分析误差产生的原因。

1. 利用构造分片线性插值多项式，并利用此分片插值多项式近似计算积分



与解析解比较，分析误差产生的原因。

1. 若希望提高积分的计算精度，试提出你自己的建议，并进行实验测试验证。

拉格朗日插值多项式

对于个插值点，有次拉格朗日插值多项式：



其中为基函数，满足.

，其中为拉格朗日余项，用于误差估计。

分片线性插值

对于个插值点，有分片线性插值公式：



其中



Answer

1. 生成的10次拉格朗日插值多项式为

 



图 4.1 10次拉格朗日插值多项式插值结果

1. 经计算



相对误差.

误差原因分析：由于所取插值点较少而插值多项式的次数较大，在差值区间的边界部分插值函数会出现很大的波动，明显偏离原始函数（龙格现象）；另外，从舍入误差看，高次插值由于计算量大，可能造成严重的误差累计。

Code

%计算插值多项式并绘制对比结果

x = linspace(-5,5,11);

y = 1./(1+x.^2);

x1=-5:0.1:5;

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

plot(x1,y1,'r');

w = length(x);

n = w-1;

L = zeros(w,w);

for k=1:n+1

V=1;

for j=1:n+1

if j~=k

V=conv(V,poly(x(j)))/(x(k)-x(j));

end

end

L(k,:)=V;

end

C = y\*L;

x2=-5:0.1:5;

p = zeros(11,length(x2));

for i=1:length(x2)

for j=1:11

p(j,i)=x2(i)^(11-j);

end

end

y2 = C\*p;

hold on

plot(x2,y2,'b')

legend("y(x)", "P\_1\_0(x)")

%计算[-5,5]上的积分值

f1 = @(xx) 1./(1+xx.^2);

I1 = integral(f1,-5,5)

q = polyint(C);

I2 = diff(polyval(q,[-5,5]))

1. 利用构造的分片线性插值结果如下



图 4.2 分片线性插值结果

经计算，与的相对误差为



相比上述10次插值多项式的积分结果极大地减小了误差。

误差原因分析：一般分片线性插值多项式在插值点都不是连续的，即插值函数不够平滑。当插值点较多时，可以减小因此而造成的误差。

Code

%插值节点

x = linspace(-5,5,11);

y = 1./(1+x.^2);

x1=-5:0.1:5;

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

plot(x1,y1,'r');

% 计算每个子插值区间的线性插值多项式，并绘图

V = zeros(10,2);

for i=1:10

V(i,1)=y(i+1)-y(i);

V(i,2)=y(i)\*x(i+1)-y(i+1)\*x(i);

hold on

plot(x(i:i+1),V(i,1).\*x(i:i+1)+V(i,2),'b')

end

legend("y(x)", "P(x)")

%计算积分

sumI = 0;

for k=1:10

q = polyint(V(k,:));

sumI = sumI+diff(polyval(q,[k-6,k-5]));

end

1. 我的建议：在区间上生成更多的等距插值节点作分段线性插值并计算该积分值，以提高积分的计算精度。

采用21个插值节点的结果如下：



图4.3 21个插值节点

此时，，相对误差进一步减小为.

Chapter 5 最小二乘拟合

Problem 1

根据下列数据，使用幂曲线拟合求解重力常量g。

1. (b)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 时间 | 距离 |  | 时间 | 距离 |
| 0.200 | 0.1960 |  | 0.200 | 0.1965 |
| 0.400 | 0.7835 |  | 0.400 | 0.7855 |
| 0.600 | 1.7630 |  | 0.600 | 1.7675 |
| 0.800 | 3.1345 |  | 0.800 | 3.1420 |
| 1.000 | 4.8975 |  | 1.000 | 4.9095 |

幂函数拟合

设有N个点, 其中横坐标时确定的。最小二乘幂函数拟合曲线的系数A为



Answer:

经过最小二乘拟合得到：，从而；，从而。

Code

%幂函数拟合y=A\*x^M

function A = chap5\_1(x,y,M)

%x,y均为列向量

A = ((x.^M)'\*y)/((x.^M)'\*(x.^M));

end

t = [0.200 0.400 0.600 0.800 1.000]';

d1 = [0.1960 0.7835 1.7630 3.1345 4.8975]';

d2 = [0.1965 0.7855 1.7675 3.1420 4.9095]';

g1 = chap5\_1(t,d1,2)\*2;

g2 = chap5\_1(t,d2,2)\*2;

Problem 2

根据点，求5种不同的三次样条插值，其中。在同一坐标系中，画出这5个三次样条插值和这些数据点。

三次样条插值

定义：设有N个点，其中。如果存在N个三次多项式，系数为和，满足如下性质：











则称函数为三次样条函数。

根据三次样条函数的定义，其满足下列关系式：



其中。

根据不同的端点约束，可以构造以下5中不同的样条：

* **紧压样条**：存在唯一的三次样条曲线，其一阶导数边界条件是
* **natural样条：**存在唯一的三次样条曲线，。
* **外推样条：**存在唯一的三次样条曲线，其中通过对点和进行外推得到，同时通过对点和进行外推得到。
* **抛物线终结样条：**存在唯一的三次样条曲线，其中在内，而在内。
* **端点曲率调整样条：**存在唯一的三次样条曲线，其中二阶导数的边界条件和是确定的。

Answer



从图中可以看出，5种样条在上几乎重合，另外，紧压样条曲线与其他样条曲线相差较远，端点曲率调整样条与外推样条最为接近。

Code

function S = spline(method,X,Y,dx0,dxn)

N=length(X)-1;

H=diff(X);

D=diff(Y)./H;

A=H(2:N-1);

B=2\*(H(1:N-1)+H(2:N));

C=H(2:N);

U=6\*diff(D);

%clamped样条

if method=='cl'

B(1)=B(1)-H(1)/2;

U(1)=U(1)-3\*(D(1)-dx0);

B(N-1)=B(N-1)-H(N)/2;

U(N-1)=U(N-1)-3\*(dxn-D(N));

for k=2:N-1

temp=A(k-1)/B(k-1);

B(k)=B(k)-temp\*C(k-1);

U(k)=U(k)-temp\*U(k-1);

end

M(N)=U(N-1)/B(N-1);

for k=N-2:-1:1

M(k+1)=(U(k)-C(k)\*M(k+2))/B(k);

end

M(1)=3\*(D(1)-dx0)/H(1)-M(2)/2;

M(N+1)=3\*(dxn-D(N))/H(N)-M(N)/2;

end

%natural样条

if method=='na'

for k=2:N-1

temp=A(k-1)/B(k-1);

B(k)=B(k)-temp\*C(k-1);

U(k)=U(k)-temp\*U(k-1);

end

M(N)=U(N-1)/B(N-1);

for k=N-2:-1:1

M(k+1)=(U(k)-C(k)\*M(k+2))/B(k);

end

M(1)=0;

M(N+1)=0;

end

%extrapolated样条

if method=='ex'

B(1)=B(1)+H(1)+(H(1))^2/H(2);

B(N-1)=B(N-1)+H(N)+(H(N))^2/H(N-1);

A(N-2)=(H(N-1)-(H(N))^2/H(N-1));

C(1)=H(2)-(H(1))^2/H(2);

for k=2:N-1

temp=A(k-1)/B(k-1);

B(k)=B(k)-temp\*C(k-1);

U(k)=U(k)-temp\*U(k-1);

end

M(N)=U(N-1)/B(N-1);

for k=N-2:-1:1

M(k+1)=(U(k)-C(k)\*M(k+2))/B(k);

end

M(1)=M(2)-H(1)\*(M(3)-M(2))/H(2);

M(N+1)=M(N)+H(N)\*(M(N)-M(N-1))/H(N-1);

end

%抛物线终结样条

if method=='pt'

B(1)=B(1)+H(1);

B(N-1)=B(N-1)+H(N);

for k=2:N-1

temp=A(k-1)/B(k-1);

B(k)=B(k)-temp\*C(k-1);

U(k)=U(k)-temp\*U(k-1);

end

M(N)=U(N-1)/B(N-1);

for k=N-2:-1:1

M(k+1)=(U(k)-C(k)\*M(k+2))/B(k);

end

M(1)=M(2);

M(N+1)=M(N);

end

%端点曲率调整样条

if method=='ep'

U(1)=U(1)-H(1)\*dx0;

U(N-1)=U(N-1)-H(N)\*dxn;

for k=2:N-1

temp=A(k-1)/B(k-1);

B(k)=B(k)-temp\*C(k-1);

U(k)=U(k)-temp\*U(k-1);

end

M(N)=U(N-1)/B(N-1);

for k=N-2:-1:1

M(k+1)=(U(k)-C(k)\*M(k+2))/B(k);

end

M(1)=dx0;

M(N+1)=dxn;

end

S=zeros(N,4);

for k=0:N-1

S(k+1,1)=(M(k+2)-M(k+1))/(6\*H(k+1));

S(k+1,2)=M(k+1)/2;

S(k+1,3)=D(k+1)-H(k+1)\*(2\*M(k+1)+M(k+2))/6;

S(k+1,4)=Y(k+1);

end

end

clear,clc

x=[0 1 2 3 4 5 6];

y=[1 0 0 1 2 2 1];

scatter(x,y,'o','filled')

%依次做紧压、natural、外推、抛物线终结、端点曲率调整样条

S1=spline('cl',x,y,-0.6,-1.8);

S2=spline('na',x,y,-0.6,-1.8);

S3=spline('ex',x,y,-0.6,-1.8);

S4=spline('pt',x,y,-0.6,-1.8);

S5=spline('ep',x,y,-0.6,-1.8);

for i=1:6

x1=i-1:0.01:i;

y1=polyval(S1(i,:),x1-x(i));

y2=polyval(S2(i,:),x1-x(i));

y3=polyval(S3(i,:),x1-x(i));

y4=polyval(S4(i,:),x1-x(i));

y5=polyval(S5(i,:),x1-x(i));

hold on

plot(x1,y1,'k',x1,y2,'r',x1,y3,'g',x1,y4,'y',x1,y5,'m');

end

legend('观测点','紧压样条','natural样条','外推样条','抛物线终结样条','端点曲率调整样条')

Chapter 6 数值微分

Problem

求解函数 在处的导数近似值，精度为小数点后13位。

Answer

使用极限的微分求解，，其中为足够大的正整数。

经计算，当时，求得导数近似值满足精度要求，。

Code

function [L,n]=difflim(f,x,tol)

max1=100;

h=1;

H(1)=h;

D(1)=(feval(f,x+h)-feval(f,x-h))/(2\*h);

E(1)=0;

for n=1:2

h=h/10;

H(n+1)=h;

D(n+1)=(feval(f,x+h)-feval(f,x-h))/(2\*h);

E(n+1)=abs(D(n+1)-D(n));

end

n=2;

while E(n)-E(n+1)>tol&&n<max1

h=h/10;

H(n+2)=h;

D(n+2)=(feval(f,x+h)-feval(f,x-h))/(2\*h);

E(n+2)=abs(D(n+2)-D(n+1));

n=n+1;

end

n=length(D)-1;

L=[H' D'];

end

Chapter 7 数值积分

Problem 1

用组合辛普森公式求习题2中的定积分, 精确到小数点后11位。

1. 
2. 
3. 

Answer

**组合辛普森公式：**通过的个等步长采样点逼近积分



1. , 被积函数, ，则，将相关参数带入组合辛普森公式得到 。
2. , 被积函数，，将相关参数带入组合辛普森公式得到 。
3. , 被积函数，，将相关参数带入组合辛普森公式得到 。

Code

function s=simprl(f,a,b,M)

h=(b-a)/(2\*M);

s1=0; s2=0;

for k=1:M

x=a+h\*(2\*k-1);

s1=s1+feval(f,x);

end

for k=1:M-1

x=a+h\*2\*k;

s2=s2+feval(f,x);

end

s=h\*(feval(f,a)++4\*s1+2\*s2+feval(f,b))/3;

end

Problem 2

修改组合梯形公式，使之可以求只有若干点函数值已知的函数积分。将程序7.1修改为求区间上过个给定点的函数的积分逼近。利用该程序求过点的函数的积分逼近。

组合梯形公式

通过的个等步长采样点逼近积分

.

当不采用等距采样时，设相邻两采样点的距离，从而，非等距采样的组合梯形公式如下：



Answer

按上述公式得到，过点的函数的积分逼近。

Code

function s=fdj\_trap(x,y)

M = length(x)-1;

h = diff(x);

s = 0;

for k=1:M

s = s+h(k)/2\*(y(k)+y(k+1));

end

end

clear,clc

k = 0:13;

x = (k.^2+1).^0.5;

y = k.^(1/3);

s = fdj\_trap(x,y)

Chapter 9 微分方程数值解

Problem 1

**流行病模型。**流行病的数学模型描述如下：设有L个成员的构成的群落，其中有P个感染个体，Q个未感染个体。令表示时刻感染个体的数量。对于温和的疾病，如普通感冒，每个个体保持存活，流行病从感染者传播到未感染者。由于两组间有PQ种可能的接触，的变化率正比于PQ。故该问题可以描述为初值问题：



1. 用和初值条件，计算上的欧拉近似解。
2. 画出(a)中的近似解。
3. 通过求(a)中欧拉方法的纵坐标平均值来估计平均感染个体的数目。
4. 通过用曲线拟合(a)中的数据，并用积分均值定理估计平均感染个体数目。

欧拉方法

通过计算



求上初值问题的近似解。

Answer

由题意，，则用欧拉方法得到近似解如下图所示：



平均感染个体的数目（用上述欧拉近似解纵坐标平均值估计）为。

尝试用前面实验已经编写好的natural三次样条插值方法拟合这些近似解，进而利用积分中值定理计算得到平均感染个体数目为。

Code

function E=euler(f,a,b,ya,M)

h=(b-a)/M;

T=zeros(1,M+1);

Y=zeros(1,M+1);

T=a:h:b;

Y(1)=ya;

for j=1:M

Y(j+1)=Y(j)+h\*feval(f,T(j),Y(j));

end

E=[T' Y'];

End

clear,clc

f1 = @(t,y) 0.00003\*y\*(25000-y);

M = 60/0.2;

E=euler(f1,0,60,250,M);

scatter(E(:,1),E(:,2))

xlabel('t');ylabel('y')

% 估计均值

avg\_y = sum(E(:,2))/301;

Problem 2

考虑一阶积分-常微分方程



1. 在区间，用欧拉方法和以及梯形公式求方程的近似解。
2. 用初值和重复(a)的计算。
3. 在同一坐标系中画出(a)和(b)的近似解。

Answer

3个不同初值的近似解最后都收敛到5.16，如下



Code

clear,clc

h=0.2;

y(1)=2.50;

a=0;b=20;

T(1)=0;

for i=2:101

y(i)=y(i-1)+h\*(1.3\*y(i-1)-0.25\*(y(i-1))^2-0.0001\*y(i-1)\*T(i-1));

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

end

t=a:h:b;

plot(t,y)

hold on

f(1)=2.00;

R(1)=0;

for i=2:101

f(i)=f(i-1)+h\*(1.3\*f(i-1)-0.25\*(f(i-1))^2-0.0001\*f(i-1)\*R(i-1));

R(i)=R(i-1)+h\*(f(i-1)+f(i))/2;

end

plot(t,f)

hold on

z(1)=3.00;

S(1)=0;

for i=2:101 z(i)=z(i-1)+h\*(1.3\*z(i-1)-0.25\*(z(i-1))^2-0.0001\*z(i-1)\*R(i-1));

S(i)=S(i-1)+h\*(z(i-1)+z(i))/2;

end

plot(t,z);

Problem 3

用休恩方法求解微分方程



1. 令程序9.2执行20步，然后令，程序9.2执行40步。
2. 比较(a)中的两个近似解与精确解。
3. 当减半时，(a)中的最终全局误差是否和预期相符？
4. 将两个近似解和精确解画在同一坐标系中。

休恩方法

通过计算

，

求上的初值问题的近似解。

Answer

精确解。

当时，得到在处的近似解，误差。

当时，得到在处的近似解，误差。

可以看出，当步长减半时，误差大约减小1/4，与预期相符。

两个近似解和精确解绘制如下：



Code

function H=heum(f,a,b,ya,h)

M=(b-a)/h;

Y=zeros(1,M+1);

T=a:h:b;

Y(1)=ya;

for j=1:M

k1 = feval(f,T(j),Y(j));

k2 = feval(f,T(j+1),Y(j)+h\*k1);

Y(j+1)=Y(j)+h/2\*(k1+k2);

end

H=[T' Y'];

End

clear,clc

f1 = @(t,y) 3\*(y+t);

h1=0.1; h2=0.05;

H1=heum(f1,0,2,1,h1);

H2=heum(f1,0,2,1,h2);

t = 0:0.1:2;

y = 4/3\*exp(3\*t)-t-1/3;

plot(H1(:,1),H1(:,2),'r',H2(:,1),H2(:,2),'b',t,y,'k')

legend('h=0.1','h=0.05','prcise')

xlabel('t');ylabel('y')

Problem 4

改用的龙格-库塔方法求解Problem 3。

4阶龙格-库塔方法

采用公式



求上的初值问题的近似解。

Answer

精确解。

当时，得到在处的近似解，误差。

当时，得到在处的近似解，误差。

可以看出，当步长减半时，误差大约减小1/16，与预期相符。

Code

function R=rk4(f,a,b,ya,M)

h=(b-a)/M;

T=zeros(1,M+1);

Y=zeros(1,M+1);

T=a:h:b;

Y(1)=ya;

for j=1:M

k1=h\*feval(f,T(j),Y(j));

k2=h\*feval(f,T(j)+h/2,Y(j)+k1/2);

k3=h\*feval(f,T(j)+h/2,Y(j)+k2/2);

k4=h\*feval(f,T(j)+h,Y(j)+k3);

Y(j+1)=Y(j)+(k1+2\*k2+2\*k3+k4)/6;

end

R=[T' Y'];

end

Chapter 11 特征值与特征向量

Problem

已知矩阵



是一个对称矩阵，且其特征值为 分别用幂法、对称幂法、反幂法求其最大特征值和特征向量。

1. 幂法

设矩阵A有一个主特征值，而且对应于有唯一的归一化的特征向量。通过下面的迭代过程可求出特征对。从下列向量开始：



用如下递归公式递归生成序列：

 ，

其中是绝对值最大的分量。序列和将分别收敛到和。

Answer

取，经过20次迭代，结果精度达到，结果为，。

Code

function [lamda,V,cnt,err]=power1(A,X,epsilon,max1)

lamda=0;

cnt=0;

err=1;

state=1;

while((cnt<=max1)&&(state==1))

Y=A\*X;

%Normalize Y

[m,~]=max(abs(Y));

c1=m;

dc=abs(lamda-c1);

Y=(1/c1)\*Y;

dv=norm(X-Y);

err=max(dc,dv);

X=Y;

lamda=c1;

state=0;

if(err>epsilon)

state=1;

end

cnt=cnt+1;

end

V=X;

end

clear,clc

A=[4 -1 1;-1 3 -2;1 -2 3];

X=[1 1 1]';

[lamda1,V,cnt,err]=power1(A,X,10^(-5),20)

1. 反幂法

设矩阵A有不同的特征值。为计算特征值，可选择常量，使得是的主特征值。选择适当的，可通过下列递归公式的得到序列 和：

，

其中

 ，

这两个序列将收敛到矩阵的主特征对，从而得到。

Answer

仍取，取，发现经过收敛于6.4000，而不是实际主特征值6。又取，经过6次迭代，结果精度达到，，。可见反幂法对于初始的选取敏感。

Code

function [lamda,V,cnt,err]=invpow(A,X,alpha,epsilon,max1)

[n n]=size(A);

A=A-alpha\*eye(n);

lamda=0;

cnt=0;

err=1;

state=1;

while((cnt<=max1)&&(state==1))

Y=A\X;

%Normalize Y

[m , ~]=max(abs(Y));

c1=m;

dc=abs(lamda-c1);

Y=(1/c1)\*Y;

dv=norm(X-Y);

err=max(dc,dv);

X=Y;

lamda=c1;

state=0;

if(err>epsilon)

state=1;

end

cnt=cnt+1;

end

lamda=alpha+1/c1;

V=X;

end

clear,clc

A=[4 -1 1;-1 3 -2;1 -2 3];

X=[1 1 1]';

[lamda,V,cnt,err]=invpow(A,X,5.8,10^-5,100);

1. 对称幂法

Answer

经过18次迭代，精度达到，结果为，。

Code

function [lamda,V,k,err]=sympower(A,X,epsilon,max1)

k=1;

X=X./sqrt(X'\*X);

while(k<=max1)

Y=A\*X;

lamda=X'\*Y;

if sqrt(Y'\*Y)==0

break;

end

err=sqrt((X-Y./sqrt(Y'\*Y))'\*(X-Y./sqrt(Y'\*Y)));

X=Y./sqrt(Y'\*Y);

if err<epsilon

lamda;

V=X;

break;

end

k=k+1;

end

end

clear,clc

A=[4 -1 1;-1 3 -2;1 -2 3];

X=[1 1 1]';

[lamda,V,k,err]=sympower(A,X,10^-9,40);