数值分析第三章补充阅读

朱宇嘉

1 函数逼近

函数逼近主要解决以下问题：

对于一个复杂的函数 $f(x)$，我们希望找到一个简单的多项式函数 $P_n(x)$，使得在函数 $f(x)$ 的整个定义域上，$f(x)$ 与 $P_n(x)$ 在某种度量误差下达到最小。

我们做以下几点补充：

1. 事实上，函数逼近问题不一定必须用多项式函数去逼近复杂函数，也可以使用其他形式的函数族去逼近。
2. 与插值问题不同，插值问题是在给定的点上完美契合原函数以试图在整个定义域上逼近原函数；而函数逼近则是直接在整个定义域上全局考虑以试图直接逼近原函数。
3. 误差度量的方式有很多种。其中最为常用的是最优一致逼近和最佳平方逼近。而教材上的最小二乘拟合其实就是多项式回归，不过教材上会使用正交多项式的方法来加速运算。

本章，我们主要用编程语言来实现最佳一致逼近和最佳平方逼近。其中最佳一致逼近使用 Python 语言实现，最佳平方逼近使用 R 语言实现。我们还将对课程上未涉及的任意函数的最佳一致逼近进行简单的过程描述。

2 最佳一致逼近多项式

最佳一致逼近多项式是指：对于定义在 $[a,b]$ 函数 $f(x)$，我们希望从多项式族 $H_n = \{P_n(x), n \in Z^+\}$ 中找到 $P^*(x)$，使得:

$$||f(x) - P^*(x)||_\infty = \min_{P \in H_n} ||f(x) - P(x)||_\infty = \min_{P \in H_n} \max_{a \le x \le b} |f(x) - P(x)|$$

2.1 n 次多项式的 $n-1$ 次最佳一致逼近多项式的 Python 实现

最佳一致逼近多项式的一个简单例子就是用一个 $n-1$ 次的多项式去逼近一个 n 次的多项式。我们可以简单地利用切比雪夫多项式去构造最佳一致逼近，具体步骤如下：

- **Step 1：** 对 $f(x)$ 的定义域做变换 $g:[a,b]\to[-1,1]$，使其变为 $[-1,1]$ 上的 n 次多项式。
- **Step 2：** 得到 n 次的切比雪夫多项式 $T_n(x)$，对其数乘并与 $f(x)$ 相减，消去 n 次项，得到 $P^*_{n-1}(x)$。
- **Step 3：** 将 $P^*_{n-1}(x)$ 的定义域从 $[-1,1]$ 变回 $[a,b]$ 即可。

我们使用 numpy 的数组来存储并表示多项式，例如：[2,3,1] 表示多项式 $g(x)=x^2+3x+2$，[-1,2,1,2] 表示多项式多项式 $g(x)=2x^3+x^2+2x-1$。我们首先用函数写出定义在 $[-1,1]$ 上的切比雪夫多项式函数。

```
## 切比雪夫多项式
import numpy as np

def Chebyshev(n):
    assert isinstance(n, int) and n >= 0, 'n should be a non-negative integer'
    if n == 0: return np.array([1])
    elif n == 1: return np.array([0,1])
    else:
        return np.append(0,2 * Chebyshev(n-1)) - np.append(Chebyshev(n-2), [0,0])

for i in range(5):
    print(i," 次切比雪夫多项式的系数是: ", Chebyshev(i))
```

```
## 0 次切比雪夫多项式的系数是:  [1]
## 1 次切比雪夫多项式的系数是:  [0 1]
## 2 次切比雪夫多项式的系数是:  [-1  0  2]
## 3 次切比雪夫多项式的系数是:  [ 0 -3  0  4]
## 4 次切比雪夫多项式的系数是:  [ 1  0 -8  0  8]
```

我们打印出 0-4 次的切比雪夫多项式的系数值。接下来我们就要使用这一函数来完成最佳一致逼近函数的编写。

```
from scipy.special import comb

def Comb(n, a, b):
    # 求 (a+b)^n 展开式的各项系数
    out = [comb(n,i)* b**(n-i) * a**i for i in range(n+1)]
    return np.array(out)

def tranf(f, a, b):
    # 求对多项式函数 f 做 ax+b 变换后的系数
    n = len(f)
    out = np.zeros(n)
```

```
    for i in range(n):
        out += np.append(f[i]*Comb(i,a,b), np.zeros(n-1-i))
    return out

def PolyBUA(f, lo, hi):
    # f 是输入的 n 次多项式，其是一个 numpy.ndarry 类型数据
    # lo,hi 是区间的下限和上限
    assert isinstance(f, np.ndarray), 'should input a numpy array'
    assert len(f) >=2, 'degree at least 2'
    assert f[-1] != 0, 'highest order coefficient should not equal to 0'
    assert lo < hi, 'interval length should be positive'

    # 将区间从 [lo,hi] 变为 [-1,1]
    a = (hi-lo)/2
    b = (hi+lo)/2
    tran_f = tranf(f, a, b)

    # 减去切比雪夫多项式
    cheb = Chebyshev(len(f)-1)
    tran_f -= cheb/cheb[-1]*tran_f[-1]

    # 将区间从 [-1,1] 变为 [lo,hi]
    a = 2/(hi-lo)
    b = -(hi+lo)/(hi-lo)
    out = tranf(tran_f, a, b)

    return out[:-1]
```

在最佳一致逼近多项式的编写中，为了简化代码的重复功能，我们还编写了 2 个辅助函数 Comb() 和 tranf()，其作用分别是求 $(a+b)^n$ 展开式的各项系数和求对多项式函数 $f(x)$ 做变换:$ax+b$ 后的系数。PolyBUA() 是求最佳一致逼近多项式的主函数，其输入一个 n 次多项式和其定义域 $[lo, hi]$，输出 $n-1$ 次最佳一致逼近多项式。我们用课件上例 3 来验证一下函数的正确性。

```
f = np.array([-1,2,1,2])
print(" 原函数是: ",f,", 最佳一致逼近函数是: ",PolyBUA(f,-1,1))
```

```
## 原函数是:  [-1  2  1  2] ,最佳一致逼近函数是:  [-1.   3.5  1. ]
```

得到的结果如下：原函数是 $f(x)=2x^3+x^2+2x-1$，最佳一致逼近多项式是 $P_2^*(x)=x^2+3.5x-1$。

2.2 任意函数的最佳一致逼近多项式

前面我们编写的函数只是最佳一致逼近多项式的一个特殊情况，是用一个 $n-1$ 次的多项式去逼近一个 n 次的多项式。但我们又如何用最佳一致逼近多项式去逼近一个任意函数呢？我们这里只说结果不说过程。

假设 $f(x)$ 是定义在 $[-1,1]$ 上的**任意函数**，则其 n 次最佳一致逼近多项式 $S_n^*(x)$ 为：

$$S_n^*(x)=\frac{C_0^*}{2}+\sum_{k=1}^{n}C_k^*T_k(x)$$

这是将 $f(x)$ 在 $[-1,1]$ 上在切比雪夫正交多项式基 $\{T_k(x),k=0,1,2...\}$ 上进行展开所得级数的前 n 项。其中：

$$C_k^*=\frac{2}{\pi}\int_{-1}^{1}\frac{f(x)T_k(x)}{\sqrt{1-x^2}}dx,k=0,1,2...$$

$$T_k(x)=\cos(k\arccos x),|x|\leq 1,k=0,1,2...$$

我们依次计算 C_k^* 即可。但是需注意因为 $f(x)$ 的表达式可能十分复杂，计算这一积分可能十分困难，我们可能需要借助下一章的数值积分法来进行计算。

3 最佳平方逼近

最佳平方逼近是指：对于定义在 $[a,b]$ 函数 $f(x)$，我们希望从正交函数族 $\Phi=span\{\phi_0,\phi_1...,\phi_n\}$ 中找到 $s^*(x)\in\Phi$，使得:

$$||f(x)-s^*(x)||_2=\min_{s\in\Phi}||f(x)-P(x)||_2=\min_{s\in\Phi}\int_a^b\rho(x)[f(x)-s(x)]^2dx$$

一般情况下，为了简化该问题，我们会假定 $\rho(x)=1$。而因为这个权函数和勒让德函数族的权函数一致，因此勒让德函数在 $\rho(x)=1$ 的假定下求最佳平方逼近时有着十分重要的作用。

下面，我们均假设 $\rho(x)=1$。在此条件下，我们有两种求最佳逼近的方法：分别是使用正交函数族和不使用正交函数族。我们接下来分别对这两种方法进行 R 语言实现。

3.1 Hilbert 矩阵最佳平方逼近的 R 语言实现

在 $\rho(x)=1$ 的假定下，我们继续假定 $\phi_k(x)=x^k$，函数区间为 $[0,1]$。在这些条件下，函数 $f(x)$ 的 n 次最佳平方逼近可如下求得：

- **Step 1**：计算法方程的目标向量 $d=(d_0,d_1,...,d_n)$，其中 $d_k=(f(x),\phi_k(x))=\int_0^1x^kf(x)dx$。
- **Step 2**：使用 Hibert 矩阵求解线性方程组得到 $\phi_k(x)$ 的系数 a_k。

我们可以用以下 R 语言函数来实现这一求解过程。

```
HilbertBSA<-function(f,n){
  ## 输入: f 是一个 function, 表示待逼近函数, 区间为 [0,1]
  ## 输入: n 是逼近多项式的最高次数
  if(!is.function(f)) stop("f should be a function")
  if(n!=round(n) | n<0) stop("n should be a non-negative integer")
  ## 计算 d
  d<-rep(0,n+1)
  for(i in 0:n){
    Fun<-function(x){
      return(f(x)*x^i)
    }
    d[i+1]<-integrate(Fun,0,1)$value
  }
  ## 生成希尔伯特矩阵
  H<-1/outer(seq(0,n),seq(1,n+1),'+')
  ## 计算系数
  a<-solve(H)%*%d
  ## 返回系数
  return(a)
}
```

这个函数阅读起来十分容易，在这里就不做过多的解读了。我只强调一点，因为算法的过程中涉及到积分，所以在函数中使用了 R 语言内置的 integrate() 函数对函数做积分（而且这里做的**数值积分**）。这是因为待逼近函数 $f(x)$ 可能十分复杂，直接积分相当困难。更多关于数值积分的内容会在下一讲展开。

下面我们对课件上的例 6 做验证：

```
f<-function(x){
  return(sqrt(1+x^2))
}
HilbertBSA(f,1)
```

```
##           [,1]
## [1,] 0.9343200
## [2,] 0.4269471
```

如结果所示，$f(x)=\sqrt{1+x^2}, x\in[0,1]$ 的一次最佳平方逼近多项式是 $s_1^*(x)=0.934+0.427x$。

3.2 正交函数族最佳平方逼近的 R 语言实现

接下来，我们用正交函数族来作最佳平方逼近。我们这里讨论的是用正交多项式函数族来做最佳函数逼近。我们可以通过以下步骤来求 $f(x), x \in [a,b]$ 的 n 次最佳逼近多项式。

- **Step 1**：对 $f(x)$ 的定义域做变换 $g : [a,b] \to [-1,1]$，使其变为 $[-1,1]$ 上的函数。
- **Step 2**：使用勒让德多项式 $\{P_n(x), n = 1,2,...\}$ 计算多项式系数：$a_k^* = \frac{2k+1}{2}(f(x), P_k(x)) = \frac{2k+1}{2}\int_{-1}^{1} f(x)P_k(x)dx$。
- **Step 3**：将勒让德多项式线性组合，得到 $[-1,1]$ 上的最佳逼近函数 $s_n^*(x) = \sum_{i=0}^{n} a_k^* P_k(x)$。
- **Step 4**：将 $s_n^*(x)$ 的定义域从 $[-1,1]$ 变回 $[a,b]$ 即可。

同样的，我们用 vector 来表示多项式。例如：[2,3,1] 表示多项式 $g(x) = x^2 + 3x + 2$，[-1,2,1,2] 表示多项式多项式 $g(x) = 2x^3 + x^2 + 2x - 1$。我们首先用函数写出定义在 $[-1,1]$ 上的勒让德多项式函数。由于之后要用 integrate 函数进行积分运算，所以这里不仅要写一个导出系数的 Legendre() 函数，还要写一个能对特定 x 返回计算结果的 Legendre_fun() 函数。

```
Legendre<-function(n){
  if(n!=round(n) | n<0) stop("n should be a non-negative integer")
  if(n==0) return(c(1))
  else if(n==1) return(c(0,1))
  else
    return(c(0,(2*n-1)/n*Legendre(n-1))-c((n-1)/n*Legendre(n-2),c(0,0)))
}

Legendre_fun<-function(n,x){
  if(n!=round(n) | n<0) stop("n should be a non-negative integer")
  if(n==0) return(1)
  else if(n==1) return(x)
  else
    return(x*(2*n-1)/n*Legendre_fun(n-1,x)-(n-1)/n*Legendre_fun(n-2,x))
}

for(i in 0:4) cat(i," 次勒让德多项式的系数是: ", Legendre(i),"\n",sep=" ")
```

```
## 0 次勒让德多项式的系数是:  1
## 1 次勒让德多项式的系数是:  0 1
## 2 次勒让德多项式的系数是:  -0.5 0 1.5
## 3 次勒让德多项式的系数是:  0 -1.5 0 2.5
## 4 次勒让德多项式的系数是:  0.375 0 -3.75 0 4.375
```

我们同样打印出 0-4 次勒让德多项式的系数。接下来，我们同样基于此来编写正交函数族最佳平方逼近

多项式。同样我们需要 2 个辅助函数在线性变换下求变换后的多项式系数。

```
Comb<-function(n,a,b){
  # 求 (a+b)^n 展开式的各项系数
  out<-seq(0,n)
  out<-choose(n,out)*a^out*b^(n-out)
  return(out)
}

tranf<-function(f,a,b){
  # 求对多项式函数 f 做 ax+b 变换后的系数
  n<-length(f)
  out<-rep(0,n)
  for(i in 1:n){
    out<-c(f[i]*Comb(i-1,a,b),rep(0,n-i))+out
  }
  return(out)
}

BSA<-function(f,lo,hi,n){
  ## 输入: f 是待逼近函数
  ## 输入: lo,hi 是函数定义域的上限和下限
  ## 输入: n 是逼近函数的最高项系数
  if(!is.function(f)) stop("f should be a function")
  if(lo>=hi) stop("interval length at least 0")
  if(n!=round(n) | n<0) stop("n should be a non-negative integer")
  # 将区间从 [lo,hi] 变为 [-1,1]
  k<-(hi-lo)/2
  b<-(hi+lo)/2
  # 计算系数 a
  a<-rep(0,n+1)
  for(i in 0:n){
    Fun<-function(x){
      return(Legendre_fun(i,x)*f(k*x+b))
    }
    a[i+1]<-(2*i+1)/2*integrate(Fun,-1,1)$value
  }
  # 线性组合勒让德多项式
  lc<-rep(0,n+1)
```

```
  for(i in 1:(n+1)){
    lc<-lc+c(a[i]*Legendre(i-1),rep(0,n+1-i))
  }
  # 将区间从 [-1,1] 变为 [lo,hi]
  k<-2/(hi-lo)
  b<--(hi+lo)/(hi-lo)
  out<-tranf(lc,k,b)
  # 返回结果
  return(out)
}
```

同样的，我们使用课件上的例 7 来验证我们的程序。

```
f<-function(x){
  return(exp(x))
}
BSA(f,-1,1,3)
```

```
## [1] 0.9962940 0.9979549 0.5367215 0.1761391
```

结果显示 $f(x) = e^x, x \in [-1, 1]$ 的三次最佳平方逼近多项式是 $s^*(x) = 0.9963 + 0.9980x + 0.5367x^2 + 0.1761x^3$。从中，我们发现课件上有一个打印错误。