数值分析作业: 插值与多项式近似

Due on May, 2022

葛雨辰 201800150053

1. 序言

本章节研究插值与多项式逼近问题，**不考虑导数时有** Lagrange 插值多项式法，Nevile 迭代插值法与 Newton 插值法。**考虑导数时有** Hermite 插值多项式法。

首先声明以下所有程序都**按照方法名称命明**，变量含义自明。

2. 用 Lagrange，Nevile 与 Newton 插值计算 3.1 节 3c 题

2.1 运行程序与结果分析

发现 3.1 节 3c 题与 3.3 节 1d 题目数据相似，则擅自在 3.1 节 3c 题加入 3.3 节 1d 题目的导数值，由此也可以加入 Hermite 插值进行比较。(自然，Lagrange，Nevile 与 Newton 插值照常比较，无需使用导数值。)

分别用 Lagrange，Nevile 与 Newton 插值去逼近 $f(0.25)$ 处的值，运行程序的结果如下。其中第一个与第三个得到插值多项式与在 $f(0.25)$ 处的逼近值；第二个得到的是迭代矩阵，在右下角即为逼近值。第四个得到迭代矩阵（同样在右下角即为逼近值）以及插值多项式与在 $f(0.25)$ 处的逼近值。四个程序都计算了 CPU 运行时间。

```
>> x=[0.1,0.2,0.3,0.4];
y=[0.62049958,-0.28398668,0.00660095,0.24842440];
dy=[3.58502082,3.14033271,2.66668043,2.16599366];
>> lagrange(x,y,0.25)
The interpolation polynomial is 184.1*p^2 - 49.77*p - 207.3*p^3 + 3.964.
The value of approximation is -0.21034.
历时 0.071269 秒。
>> neville(x,y,0.25)
历时 0.000083 秒。

ans =

      0.6205           0           0           0
    -0.28399    -0.73623           0           0
    0.006601    -0.13869    -0.28808           0
     0.24842    -0.11431     -0.1326    -0.21034

>> newton(x,y,0.25)
The interpolation polynomial is 184.1*s^2 - 49.77*s - 207.3*s^3 + 3.964.
The value of approximation is -0.21034.
历时 0.060665 秒。
>> hermite(x,y,dy,0.25)

Q =

      0.6205           0           0           0           0           0           0           0
      0.6205       3.585           0           0           0           0           0           0
    -0.28399     -9.0449      -126.3           0           0           0           0           0
    -0.28399      3.1403      121.85      2481.5           0           0           0           0
    0.006601      2.9059     -2.3446     -620.98      -15512           0           0           0
    0.006601      2.6667      -2.392    -0.47395      3102.5       93075           0           0
     0.24842      2.4182     -2.4845     -0.4625    0.057215      -10342 -3.4472e+05           0
     0.24842       2.166     -2.5224    -0.37949     0.41506      1.7892       34478   1.264e+06

The interpolation polynomial is 1166.403111*s - 14535.48811*s^2 + 90282.26593*s^3 - 299219.063*s^4 + 506742.4272*s^5 - 344723.5914*s^6 + s^7 - 35.7479909.
The value of approximation is -0.0746.
历时 0.090199 秒。
```

其中注意到前三个三个方程计算**所得结果相同**，其原因为所用的插值多项式都为比数据的维度低一维的多项式（唯一性由数学定理保证)。最终逼近值为

$$f(0.25) = -0.21034$$

以及**唯一**的多项式（结果保留四位有效数字）

$$f(x) = -207.3 \cdot x^3 + 184.1 \cdot x^2 - 49.77 \cdot x + 3.964.$$

考察四个程序的 CPU 时间发现：Neville 的计算时间最短，hermite 由于加入导数的计算时间变长。其余两个处于中等水平且 Newton 由于迭代风格出众略居其上。

2.2 程序说明

在 Lagrange 差值多项式与 Nevile 中都是循规蹈矩按照书上的程序图写下代码。但在第三个代码中，我们改进了书中的公式。计算所有形如:

$$f[x_0,\dots]$$

的 newton 差分，发现也可以得到每一个 ewton 任意阶差分。例如具体在在代码中，有如下的迭代

$$\mathrm{coeff}(j)=\frac{y(j)-y(i)}{x(j)-x(i)};$$

其中任意次迭代的 coeff（变量）代表 newton 差分表这一列的每一个系数。因此有更高效的迭代公式路径，最终代码如下。

```
% Newton's interpolatory divided difference fomula
% t is coordinate number to be evaluated
function output=newton(x,y,t)
% Calculate runtime of the program
tic;
% initialize
newpoly=y(1);
syms s;
% compute the dimension of x
n=length(x);
coeff=zeros(1,n);
temp1=zeros(1,n);
dxs=1;
for i=1:n-1
    for j=i+1:n
        coeff(j)=(y(j)-y(i))/(x(j)-x(i));
    end
    temp1(i)=coeff(i+1);    % tempoary variable
    dxs=dxs*(s-x(i));
    newpoly=newpoly+temp1(i)*dxs;
    y=coeff;
end
% simplify the polynomial
newpoly=simplify(newpoly);
newpoly=vpa(newpoly,4);
% output the evaluted number
m=length(t);
% temporary value
temp=zeros(1,m);
for i=1:m
    temp(i)=subs(newpoly,'s',t(i));
end
disp(['The interpolation polynomial is ',char(newpoly),'.']);
disp(['The value of approximation is ',num2str(temp),'.']);
toc;
```

结果显示最终仍得到正确的逼近值 $f(0.25)=-0.21034$ 与逼近多项式

$$f(x)=-207.3\cdot x^3+184.1\cdot x^2-49.77\cdot x+3.964.$$

这说明上述改进是正确的。

3. 用 Hermite 插值计算 3.3 节 1d 题

3.1 程序运行

本节使用 3.3 节 1d 题的数据。注意：该数据在 $x = 0.1$ 与 $x = 0.2$ 处的函数值与 3.1 节 3c 题的函数值是互为相反数的。因此不可以将该数据运行的 Hermite 插值算法与上题数据运行的剩余插值算法进行比较！（将该数据在 $x = 0.1$ 与 $x = 0.2$ 处的函数值取反就可以比较，比较结果见第二节。）用 Hermite 插值多项式去逼近任意 $f(x)$ 处的值，运行程序的结果如下。

```
>> x=[0.1,0.2,0.3,0.4];
>> y=[-0.62049958,0.28398668,0.00660095,0.24842440];
>> dy=[3.58502082,3.14033271,2.66668043,2.16599366];
>> hermite(x,y,dy,0.25)

Q =

  列 1 至 4

     -0.6205           0           0           0
     -0.6205       3.585           0           0
     0.28399      9.0449      54.598           0
     0.28399      3.1403     -59.045     -1136.4
    0.006601     -2.7739     -59.142    -0.48301
    0.006601      2.6667      54.405      1135.5
     0.24842      2.4182     -2.4845     -284.45
     0.24842       2.166     -2.5224    -0.37949

  列 5 至 8

           0           0           0           0
           0           0           0           0
           0           0           0           0
           0           0           0           0
      5679.8           0           0           0
      5679.8    0.040475           0           0
     -7099.6      -42598  -1.4199e+05           0
      1420.3       42600   2.8399e+05    1.42e+06

The interpolation polynomial is 55.11816729*s - 1492.939126*s^2 + 15902.83112*s^3 - 76676.78029*s^4 + 170393.1115*s^5 - 141994.9425*s^6 + s^7 - 1.000014416.
The value of approximation is 0.16675.
历时 0.109415 秒。
```

程序输出的是所有的 Hermite 多项式系数。最终得到逼近值为 $f(0.25) = 0.16675$。以及插值多项式，写成标准形式为：(保留 10 位有效数字)

$$\begin{aligned} f(s) =& 55.11816729 \cdot s - 1492.939126 \cdot s^2 + 15902.83112 \cdot s^3 - 76676.78029 \cdot s^4 \\ &+ 170393.1115 \cdot s^5 - 141994.9425 \cdot s^6 + s^7 - 1.000014416. \end{aligned}$$

3.2 程序分析

书中流程图的 x 下标由 0 开始，而 **matlab 的检索一般由 1 开始**。因此针对这个问题需要改进迭代公式，经过计算获得如下的迭代公式：

$$Q(i,j) = \frac{Q(i,j-1) - Q(i-1,j-1)}{z(i) - z(i-j+1)}$$

结果显示上述改进是正确的。

3.3 试运行

虽说本段居于最后一节，但在逻辑上是首先的。由于该算法的迭代公式较为复杂，笔者运用所写算法去运行书中数据进行检验：考察**书中 Table3.12** 的数据，运行如下：

```
>> x=[1.3,1.6,1.9];
>> y=[0.6200860,0.4554022,0.2818186];
>> dy=[-0.5220232,-0.5698959,-0.5811571];
>> hermite(x,y,dy,1.5)

Q =

     0.62009           0           0           0           0           0
     0.62009    -0.52202           0           0           0           0
      0.4554    -0.54895   -0.089743           0           0           0
      0.4554     -0.5699   -0.069833    0.066366           0           0
     0.28182    -0.57861   -0.029054    0.067966   0.0026667           0
     0.28182    -0.58116  -0.0084837    0.068567   0.0010019  -0.0027747

The interpolation polynomial is 27.31537557*s - 36.281958*s^2 + 23.64089889*s^3 - 7.697333333*s^4 + s^5 - 7.241024342.
The value of approximation is 0.51167.
历时 0.075236 秒。
```

与 3.14 进行比较发现**表格数据与最终拟合的数据完全一致**, 且逼近值

$$f(0.25) = 0.512 = H_5(1.5)$$

最终验证了所写算法的正确性!

4. 代码附录

4.1 Lagrange 插值

```
% lagrange polynomial for interpolation
% t is coordinate number (vector) to be evaluated
function output=lagrange(x,y,t)
% Calculate runtime of the program
tic;
% calculate the dimension of x
n=length(x);
syms p;
% calulate the lagrange polynomial
lapoly=0;
for i=1:n
    labase=y(i);
    for j=1:i-1
        labase=labase*(p-x(j))/(x(i)-x(j));
    end
    for j=i+1:n
        labase=labase*(p-x(j))/(x(i)-x(j));
    end
    lapoly=lapoly+labase;
end
lapoly=simplify(lapoly);
lapoly=vpa(lapoly,4);
% output the evaluted number
m=length(t);
% temporary value
temp=zeros(1,m);
for i=1:m
    temp(i)=subs(lapoly,'p',t(i));
end
disp(['The interpolation polynomial is ',char(lapoly),'.']);
disp(['The value of approximation is ',num2str(temp),'.']);
toc;
```

4.2 Neville 迭代插值

```
% neville's iterated interpolation
% t is coordinate number to be evaluated
function output=neville(x,y,t)
n=length(x);
% Calculate runtime of the program
tic;
% initialize the table
Q=zeros(n,n);
for i=1:n
    Q(i,1)=y(i);
end
for i=2:n
    for k=2:i
        Q(i,k)=((t-x(i-k+1))*Q(i,k-1)-(t-x(i))*Q(i-1,k-1))/(x(i)-x(i-k+1));
    end
end
toc
output=Q;
```

4.3 Newton 插值

```
% Newton's interpolatory divided difference fomula
% t is coordinate number to be evaluated
function output=newton(x,y,t)
% Calculate runtime of the program
tic;
% initialize
newpoly=y(1);
syms s;
% compute the dimension of x
n=length(x);
coeff=zeros(1,n);
temp1=zeros(1,n);
dxs=1;
for i=1:n-1
    for j=i+1:n
        coeff(j)=(y(j)-y(i))/(x(j)-x(i));
    end
    temp1(i)=coeff(i+1);    % tempoary variable
    dxs=dxs*(s-x(i));
    newpoly=newpoly+temp1(i)*dxs;
    y=coeff;
end
% simplify the polynomial
newpoly=simplify(newpoly);
newpoly=vpa(newpoly,4);
% output the evaluted number
m=length(t);
% temporary value
temp=zeros(1,m);
for i=1:m
    temp(i)=subs(newpoly,'s',t(i));
end
disp(['The interpolation polynomial is ',char(newpoly),'.']);
disp(['The value of approximation is ',num2str(temp),'.']);
toc;
```

4.4 Hermite 插值

```
% Hermite interpolation
% t is coordinate number to be evaluated
function output=hermite (x,y,dy,t)
% Calculate runtime of the program
tic;
% compute the dimension of x
n=length(x);
syms s;
z=zeros(2*n,1);
Q=zeros(2*n,2*n);
for i=1:n
    z(2*i-1)=x(i);
    z(2*i)=x(i);
    Q(2*i-1,1)=y(i);
    Q(2*i,1)=y(i);
    Q(2*i,2)=dy(i);
    if(i≠1)
        Q(2*i-1,2)=(Q(2*i-1,1)-Q(2*i-2,1))/(z(2*i-1)-z(2*i-2));
    end
end
for i=3:(2*n)
    for j=3:i
        Q(i,j)=(Q(i,j-1)-Q(i-1,j-1))/(z(i)-z(i-j+1));
    end
end

% compute the hermite poynomial
temp=1;
hermitepoly=Q(1,1);
for i=1:n-1
    temp=temp*(s-x(i));
    hermitepoly=hermitepoly+Q(2*i,2*i)*temp;
    temp=temp*(s-x(i));
    hermitepoly=hermitepoly+Q(2*i+1,2*i+1)*temp;
end
hermitepoly=hermitepoly+temp*(s-x(n));
hermitepoly=simplify(hermitepoly);
hermitepoly=vpa(hermitepoly,10);
% output the evaluted number
m=length(t);
% temporary value
temp=zeros(1,m);
for i=1:m
    temp(i)=subs(hermitepoly,'s',t(i));
end
Q
disp(['The interpolation polynomial is ',char(hermitepoly),'.']);
disp(['The value of approximation is ',num2str(temp),'.']);
toc;
```