计算方法实验报告

912000720203
陈狄

1. Lagrange 插值

```
x=input('Enter x vector in the table: ');
y=input('Enter y vector in the table: ');
x_0=input('Enter the interpolation point: ');

n=length(x);

L=0;
for i=1:n
    M=y(i);

   for j=1:n
       if j == i
           continue;
       else
           M=M*(x_0-x(j))/(x(i)-x(j));
       end
   end

    L=L+M;
end

L
```

实验结果:

```
>> Lagrange
Enter x vector in the table: [0 1 2 3]
Enter y vector in the table: [1 1.6487 2.7183 4.4817]
Enter the interpolation point: 2.8

L =

   4.060416800000000
```

2. Newton 插值

```
x_0=input('Enter the interpolation point: ');

n=length(x);

dqtable=zeros(n,n-1);
```

```
for i=2:n
   dqtable(i,1)=(y(i)-y(i-1))/(x(i)-x(i-1));
end

for i=2:n
   for j=(i+1):n
       dqtable(j,i)=(dqtable(j,i-1)-dqtable(j-1,i-1))/(x(j)-x(j-i));
   end
end

N=y(1);
M=x_0-x(1);
for i=1:(n-1)
    N=N+dqtable((i+1),i)*M;
    M=M*(x_0-x(i+1));
end

N
```

实验结果:

```
>> Newton
Enter x vector in the table: [0 1 2 3]
Enter y vector in the table: [1 1.6487 2.7183 4.4817]
Enter the interpolation point: 2.8

N =

   4.060416800000000
```

3. 复化 Simpson 求积公式

```
function y=f(x)
if x ~= 0
   y=sin(x)/x;
else
   y=1;
end

a=input('Type the lower bound(a): ');
b=input('Type the upper bound(b): ');
n=input('Type the number of intervals(n): ');
```

```
h=(b-a)/n;
Sn=f(a)+f(b);

for i=1:n-1
    Sn=Sn+2*f(a + i*h);
end

for i=0:n-1
   Sn=Sn+4*f(a+ h/2 + i*h);
end

Sn=(h/6)*Sn
```

实验结果:

```
>> Simpson
Type the lower bound(a): 0
Type the upper bound(b): 1
Type the number of intervals(n): 4

Sn =

   0.946083310888472
```

4. Romberg 求积公式

```
function y=f(x)
if x ~= 0
   y=sin(x)/x;
else
   y=1;
end

a=input('Type the lower bound(a): ');
b=input('Type the upper bound(b): ');
e=input('Type the minimal error(epsilon): ');

k=0;
n=1;
h=b-a;
T(1,1)=h/2*(f(a)+f(b));
err=10;

while err>=e
```

```
    k=k+1;
    h=h/2;
    tmp=0;

    for i=1:n
        tmp=tmp+f(a+(2*i-1)*h);
    end

    T(k+1,1)=T(k,1)/2+h*tmp;

    if k<3
        for j=1:k
            T(k+1,j+1)=(4^j*T(k+1,j) - T(k,j))/(4^j-1);
        end
    else
        for j=1:3
            T(k+1,j+1)=(4^j*T(k+1,j) - T(k,j))/(4^j-1);
        end
    end

    n=n*2;
    if k<3
        err=abs(T(k+1,k+1)-T(k+1,k));
    else
        err=abs(T(k+1,4)-T(k+1,3));
    end
end
R=T(k+1,4)
```

实验结果：

```
>> Romberg
Type the lower bound(a): 0
Type the upper bound(b): 1
Type the minimal error(epsilon): 5e-8

R =

   0.946083070387223
```

5. 改进Euler公式

```
function [ f ] = f( x, y )
```

```
f=-(0.9*y)/(1+2*x);
end

field= input('Type the interval for x: ');
h=     input('Type the step length h: ');
y=zeros(floor((field(2)-field(1))/h+1));
y(1)=input('Type the initial condition y(1): ');

k_max=floor((field(2)-field(1))/h+1);

x=field(1);
for k=1:k_max
    y(k+1)=y(k)+h*f(x,y(k));
    y(k+1)=y(k)+(h/2)*(f(x,y(k)) + f(x+h, y(k+1)));
    fprintf('x = %f, y(%d) = %f \n', x, k, y(k));
    x=x+h;
end
```

实验结果：

```
>> Euler_Advanced
Type the interval for x: [0 0.1]
Type the step length h: 0.02
Type the initial condition y(1): 1
x = 0.000000, y(1) = 1.000000
x = 0.020000, y(2) = 0.982502
x = 0.040000, y(3) = 0.965954
x = 0.060000, y(4) = 0.950271
x = 0.080000, y(5) = 0.935381
x = 0.100000, y(6) = 0.921217
```

6. 四阶Runge-Kutta公式

```
function [ f ] = f( x, y )
f=-(0.9*y)/(1+2*x);
end

field= input('Type the interval for x: ');
h=     input('Type the step length h: ');
y=zeros(floor((field(2)-field(1))/h+1));
y(1)=input('Type the initial condition y(1): ');

k_max=floor((field(2)-field(1))/h+1);
```

```
x=field(1);
for k=1:k_max
    K1= f(x,y(k));
    K2= f(x+h/2, y(k) + h/2 *K1);
    K3= f(x+h/2, y(k) + h/2 *K2);
    K4= f(x+h, y(k) +h*K3);
    y(k+1)=y(k) + h/6 *(K1 + 2*K2 + 2*K3 +K4);

    fprintf('x = %f, y(%d) = %f \n', x, k, y(k));
    x=x+h;
end
```

实验结果:

```
>> Runge_Kutta
Type the interval for x: [0 0.1]
Type the step length h: 0.02
Type the initial condition y(1): 1
x = 0.000000, y(1) = 1.000000
x = 0.020000, y(2) = 0.982506
x = 0.040000, y(3) = 0.965960
x = 0.060000, y(4) = 0.950281
x = 0.080000, y(5) = 0.935393
x = 0.100000, y(6) = 0.921231
```

7. Newton迭代法

```
function [ f, diff_f ] = f( a )
syms y x
y=x^3+10*x-20;
f=double(subs(y,a));
diff_f=double(subs(diff(y), a));
end

I=input('Type the interval of root: ');
epsilon=input('Type the maximal error: ');

x_pre=I(1);
x_for=0;
e=100;

steps=0;
while(e > epsilon)
```

```
    [fun, diff_fun]=f(x_pre);
    x_for = x_pre -fun/diff_fun ;
    e=abs(x_pre-x_for);
    x_pre=x_for;
    steps=steps+1;
end

fprintf('The root is x=%f \n', x_for);
fprintf('Iteration steps is: %d \n', steps);
```

实验结果:

```
>> Newton
Type the interval of root: [1.5 2]
Type the maximal error: 1e-8
The root is x=1.594562
Iteration steps is: 4
```

8. Gauss列主元消去法

```
function [x]=ColumnElimination_f(A,b)
[n,~]=size(A);
x=zeros(n,1);

for k=1:n-1
    a_max=0;
    for i=k:n
        if abs(A(i,k))>a_max
            a_max=abs(A(i,k));
            r=i;
        end
    end

    if a_max<1e-10
        return;
    end

    if r>k
        for j=k:n
            z=A(k,j);
            A(k,j)=A(r,j);
            A(r,j)=z;
        end
        z=b(k);
```

```
        b(k)=b(r);
        b(r)=z;
    end

    for i=k+1:n
        m=A(i,k)/A(k,k);
        for j=k:n
            A(i,j)=A(i,j)-m*A(k,j);
        end
        b(i)=b(i)-m*b(k);
    end
end

if abs(A(n,n))==0
   return;
end

x(n)=b(n)/A(n,n);

for i=n-1:-1:1
    for j=i+1:n
        b(i)=b(i)-A(i,j)*x(j);
    end
    x(i)=b(i)/A(i,i);
end
```

实验结果：

```
>> A=[1 -1 1; 5 -4 3; 2 1 1];
>> b=[-4; -12; 11];
>> ColumnElimination_f(A,b)

ans =

   3.000000000000000
   6.000000000000000
  -1.000000000000000
```

9. Gauss-Seidel迭代法

```
function x=GaussSeidel_f(A,b)
D=diag(diag(A));
x=zeros(size(A,1),1);
```

```
X=ones(size(x));
L=zeros(size(A));
for i=2:size(A,1)
    for j=1:(i-1)
        L(i,j)=-A(i,j);
    end
end
U=D-L-A;

max_iter_num=100;
epsilon=1e-6;

for iter_num=0:max_iter_num
    X=(D-L)\(U*x)+(D-L)\b;

    if max(abs(X-x))<=epsilon
        break
    end
    x=X;
end
end
```

实验结果:

```
>> A=[10 -1 -2; -1 10 -2; -1 -1 5];
>> b=[7.2; 8.3; 4.2];
>> GaussSeidel_f(A,b)

ans =

   1.099999781713155
   1.199999866227841
   1.299999929588199
```