数值分析第二章补充阅读

朱宇嘉

1 插值法

插值法主要解决的是以下问题：

设函数 $y = f(x)$ 定义在区间 $[a, b]$ 上，$x_0, x_1, ..., x_n$ 是 $[a, b]$ 上取定的 $n + 1$ 个互异节点，且在这些节点上的取值分别是 $y_0 = f(x_0), y_1 = f(x_1), ..., y_n = f(x_n)$。那么我们可以找到一个且只能找到一个 **至多 n 次的多项式函数** $P(x), s.t. P(x_i) = f(x_i) = y_i, i = 0, 1, ..., n$。将 $P(x)$ 称为 $f(x)$ 的一个插值函数。

从中我们需要做以下说明：

1. 插值法相当于是用一个简单的多项式函数 $P(x)$ 在给定的点上去完美逼近一个复杂函数 $f(x)$，以试图在整个定义域上可以用 $P(x)$ 近似 $f(x)$。而这个复杂函数 $f(x)$ 可能是未知的，也可能是过于复杂难以直接使用的，只能取出几个十分重要的函数点并用多项式函数插值以简化运算。
2. 上面给出的定义对应的是 Largrange 插值和 Newton 插值的形式：给定函数取值的 $n + 1$ 个限制，可以得到一个至多 n 次的插值结果。
3. 一般情况下，给定 $n + 1$ 个插值限制，就可以构造出一个至多 n 次的插值函数。但是，这些插值条件并不一定只是函数取值相等，也可能是导数值相等，二阶导数值相等。之后的埃尔米特插值就是讨论存在一阶导数相等的限制下如何构造插值函数。
4. 要从数学的角度看插值的意义。如果从统计或者机器学习的角度看插值，你可能会认为插值去 100% 拟合给定的点的取值，这是一种过拟合现象（其实龙格现象也是过拟合现象的一种表现）。但其实插值法只是去找到一个简单函数去让其拥有复杂函数的部分性质，而不是找到变量之间含有的函数关系。

在这一章里，我们主要来用编程语言实现 Largrange 插值和 Newton 插值。因为 R 语言和 Python 语言对于统计的实际应用帮助较大，所以之后所有的程序实现，我都将主要使用这两种语言。如果你喜欢其他语言，也可以自己编写相应程序。为了保证公平性，我将用 R 语言实现 Largrange 插值，用 Python 语言实现 Newton 插值。最后，我们再简要讲一下龙格现象的产生与避免。

2 Lagrange 插值的 R 语言实现

Lagrange 插值的形式是：

$$L(x) = \sum_{k=0}^{n} l_k(x) y_k,$$

其中:

$$l_k(x) = \prod_{\substack{j=0 \\ j \neq k}}^{n} \frac{x - x_j}{x_k - x_j}$$

我们可以用以下 R 语言代码来实现 Lagrange 插值法。其中，为了加速程序运行速度，我避免了使用 for 循环，而是使用了 outer 函数，apply 函数和矩阵乘法这三个工具。大家也可自行写一个使用 for 循环的函数，并比较一下两者的运行速度。

```
Lagrange<-function(x,y){
  ## input:x & y are vectors, all length n+1
  ## Step 1: 检验输入数据的合法性
  if(length(x)!=length(y)) stop("x & y have different lengths")
  if(length(x)<=1) stop("number of points less than 2")
  if(length(x)!=length(unique(x))) stop("have repeated points")
  ## Step 2: 构造 x_i-x_j 的表格并计算分母
  x_diff<-outer(x,x,'-')
  diag(x_diff)<-1
  denominator<-apply(x_diff,1,prod)
  ## Step 3: 构造 Lagrange function
  f<-function(newpoint){
    ## input a newpoint and return the y value
    new_diff<-as.matrix((newpoint-x),nrow=1)%*%matrix(1,nrow=1,ncol=length(x))
    diag(new_diff)<-1
    numerator<-apply(new_diff,2,prod)
    return(sum(numerator/denominator*y))
  }
  ## Step 4:return the function
  return(f)
}
```

函数编写主要是由以下的步骤实现的：

Step 1：检验输入数据的合法性。3 个检验分别是检验 x 和 y 的长度是否相同；输入的点的个数是否比 2 大；x 中是否含有重复的元素。

Step 2：计算 $l_k(x)$ 的分母。请大家自行理解这一快速计算的技巧。

Step 3：构造一个函数，其表示 Lagrange 插值多项式，输入一个新点，输出 Lagrange 插值结果。在函数内部进行的是 $l_k(x)$ 分子的计算以及最终函数值的计算。

Step 4：返回 Lagrange 插值函数。

我们用课件上的例题 2.3 来测试一下结果的准确性：

```
x<-c(0,1,2,4)
y<-c(1,9,23,3)
# 得到插值函数
f<-Lagrange(x,y)
# 通过取点画出插值函数的图像
x_fit<-y_fit<-seq(0,4,length.out=100)
for(i in 1:length(x_fit)) y_fit[i]<-f(x_fit[i])
# plot 函数画点，lines 函数画插值函数
plot(x,y,ylim=c(0,30))
lines(x_fit,y_fit,type='l')
```

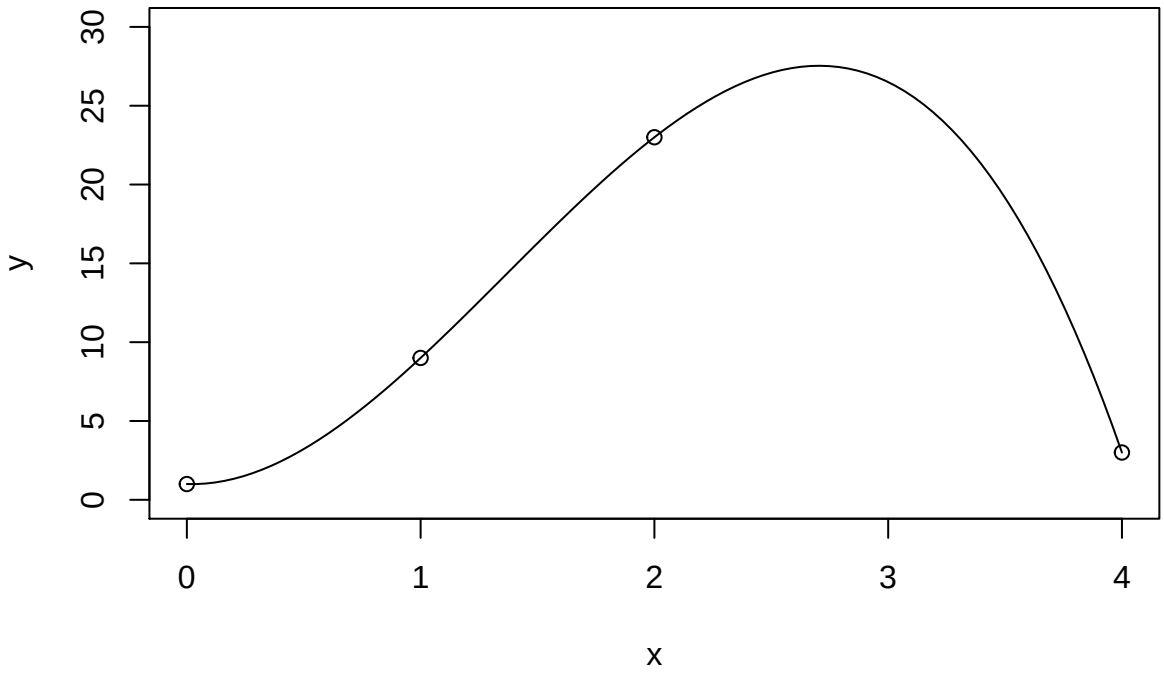

图 1: Lagrange 插值结果

我们画出了 4 个给定的点，并将得出的插值曲线绘制在图上。结果显示没有什么问题。

3 Newton 插值的 Python 实现

Lagrange 的优点是结构对称，形式优美。但其缺点是不具有承袭性，在大规模问题中对于添加节点不够友好。Newton 插值克服了这一缺点，是一种可承袭的插值，其形式如下：

$$N_n(x) = f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + ...+$$

$$f[x_0, x_1, ..., x_n](x - x_0)(x - x_1)...(x - x_{n-1})$$

Python 语言是一种面向对象的语言，面向对象的编程方式有着易于维护的优点。而 Newton 插值中时常需要添加点进行维护。因此，我们这里使用类的方法进行实现。

```
import numpy as np

class Newton:
    def __init__(self):
        self.x = []
        self.diff = [[]]
        self.number = 0

    ## 用于计算差商
    def ComputeDiffOper(self):
        if self.number >= 2:
            self.diff.append([])
            for i in range(self.number - 1):
                temp_num = self.diff[i][-1] - self.diff[i][-2]
                temp_dom = self.x[-1] - self.x[-(i+2)]
                self.diff[i+1].append(temp_num/temp_dom)

    ## 向类内添加新点，注意添加时需要更新差商
    def add(self,x,y):
        assert x not in self.x, 'have repeated x'
        self.x.append(x)
        self.diff[0].append(y)
        self.number += 1
        self.ComputeDiffOper()

    ## 计算新点的近似值
```

```
    def approximate(self, newpoint):
        assert self.number >= 2, 'number of points less than 2'
        diffoper = np.array([diff[0] for diff in self.diff])
        x = np.append(1, newpoint - np.array(self.x[:-1]))
        x = np.cumprod(x)
        return np.sum(x * diffoper)
```

这个类中 self.x 用来存储 x，self.y 用来存储所有差商，其是由 list 构成的 list，第一个 list 存储 y，第二个存储一阶差商，第三个存储二阶差商，以此类推。我们可以调用 3 个操作。第一个是声明一个类以供计算；第二个是向类中添加一对 (x, y)，在添加之后使用类内的 ComputeDiffOper() 函数进行相应的差商计算；第三个是计算新点近似值的 approximate() 函数。在类内只使用了一个明显的 for 循环。但事实上，这个 for 循环是可以避免的，但为了防止语句过长，我把这个 for 循环留了下来。熟悉 Python 的同学可以自己想一下如何避免这个 for 循环。

下面我们来验证一下结果的准确性：

```
import matplotlib.pyplot as plt
plt.subplots_adjust(wspace = 0.3, hspace = 0.8)

x = np.linspace(1,5,100).tolist()
def plot():
    y = [newton.approximate(x[i]) for i in range(100)]
    plt.plot(x,y)
    plt.plot(newton.x,newton.diff[0],'ro')
    plt.xlabel('x')
    plt.ylabel('y')

newton = Newton()
newton.add(4,8)
newton.add(2,4)
plt.subplot(2,2,1)
plot()
plt.title('2 points')
newton.add(5,6)
plt.subplot(2,2,2)
plot()
plt.title('3 points')
newton.add(1,4)
plt.subplot(2,2,3)
plot()
```

```
plt.title('4 points')
newton.add(3,7)
plt.subplot(2,2,4)
plot()
plt.title('5 points')
```

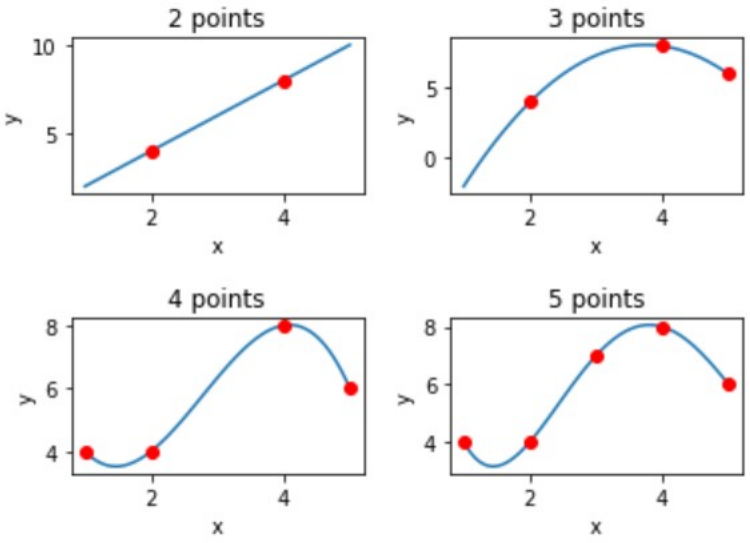

图 2: Newton 插值结果

我们同样使用课件上的练习题来验证，我们改变了点的添加顺序，从图中可以看出我们的程序没有问题。我们还可以从图中看出添加新点对插值曲线的变化过程的影响。

4 龙格现象

我们先用一段代码来查看一下龙格现象的产生。考虑以下问题：

对函数 $f(x)=\frac{1}{1+25x^2}, x\in[-1,1]$ 进行 Lagrange 插值，使用等距节点进行插值。插值点的个数分别为 3、5、7、9、11、13 个。

```
f<-function(x){
  return(1/(1+25*x^2))
}

## 此函数对于给定的节点数，对函数 f 画出等距节点下的插值图
Runge<-function(nodes){
  x<-seq(-1,1,length.out=nodes)
```

```
  y<-f(x)
  f_approx<-Lagrange(x,y)
  x_approx<-y_approx<-seq(-1,1,length.out=100)
  for(i in 1:length(x_approx)) y_approx[i]<-f_approx(x_approx[i])
  hb<-max(y_approx)+0.5;lb<-min(y_approx)-0.5
  plot(x,y,ylim=c(lb,hb),main=paste(nodes,'nodes'))
  lines(x_approx,y_approx,type='l')
  curve(f,add=T,lty=2)
}

par(mfrow = c(3,2))
par(mar=c(2,6,2,2))
Runge(3);Runge(5)
Runge(7);Runge(9)
Runge(11);Runge(13)
```

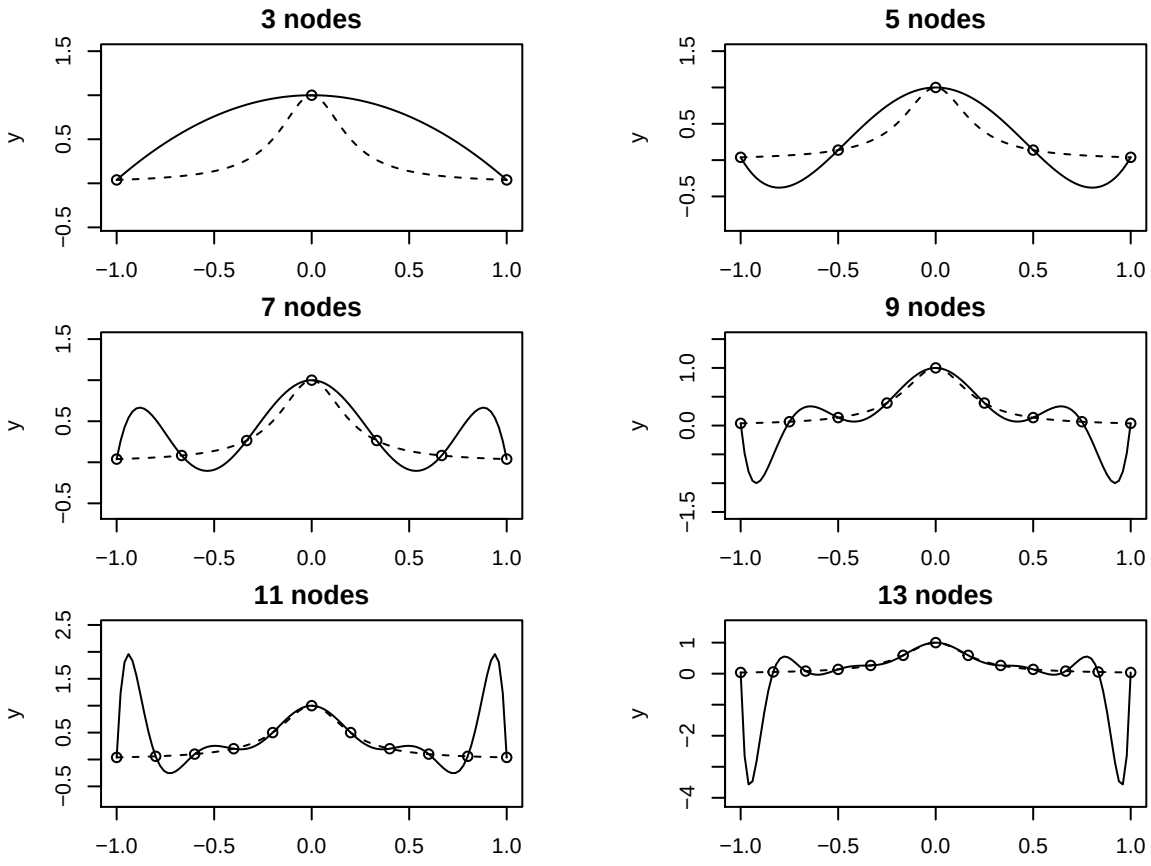

图 3: 龙格现象实例

从图中可以看出，当插值节点个数在 3、5、7 个时，插值的效果尚可；但当插值点数增加时，插值曲线会强烈地震荡，这会极大地影响插值函数在部分点的插值精度与效果。这种震荡现象就是龙格现象。

4.1 魏尔斯特拉斯定理

先来看一个定理，这个定理在第三章函数逼近中也会出现。

Weierstrass Approximation Theorem: 如果 $f(x) \in C[a,b]$，那么对 $\forall \epsilon > 0$,$\exists$ 多项式函数 $p(x), s.t. \max\limits_{a\le x\le b}|f(x)-p(x)| < \epsilon$ 在 $[a,b]$ 上一致成立。假设 $p_n(x)$ 表示次数为 n 的多项式集合，则这个定理等价于：

$$\lim_{n\to\infty}(\max_{a\le x\le b}|f(x)-p_n(x)|)=0$$

这表明，我们只要用次数更高的多项式去插值，我们就越有可能更好地去近似原函数。但实际上，随着 n 的增加，以这种方式产生的 $p_n(x)$ 可能与 $f(x)$ 背离，尤其会以一种振荡模式出现在插值点的末端。

但这与定理**并不矛盾**。这是因为该定理仅说明存在一组多项式函数可以一致近似待插值函数，但并没有提供找到一个多项式的通用方法。

而对于函数 $f(x)=\frac{1}{1+25x^2}, x\in[-1,1]$，若使用插值点 $x_i=\frac{2i}{n}-1, i=0,1...n$ 进行插值，我们却可以得到：

$$\lim_{n\to\infty}(\max_{-1\le x\le 1}|f(x)-p_n(x)|)=+\infty$$

4.2 龙格现象的产生原因

我们知道，多项式插值的余项是：

$$R(x)=\frac{f^{(n+1)}(\xi)}{(n+1)!}\omega_n(x)$$

记 $M_{n+1}=\max\limits_{a\le x\le b}|f^{(n+1)}(x)|$, $W_{n+1}=\max\limits_{a\le x\le b}|\omega_n(x)|$

我们非常容易证明，对于等距插值节点，我们有 $W_{n+1}\le n!h^{n+1}$。因此，对于对于函数 $f(x)=\frac{1}{1+25x^2}, x\in[-1,1]$，我们有 $W_{n+1}\le n!(\frac{2}{n})^{n+1}$。而对于 $f(x)$，我们又有 $M_{n+1}\le(n+1)!5^{n+1}$。因此：

$$\max_{-1\le x\le 1}|f(x)-p_n(x)|\le n!(\frac{10}{n})^{n+1}\to\infty(n\to\infty)$$

即右侧不能收敛于 0!!!

更一般的，龙格现象产生的原因有两条：

1. 当 n 增加时，此特定函数的 $n+1$ 阶导数的大小快速增长（即 M_{n+1} 太大）。
2. 当 n 增加时，点之间的等距导致 Lebesgue 常数迅速增加（即 W_{n+1} 太大）。

其中，Lebesgue 常数是一个衡量插值函数近似效果的值，该值越大则震荡效果越明显。

4.3 避免龙格现象

避免龙格现象的方法很多，比如书上所学的分段低次插值。此外，如果我们可以自由地选择插值点的话，我们有一种简单的方法：改变插值点。

特别地，经过研究发现：**使用切比雪夫点**插值可以在一定程度上避免龙格现象。

我们从之前等距节点的震荡图中可以看出，震荡的区域基本存在于区间的两端。因此一个直观的想法是：我们在区间的两端处把插值点安排的密一些，从而引导插值函数在区间两端表现平稳。切比雪夫点就使用了三角函数将原来的等距节点变换为了“外密内松”的插值节点。

定义在 $[a,b]$ 上的切比雪夫点是 $\frac{1}{2}(a+b)+\frac{1}{2}(b-a)\cos(\frac{2k-1}{2n}\pi), k=1,2,...,n$。

我们来写一段程序验证一下，我们同时使用 11 个节点分别进行等距插值和切比雪夫插值：

```
par(mfrow=c(1,2))
## 切比雪夫点插值
x<-cos(pi*(2*seq(1,11)-1)/(2*11))
y<-f(x)
f_approx<-Lagrange(x,y)
x_approx<-y_approx<-seq(-1,1,length.out=100)
for(i in 1:length(x_approx)) y_approx[i]<-f_approx(x_approx[i])
hb<-max(y_approx)+0.5
lb<-min(y_approx)-0.5
plot(x,y,ylim=c(lb,hb),main='Chebyshev Nodes')
lines(x_approx,y_approx,type='l')
curve(f,add=T,lty=2)

## 等距节点插值
x<-seq(-1,1,length.out=11)
y<-f(x)
f_approx<-Lagrange(x,y)
x_approx<-y_approx<-seq(-1,1,length.out=100)
for(i in 1:length(x_approx)) y_approx[i]<-f_approx(x_approx[i])
hb<-max(y_approx)+0.5
lb<-min(y_approx)-0.5
plot(x,y,ylim=c(lb,hb),main='Equidistant Nodes')
lines(x_approx,y_approx,type='l')
curve(f,add=T,lty=2)
```

效果拔群！但这是为什么呢？让我们回到 Lebesgue 常数 Λ_n。

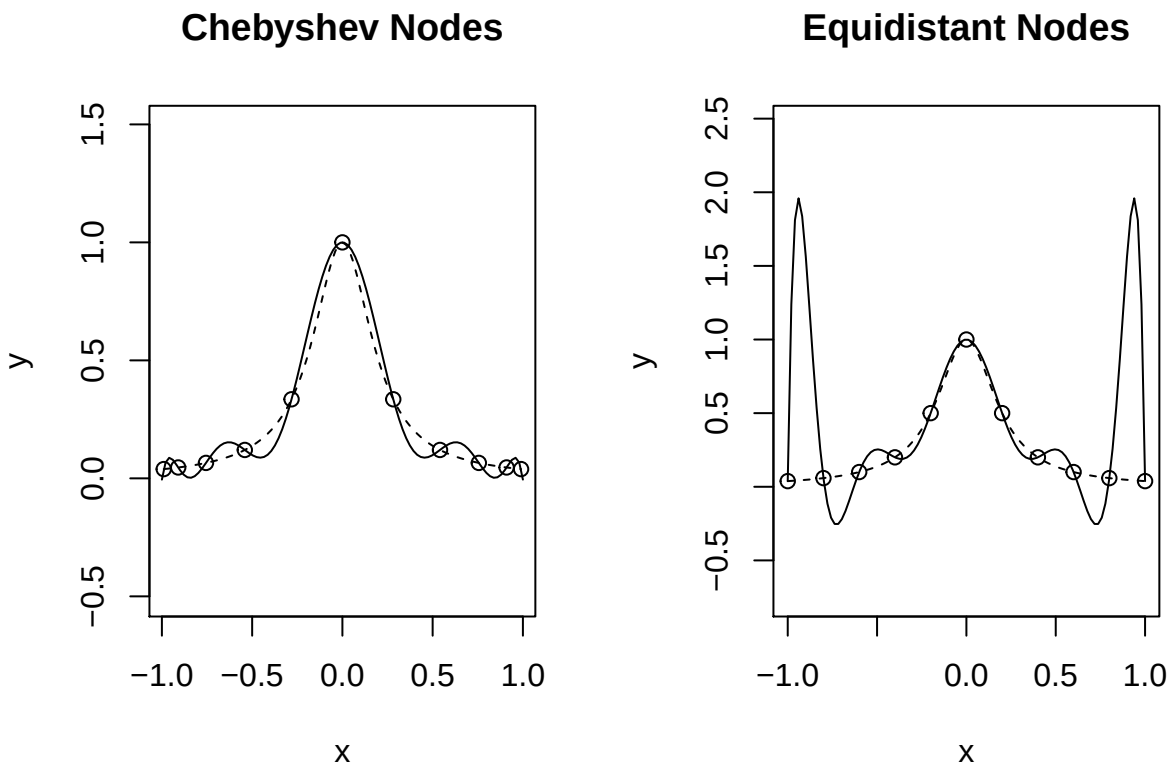

图 4: 切比雪夫点插值与等距点插值

对于等距插值节点我们有:

$$\Lambda_n \sim \frac{2^{n+1}}{en\log n}(n\to\infty)$$

而对于切比雪夫插值节点我们则有:

$$\frac{2}{\pi}\log(n+1)+\alpha<\Lambda_n<\frac{2}{\pi}\log(n+1)+1,\alpha=0.9625...$$

一个和无穷大同阶无穷大，一个却对数缓慢增长，原来那么神奇!!!

不过事实上，还有一个更为直接，更为简单，更为明了的解释。大家在接下来的第三章函数逼近中就会知道了：因为切比雪夫节点插值代表着函数的**最佳一致逼近**。

5 参考

1. Runge's phenomenon. Wikipedia.
2. Lebesgue constant. Wikipedia.