牛顿插值公式及其算法程序

罗子林

摘要： 本文介绍了牛顿插值公式的推导过程，并且给出了它的算法，以及在计算机上数据处理、数据存贮和实现计算方法。用此算法可较直观简明地转成多种高级语言程序，并给出了相应的PASCAL语言程序，还通过对此算法的改进，得到了在插值点不同次数的牛顿插值多项式，有利于实际计算中对计算结果的研究。

关键词： 牛顿插值公式；程序

用插值法研究函数的近似表达式是古典数值分析中的重要内容之一。在实际问题中，我们只知道某个函数$f(x)$对映彼此不同点x_0，x_1，…，x_n的近似值，甚至直至某阶导数。要应用它们来求出函数$f(x)$的近似表达式，这种近似处理在计算方法里叫做逼近。关于逼近方式常用到的是：插值，一致逼近及均方逼近（或平方逼近）。我们把寻找一个较为简单的函数$y(x)$作为$f(x)$的插值函数，使得在x_i（$i=0$，1，…，n）上，$y(x)$与$f(x)$有相同的函数值，甚至直至某阶导数。这就是插值。在计算方法里，所谓简单的函数主要是指用四则运算进行计算的函数，其一般形式是有理分式函数。而其中较为简单的则是多项式。一旦$y(x)$取为多项式，则上述插值问题就变成为多项式插值问题。

我们先给出多项式插值的确切涵义。设P_n表示次数不超过n的所有实系数多项式的集合。对给出的函数$f(x)$的$n+1$个数据有$\{(x_i, f_i)\}_{i=0}^{n}$，需要寻找$y(x)\in P_n$，使之满足

$$y(x_i)=f(x_i),\quad i=0,\ 1,\ \cdots,\ n \qquad (1)$$

这就是所谓的多项式插值问题。通常这$n+1$个相异点x_0，x_1，…，x_n叫做插值基点。它们在实轴上占据的闭区间$[\min\{x_i\},\ \max\{x_i\}]$称为插值区间，而满足（1）式的多项式$y(x)\in P_n$叫做插值多项式，它是唯一存在的。从几何角度上来说，上述问题就相当为：对平面上$n+1$个不同点(x_i, f_i)（$i=0$，1，…，n）要寻找一条次数不超过n的经过这些已知点的多项式曲线。牛顿插值公式就属于多项式插值问题，它在多项式插值问题中占有重要地位。

为了推导牛顿插值公式，我们先介绍差商的概念。

设已知函数$f(x)$的$n+1$个数据如下表所示：

x	x_0	x_1	x_2	x_3	…
$f(x)$	f_0	f_1	f_2	f_3	…

收稿日期：1994—03—25　作者单位：景德镇高专数学系

（这里，当$i\neq j$时，$x_i\neq f_i$）。其中f_i可能是连续变量x的函数$f(x)$在$x=x_i$处的值，也可能是离散变量的函数的值。

我们先令$f[x_k]=f(x_k)$，将

$$f[x_0,\ x_1,\ \cdots,\ x_k]=\frac{f[x_1,\ x_2,\ \cdots,\ x_k]-f[x_0,\ x_1,\ \cdots x_{k-1}]}{x_k-x_0}$$

称为函数f在x_0，x_1，$\cdots$，x_k处的k阶差商（$k\geqslant 1$）。

于是按上述定义，f在x_0，x_1处的一阶差商为：

$$f[x_0,\ x_1]=\frac{f[x_1]-f[x_0]}{x_1-x_0}$$

显然，$f[x_0,\ x_1]=f[x_1,\ x_0]$

f在x_0，x_1，x_2处的二阶差商为：

$$f[x_0,\ x_1,\ x_2]=\frac{f[x_1,\ x_2]-f[x_0,\ x_1]}{x_2-x_0}$$

同样可以验证：

$$f[x_0,\ x_1,\ x_2]=f[x_1,\ x_0,\ x_2]=\cdots=f[x_2,\ x_1,\ x_0]$$

实际上任意阶差商均具有这种对称的性质。

假如$f(x)$本身是n次多项式，其在x_0，x_1，$\cdots x_{k-1}$处的k阶差商$f[x_0,\ x_1,\ \cdots x_{k-1}]$，当$k\leqslant n$时，它是关于$x$的$n-k$次多项式，而当$k>n$时，此差商恒等于零。这种性质同多项式导数的有关性质极为类似。

由差商的定义可以看出，要计算f在x_0，x_1，$\cdots$，x_k处的k阶差商$f[x_0,x_1,\ \cdots,\ x_k]$，需要两个$k-1$阶的差商，即$f[x_1,x_2,\cdots,x_k]$与$f[x_0,x_1,\cdots,x_{k-1}]$的值，而要计算这两个$k-1$阶差商又需要三个$k-2$阶的差商，即$f[x_2,x_3,\cdots,x_k]$、$f[x_1,x_2,\cdots,x_{k-1}]$及$f[x_0,x_1,\ \cdots,\ x_{k-2}]$的值，$\cdots$，最后需$f(x_0)$，$f(x_1)$，$\cdots$，$f(x_k)$的值。我们把这个过程可以列成如下的差商表：

x	$f(x)$	一阶差商	二阶差商	$\cdots$	k阶差商
x_0	$f(x_0)$	$\searrow$			
		$f[x_0,x_1]$	$\searrow$		
x_1	$f(x_1)$		$f[x_0,x_1,x_2]$	$\searrow$	
		$f[x_1,x_2]$		$\cdots$	$\searrow$
x_2	$f(x_2)$		$f[x_1,x_2,x_3]$		
		$f[x_2,x_3]$	$\vdots$	$\cdots$	$f[x_0,x_1,\cdots,x_k]$
x_3	$f(x_3)$	$\vdots$			
$\vdots$	$\vdots$		$f[x_{k-2},x_{k-1},x_k]$		
		$f[x_{k-1},x_k]$			
x_k	$f(x_k)$				

运用上述表格的头两列可以算出k个一阶差商，接着利用第一列与第三列可以算出$k-1$个二阶差商，按此步骤由左向右就可以求出k阶差商$f[x_0,\ x_1,\ \cdots,\ x_k]$的值。

有了差商的概念，我们就可以方便地推导牛顿插值公式了。设有$n+1$个不同的插值基点$x_0,x_1,\ \cdots x_n$并已知对应的$f(x)$的函数值$f(x_0)$，$f(x_1)$，$\cdots$，$f(x_k)$，则根据差商定义有：

$$f[x]=f[x_0]+(x-x_0)f[x_0,\ x_1]$$

$$f[x,\ x_0]=f[x_0,\ x_1]+(x-x_1)f[x,\ x_0,\ x_1]$$

$$f[x, x_0, x_1]=f[x_0, x_1, x_2]+(x-x_2)f[x, x_0, x_1, x_2]$$
$$\cdots\quad\cdots\quad\cdots\quad\cdots\quad\cdots\quad\cdots\quad\cdots$$
$$f[x, x_0, x_1, \cdots, x_{n-1}]=f[x_0, x_1, \cdots, x_n]+(x-x_n)\cdot f[x, x_0, x_1, \cdots, x_n]$$

将上面第二式代入第一式得

$$f[x]=f[x_0]+(x-x_0)f[x_0, x_1]+(x-x_0)(x-x_1)f(x, x_0, x_1]$$

再将第三式代入上式，一直做到底，最后得到

$$f(x)=f[x_0]+(x-x_0)f[x_0, x_1]+(x-x_0)(x-x_1)\cdot f[x_0, x_1, x_2]+\cdots+(x-x_0)(x-x_1)\cdots(x-x_{n-1})f[x_0, x_1, \cdots, x_n]+(x-x_0)(x-x_1)\cdots(x-x_n)f[x, x_0, x_1, \cdots, x_n]\quad\cdots\cdots\cdots(2)$$

上式右端前$n+1$项组成关于x的多项式，令为$y(x)$，则$y(x)\in P_n$

$$y(x)=f[x_0]+(x-x_0)f[x_0, x_1]+(x-x_0)(x-x_1)f[x_0, x_1, x_2)+\cdots+(x-x_0)(x-x_1)\cdots(x-x_{n-1})f[x_0, x_1, \cdots, x_n[\quad(3)$$

而（2）式中最后一项即是插值多项式与原函数的误差，令为$E(x)$，则

$$E(x)=(x-x_0)(x-x_1)\cdots(x-x_n)f[x_1, x_0, \cdots, x_n]$$

实际上，我们把牛顿插值公式与差商表一比较就可看出，按差商表中箭头所示就可以写出牛顿插值公式。这些差商实际上就是牛顿插值公式的系数。差商表的构造过程实际上就是我们给出牛顿插值公式算法的基础。

要给出牛顿插值公式首先要讨论牛顿公式系数$f[x_0, x_1, \cdots, x_k](k=1, 2, \cdots n)$，即各阶差商的算法。由差商表的构造可以看出，要求得这些系数需要用初始$n+1$个数据对$\{x_j, f(x_j)\}_{j=0}^{n}$自左向右计算出相应的各阶差商。例如，为了计算出$f[x_0, x_1]$，$f[x_0, x_1, x_2]$，$f[x_0, x_1, x_2, x_3]$，我们还必需多计算差商$f[x_1, x_2]$，$f[x_2, x_3]$，及$f[x_1, x_2, x_3]$的值。又由于在算出$f[x_0, x_1]$之后，$f(x_1)$就不起作用了，在算出所有一阶差商之后，$f(x_1)$，…，$f(x_n)$便可退出计算过程，照此想法，我们可以把牛顿公式的系数，即相应的各阶差商最后存放在初始数据对的函数值的单元之中，即$f[x_0, x_1]\to f(x_1)$，$f[x_0, x_1, x_2]\to f(x_2)$，…，$f[x_0, x_1, \cdots, x_n]\to f(x_n)$。这样，节省了存贮空间，又简化了算法。我们先给出计算牛顿公式系数$f[x_0, x_1, \cdots, x_n](k=1, 2, \cdots, n)$的算法

1．输入$\{x_0, x_1, \cdots, x_n, f_0, f_1, \cdots f_n\}$

2．对$j=0, 1, \cdots, n$

2.1　$y_j\leftarrow f_j$　　（算法Ⅰ）

3．对$k=0, 1, \cdots, n-1$

3.1　对$j=n, n-1, \cdots, k+1$

3.1.1　$y_j=(y_j-y_j-1)/(x_j-x_j-k-1)$

4．输出$\{y_0, y_1, \cdots, y_n\}$

有了上述算法，我们可以容易地用牛顿插值公式（3），按下列算法求出在任意给出的插值点$\overline{x}$处的$y(\overline{x})$值作为$f(\overline{x})$的近似值，下面是牛顿插值公式的求值算法

1．输入$\{x_0, \cdots x_n, y_0, \cdots, y_n$及$\overline{x}\}$

2．$P\leftarrow y_n$

3．对于$i=n-1, n-2, \cdots, 0$

3.1 $P \leftarrow P(\bar{x}-x_i)+y_i$ （算法Ⅱ）

4．输出$\{\bar{x}, P\}$

几点说明：1.算法Ⅰ中计算各阶差商是按差商表，由左向右，由下往上计算的，这是为了有效利用$f(x_i)(i=1, 2, \cdots, n)$的单元；2．算法Ⅱ的输入部分，除$\bar{x}$是插值点外，其余数据均为算法Ⅰ的自然输出；3．算法Ⅱ中的步骤3是按牛顿公式作如下改写的

$y(x)=(x-x_0)((x_1-x_2)(\cdots((x-x_{n-1})f[x_0, x_1, \cdots, x_n])+f[x_0, x_1, \cdots, x_{n-1}])+\cdots)+f[x_0, x_1, x_2])+f[x_0, x_1])+f[x_0]$，这种改写将能充分利用高级语言提供的循环语句，使计算量大为节省。这是需要注意的地方。

以上算法同各种高级语言的形式非常接近，我们可以容易地把它转为各种算法语言。现结合下面的实例

例：已知函数$f(x)$如下数据对：$x_0=-1.0$，$f_0=1.5$；$x_1=1.0$，$f_1=2.0$；$x_2=2.0$，$f_2=2.0$，$x_3=2.5$，$f_3=1.5$．试用牛顿插值公式求$f(0.3)$的近似值。

给出算法Ⅰ、Ⅱ对应的PASCAL语言程序如下：

```
PROGRAM  NewTon(Input, output);
CONST      N=3;
VAR
x, f, y: ARAAy[O..N]OF real;
i, j, k: integer; P, S: real;
BEGIN(*main*)
write ln('please enter the numbers:')
FOR i:=O TO N DO
Read(x[i], f[i]); (*读入N+1个数据对*)
Read ln;
FOR i:=O TO N DO
y[i]:=f[i];
FORK:=O TO N--1 DO
BEGIN
FOR j:=N DOWNTO k+1 DO
y[j]:=(y[j]-y[j-1])/(x[j]-x[j-k-1]);
END;
Write ln('please)
readln(S); (*读入插值点S，即x*)
P:=y[N];
FOR i:=N-1 DOWNTO O DO
P:=P*(S-x[i])+y[i];
Writeln('f(',S,')=',P:8:4)(*输出f(S)近似值*)
END. (*of main*)
```

根据所给实例，程序运行记录如下：

```
please enter the numbers:
-1.0    1.5    ↵
 1.0    2.0    ↵
 2.0    2.0    ↵
 2.5    1.5    ↵
 plcase enter
 f(0.3)=1.643
```

说明：对不同的数据时，只要修改常数说明部分N的值即可；对给出的N＋1个基点如是连续函数（如$f(x)=e^{-x}$）的情形，需使用PASCAL语言的自定义函数，程序可作如下修改：

1．变量说明部分删去f;

2．程序执行部分开始符BEGIN之前增加如下函数说明部分：

```
FUNCTION  F(i：integer)：ARRAY[O..N]OF real;
BEGIN
F[i]：=EXP(-x[i]);
END
```

3．在执行部分读入数据中删去$f[i]$。

根据算法Ⅰ、Ⅱ，我们也能很容易地写出BASIC语言程序，在此不再多叙。

需要指出以上算法只能给出在$\overline{x}$处的插值多项式$y(x)\in P_n$的值作为$f(\overline{x})$的近似值。实际上我们只要对算法Ⅱ进行修改就能算出在$\overline{x}$处$P_i(i=n, n-1, \cdots, 1)$中的插值多项式的值。这有利于我们对在$\overline{x}$处n个插值多项式的值进行比较，以选择最接近于$f(\overline{x})$的值作为其近似值。修改后的算法如下：

1．输入$\{x_0, x_1, \cdots, x_n, y_0, y_1, \cdots, y_n$及$\overline{x}\}$

2．对$i=1, 2, \cdots, n$

　2.1　$P_i\leftarrow y_i$

　2.2　对$j=i-1, i-2, \cdots, 0$　　算法Ⅲ

　2.2.1　$P_i\leftarrow P_i(\overline{x}-x_j)+y_j$

3．对$i=1, 2, \cdots, n$

3.1　输出$\{\overline{x}, P_1\}$

对以上算法的说明类似于算法Ⅱ。

我们之所以可以较容易地把算法Ⅱ改为算法Ⅲ，是由于牛顿插值公式（3）右端的前$k+1$项实际上就是满足条件$y(x_j)=f(x_j)(j=0, 1, \cdots, k)$的$P_k$中的插值多项式，这是牛顿公式的优点所在。另一大优点是牛顿公式的计算量比其它多项式插值方法的计算量小得多，整个算法的费用较低。

虽然牛顿插值公式有显著优点，但我们对该选取何种方法还须根据初始数据的特点来加以选择。只有这样才可能找到最好的近似方法以求得最满意的结果。