计算物理 III

数值积分

Author[a,1]

[a] 武汉大学，物理科学与技术学院

Keywords—*Lagrange polynomial,Trapezoidal Rule，Simpson’s rule，Python*

Contents

1. Lagrange Polynomial

如果我们想要从一组离散的实数点拟合一个函数，代数多项式（algebraic polynomials）是一种常用且有效的基组。根据魏尔斯特拉斯逼近定理，代数多项式可以逼近闭区间上的任意连续函数 [1]; 另一方面，代数多项式是可导的并且容易求出不定积分。

$$P_n(x) = a_0 + a_1x^1 + \cdots + a_nx^n \tag{1}$$

Weierstrass Approximation Theorem:

假设 f 是定义在区间 [a,b] 上的连续函数，那么对任意 ε 存在 P(x) 使得:

$$|f(x) - P(x)| < \varepsilon$$

关于其中的系数 $\{a_0, a_1, \cdots, a_n\}$，你可能首先想到使用 Taylor 公式求出。尽管 Taylor 公式在 x_0 点的邻域是准确的，但是一个好的多项式需要在项数尽可能低的情况下，在整个区间都保持良好的精度。

最简单的是只有两个点的情况，此时拟合的结果是一条直线，显然 a_0 和 a_1 分别是它的截距与斜率。考虑两个已知的点 (x_0, y_0) 和 (x_1, y_1).

$$P(x) = L_0(x)f(x_0) + L_1(x)f(x_1) \tag{2}$$

$$L_0(x) = \frac{x - x_1}{x_0 - x_1}, \qquad L_1(x) = \frac{x - x_1}{x_0 - x_1}$$

受到上式的启发，我们可以将由 n+1 个点得到的插值多项式写成:

$$P_n(x) = \sum_{k=0}^{n} L_{n,k}(x)f(x_k) \tag{3}$$

其中

$$L_{n,k}(x) = \begin{cases} 1 & \text{if } x = x_k, \\ 0 & \text{if } x = x_i, i \neq k \end{cases} \tag{4}$$

不难验证

$$L_{n,k} = \prod_{i=0,i\neq k}^{n} \frac{x - x_i}{x_k - x_i} \tag{5}$$

对于这种插值方式，我们还可以得到它的误差范围:
考虑区间 [a,b] 的 n+1 个确定的点 $x_0, x_1, \cdots, x_n$，f 是定义在 [a,b] 的连续可微函数，那么对 [a,b] 上的任意 x，存在一个 $\xi \in [a, b]$:

$$f(x) = P_n(x) + \frac{f^{(n+1)}(\xi)}{(n+1)!}\prod_{i=0}^{n}(x - x_i) \tag{6}$$

2. Numerical Integral

如果要计算函数 f（x）在区间 [a,b] 的数值积分，最简单的方法是将 [a,b] 划分为许多小区间，计算每一个区间的 $f(x_i)\Delta x$ 然后相加，这种方法称为矩形法。可以预想，当划分的 Δx 足够小时，这种方法将足够精确。但是，上一节关于拉格朗日多项式的方法启发我们，当我们有 n 组 $\{f(x_i), x_i\}$ 时，存在更加精确的算法。

$$\int_a^b f(x)dx = \int_a^b P_n(x)dx + \frac{1}{(n+1)!}\int_a^b \prod_{i=0}^{n}(x - x_i)f^{(n+1)}(\xi)dx \tag{7}$$

因为 ξ 是未知的，因此在计算时舍掉后半部分（误差是已知的），根据 n 的不同，常用的两种方法是梯形法 (n=1) 和辛普森法 (n=2)。

2.1. 梯形法则（Trapezoidal rule）

Trapezoidal rule

$$\int_a^b f(x)dx = \frac{h}{2}[f(a) + f(b)] - \frac{h^3}{12}f\varepsilon(\xi) \tag{8}$$

其中 $h = (b - a)/2$.

2.2. Simpson’s rule

Simpson’s rule

$$\int_a^b f(x)dx = \frac{h}{3}[f(a) + 4f(\frac{a+b}{2}) + f(b)] - \frac{h^5}{90}f^{(4)}(\xi) \tag{9}$$

在实际计算 f(x) 在区间 [a,b] 的积分时，首先将 [a,b] 划分为 n 个小区间 $[x_i, x_{i+1}]$，然后使用梯形法或者辛普森法求出每个小区间的积分，并求和。

$$\int_a^b f(x)dx = \sum_{i=0}^{n}\int_{x_i}^{x_{i+1}} f(x)dx$$

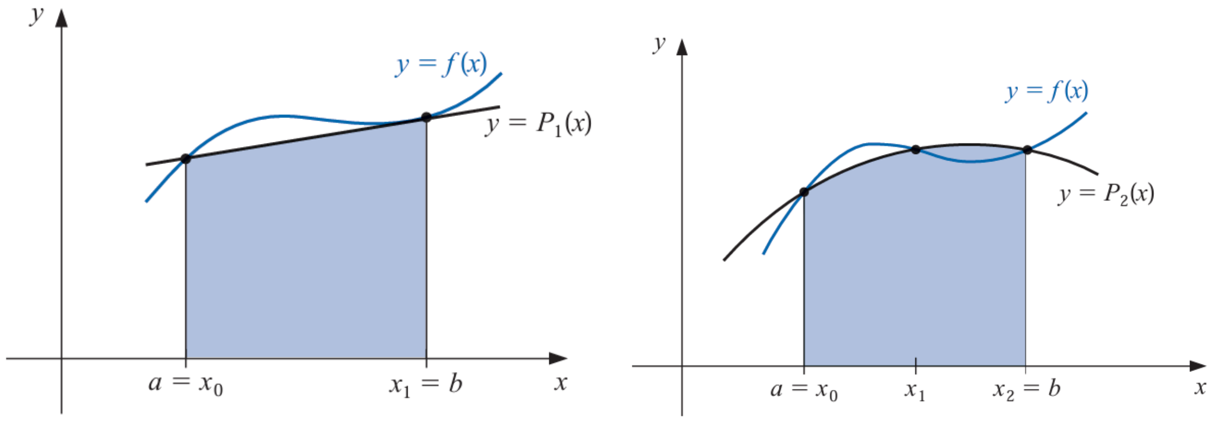
下面给出一个使用梯形法和辛普森法计算数值积分的 Python 代码实例。

2.3. Example

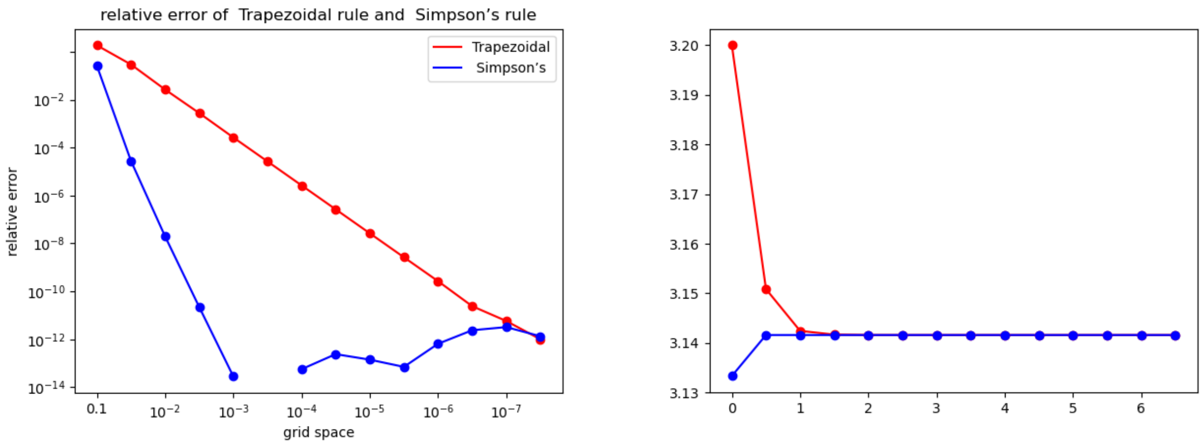
Formula

$$I = \int_0^1 \frac{1}{1+x^2} = \pi \tag{10}$$

```
import numpy as np
def f(x):
    return 4/(1+x**2)
def trapzoidal(N):
    x1,x2=0.,1.
```



(a) Trapezoidal rule[2]　　**(b)** Simpson's rule



(a) 误差　　**(b)** 梯形法和辛普森法在不同格点数下计算的积分值

```
    I=0
    d=(x2-x1)/N
    x=np.arange(x1,x2+d,d)
    for i in range(0,N):
        m=(x[i]+x[i+1])/2
        I+=f(m)*d
    return I
def simpson(N):
    x1,x2=0.,1.
    I=0
    d=(x2-x1)/N
    x=np.arange(x1,x2+d,d)
    for i in range(0,N):
        m=(x[i]+x[i+1])/2
        I+=(d/6)*(f(x[i])+f(m)*4+f(x[i+1]))
    return I
```

Code 1. code

3. Monte Carlo Method

蒙特卡罗方法又称随机模拟法或统计试验法,是通过随机变量的统计实验求解数学物理问题的数值方法. 用蒙特卡罗方法解决实际问题时,无论是随机性问题或是确定性问题需要把具体问题看做某一随机事件,在计算机上选用恰当的数学模型进行模拟. 其基本特点有:

1. 程序结构简单,占内存少
2. 收敛慢,但其收敛速度与误差和维数无关,这是其他方法不具备的优点
3. 边界条件复杂不带来求解的复杂性

设区间 [a,b] 中的随机变量 的分布由密度函数 f(x) 给出,g(x) 是定义在 [a,b] 上的连续函数. 则给 g(ξ) 的数学期望:

$$E(g(\xi)) = \int_a^b g(x)f(x)dx \tag{11}$$

同时，根据大数定律

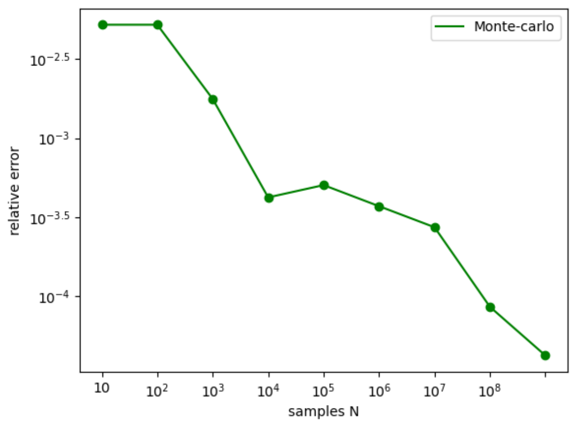
$$E = \lim_{N\to\infty} \sum_{i=1}^{N} \frac{g(x_i)}{N} \tag{12}$$

通常，为了方便计算选择 f(x) 为均匀分布的概率密度，从而

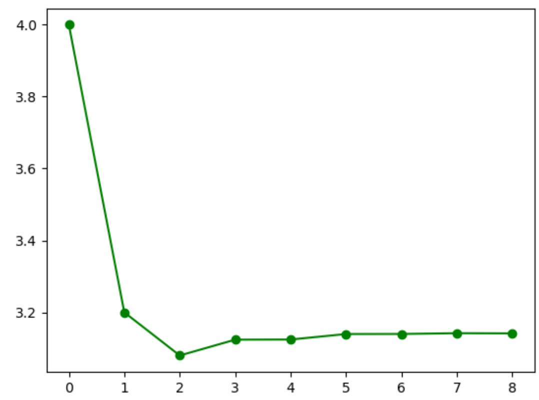
$$\int_a^b g(x)dx = (b-a)\int_a^b \frac{g(x)}{b-a}dx = (b-a)\lim_{N\to\infty}\sum_{i=1}^{N}\frac{g(x_i)}{N} \tag{13}$$

这个结果不难推广到二维:

$$\iint H(x,y)dxdy = A\iint \frac{H(x,y)}{A}dS = A\lim_{N\to\infty}\sum_{i=1}^{N}\frac{H(x_i,y_i)}{N} \tag{14}$$



(a) 蒙特卡洛法的误差



(b) 不同试验次数蒙特卡洛法的积分值

Example

$$\int_{-1}^{1}\int_{-1}^{1} H(x,y) = 4\lim_{N\to\infty}\sum_{i=1}^{N}\frac{H(x_i,y_i)}{N} \tag{15}$$

$$H(x,y)(x) = \begin{cases} 1 & \text{if} \quad x^2+y^2 \leqslant 1 \\ 0 & \text{if} \quad else \end{cases}$$

```
import numpy as np
def H(x,y):
    if x**2+y**2>=1:
        return 0
    else:
        return 1
def MonteCarlo(N):
    j=0
    x=-1+2*np.random.rand(N)
    y= -1+2*np.random.rand(N)
    z=np.random.rand(N)
    for i in range(0,N):
        if H(x[i],y[i])>=z[i]:
            j+=1
    I=j/N*4
    return I
```

Code 2. code

4. Codes

以下是全部的实例代码以及 matploylib 绘图部分：

```
import numpy as np
import matplotlib.pyplot as plt
def f(x):
    return 4/(1+x**2)
def H(x,y):
    if x**2+y**2>=1:
        return 0
    else:
        return 1
def trapzoidal(N):
    x1,x2=0.,1.
    I=0
    d=(x2-x1)/N
    x=np.arange(x1,x2+d,d)
    for i in range(0,N):
        m=(x[i]+x[i+1])/2
        I+=f(m)*d
    return I
def simpson(N):
    x1,x2=0.,1.
    I=0
    d=(x2-x1)/N
    x=np.arange(x1,x2+d,d)
    for i in range(0,N):
        m=(x[i]+x[i+1])/2
        I+=(d/6)*(f(x[i])+f(m)*4+f(x[i+1]))
    return I


def MonteCarlo(N):
    j=0
    x=-1+2*np.random.rand(N)
    y= -1+2*np.random.rand(N)
    for i in range(0,N):
        if H(x[i],y[i])>0:
            j+=1
    I=4*j/N
    return I


def err1(N):
    return np.abs(trapzoidal(N)-np.pi)/np.pi
def err2(N):
    return np.abs(simpson(N)-np.pi)/np.pi
def err3(N):
    return np.abs(MonteCarlo(N)-np.pi)/np.pi
def fit():
    fig,ax1=plt.subplots()
    xx=np.arange(0,7,0.5)
    xx1=np.arange(0,9,1)
    y10,y1=[],[]
    for i in range(0,14):
        y10.append(trapzoidal(int(10**(i/2))))
        y1.append(np.log10(err1(int(10**(i/2))))
    )
    y20,y2=[],[]
    for i in range(0,14):
        y20.append(simpson(int(10**(i/2))))
        y2.append(np.log10(err2(int(10**(i/2))))
    )
    y30,y3=[],[]
    for i in range(0,9):
        y30.append(MonteCarlo(10**(i)))
        y3.append(np.log10(err3(10**(i))))
#axisx=np.log10(xx)
    ax1.set_yticklabels(['$10^{-16}$','$10^{-14}$','$10^{-12}$','$10^{-10}$','$10^{-8}$','$10^{-6}$','$10^{-4}$','$10^{-2}$'])
    ax1.set_xticklabels(['1','0.1','$10^{-2}$','$10^{-3}$','$10^{-4}$','$10^{-5}$','$10^{-6}$','$10^{-7}$'])
    ax1.plot(xx,y1,'r',xx,y2,'b')
    ax1.plot(xx,y1,'ro',xx,y2,'bo')
    ax1.set_title('relative error of Trapezoidal rule and  Simpson's rule')
    plt.legend( ('Trapezoidal',' Simpson's '), loc='upper right')
```

```python
    plt.xlabel('grid space')
    plt.ylabel('relative error')
    fig,ax2=plt.subplots()
    ax2.plot(xx1,y3,'g',xx1,y3,'go')
    ax2.set_yticklabels(['$10^{-4.5}$','$10^{-4}$','$10^{-3.5}$','$10^{-3}$','$10^{-2.5}$','$10^{-2}$', '$10^{-1.5}$','$10^{-1}$','$10^{-0.5}$'])
    plt.legend( ('Monte-carlo',),loc='upper right')
    plt.xlabel('samples N')
    plt.ylabel('relative error')
    ax2.set_xticklabels(['1','10','$10^{2}$','$10^{3}$','$10^{4}$','$10^{5}$','$10^{6}$','$10^{7}$','$10^{8}    $'])
    #ax2.set_title('relation between the relative error and the MonteCarlo samples N.')
    fig,ax3=plt.subplots()
    ax3.plot(xx,y10,'r',xx,y20,'b')
    ax3.plot(xx,y10,'ro',xx,y20,'bo')
    fig,ax4=plt.subplots()
    ax4.plot(xx1,y30,'g',xx1,y30,'go')
    plt.show()
if __name__=='__main__':
    fit()
```

Code 3. code

References

[1] Richard L. Burden and J. Douglas Faires. *Numerical Analysis*. Richard Stratton, 2010.

[2] *PGFPlots - A LaTeX package to create plots.* url: https://pgfplots.sourceforge.net/.