Numerical Analysis Project 1

刘浚哲
北京大学物理学院 *1500011370*

November 12, 2018

1. 分别利用 8,16,32,64 个等间距节点, 利用复合梯形公式求解积分方程. 其精确解为 $y = e^{2t}$. 列表显示 $t = 0, 0.25, 0.75, 1$ 处的误差, 比较精确解与数值解的函数图像

解: 积分方程为:

$$y(t) = \int_0^1 (s-t)y(s)\mathrm{d}s + e^{2t} + \frac{e^2-1}{2}t - \frac{e^2+1}{4} \tag{1}$$

8 个等距节点下:

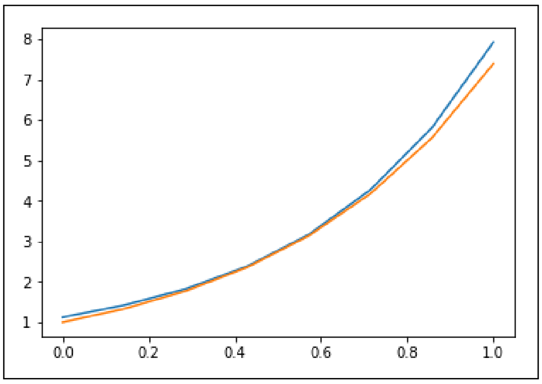

图 1: 8 间距点

16 个等距节点下:

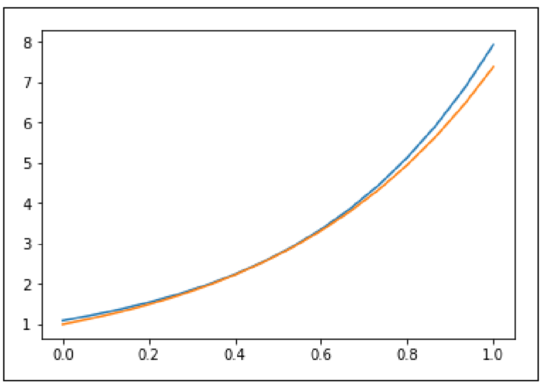

图 2: 16 间距点

32 个等距节点下:

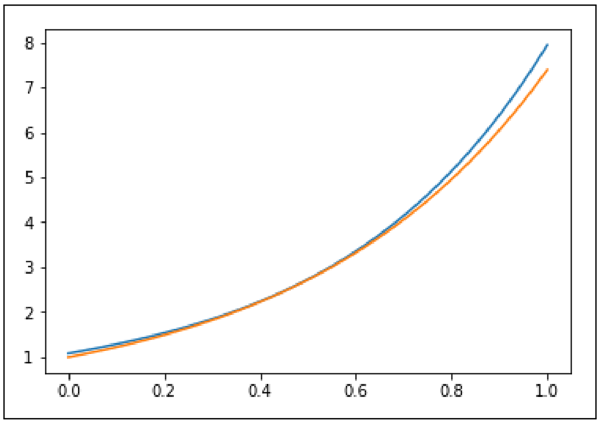

图 3: 32 间距点

64 个等距节点下:

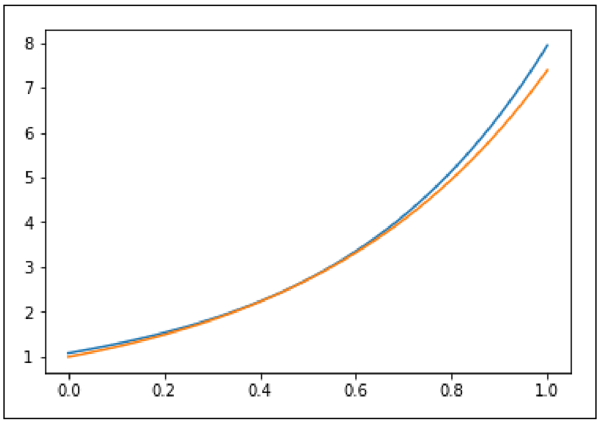

图 4: 64 间距点

由于是等距划分区间, 故我们取区间长度的 1/4 处的函数值作为 0.25 函数值的估计, 同理 3/4 长度处的函数值作为 0.75 函数值的估计, 得到误差估计如下表:

表 1: 误差分析

间距	0.00	0.25	0.75	1.00
8	0.123	0.532	0.174	1.322
16	0.090	0.549	0.092	0.654
32	0.083	0.552	0.060	0.376
64	0.082	0.553	0.047	0.249

可见随着点数的增加, 每个数值解对应的函数值与精确解的误差都相应地减小, 反映在函数图像上, 则是两条图线越来越靠近.

实现代码如下:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
t=np.linspace(0,1,64)
print(t)
y=np.ones_like(t)
yt=np.ones_like(t)

for _ in range(5):
    for i in range(len(yt)):
        yt[i]=np.trapz((t-t[i])*y,t)
    yt+=np.exp(2*t)+(np.exp(2)-1)*t/2-(np.exp(2)+1)/4
    print(np.linalg.norm(yt-y))
    y=yt

plt.plot(t,yt)
plt.plot(t,np.exp(2*t))
plt.show()

print(yt[0]-np.exp(0))
print(yt[63]-np.exp(2))
print(yt[16]-np.exp(2*0.25))
print(yt[48]-np.exp(2*0.75))
```

2. 计算积分:

$$\int_0^1 \sqrt{x}\ln x \mathrm{d}x \tag{2}$$

Composite simpson's rule:

$$\int_{x_0}^{x_2} f(x)\mathrm{d}x = \frac{h}{3}\left[f(x_0) + 2\times\sum_{i}^{n/2-1} f(x_{2i}) + 4\times\sum_{i}^{n/2} f(x_{2i-1}) + f(x_2)\right] \tag{3}$$

实现代码如下:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from math import *

def f(x):
    if x==0:
        return 0
    else :
        return sqrt(x)*log(x)
```

```
def get_n(a, b, h):
    n = int((b-a)/h)
    if n % 2 == 1:
        n = n + 1
    return n

def generate_data(a, b, h):
    n = get_n(a, b, h)
    data = []
    x = a
    for i in range(n):
        data.append(f(x))
        x += h
    return data

def simpson(data, h, n):
    sum = data[0] + data[n-1]
    for i in range(2, n):
        if i % 2 == 0:
            sum += 4 * data[i-1]
        else:
            sum += 2 * data[i-1]
    sum *= h / 3.0
    return sum

a = 0.0
b = 1.0
h = []
simp = []
error = []
for i in range(1, 10):
    h.append(pow(10, -i))
for i in range(9):
    n = get_n(a, b, h[i])
    data = generate_data(a, b, h[i])
    simp.append(simpson(data, h[i], n))
    error.append(simp[i]-4/9)
    print(simp[i])
    print("\n")
plt.plot(h, simp)
plt.xlabel("h")
plt.ylabel("simpson")
```

这里我们先设置步长是 10 的幂次级别, 在宏观尺度下观察步长与误差之间的差距, 得到数值积分值与实际值差别如下:

步长	误差
0.1	0.024659090319336552
0.01	$7.506457224237262\times10^{-4}$
0.001	$2.748662588897277\times10^{-5}$
1×10^{-4}	$1.0345151118529294\times10^{-6}$
1×10^{-5}	$3.8409069702538545\times10^{-8}$
1×10^{-6}	$1.3991085512365942\times10^{-9}$
1×10^{-7}	$-6.381450923242937\times10^{-12}$
1×10^{-8}	$4.7945253323078425\times10^{-10}$

可见步长在 $1\times10^{-7}\sim1\times10^{-8}$ 区间, 误差增大, 我们研究这一部分的积分值, 得到如下结果:

步长	积分值
1×10^{-8}	-0.4444444439649919
2×10^{-8}	-0.44444444462006527
3×10^{-8}	-0.44444444431825436
4×10^{-8}	-0.44444444455258036
5×10^{-8}	-0.4444444444277282
6×10^{-8}	-0.4444444444338511
7×10^{-8}	-0.4444444443507025
8×10^{-8}	-0.4444444444336884
9×10^{-8}	-0.4444444443710892

得到误差值随步长的变化如下图:

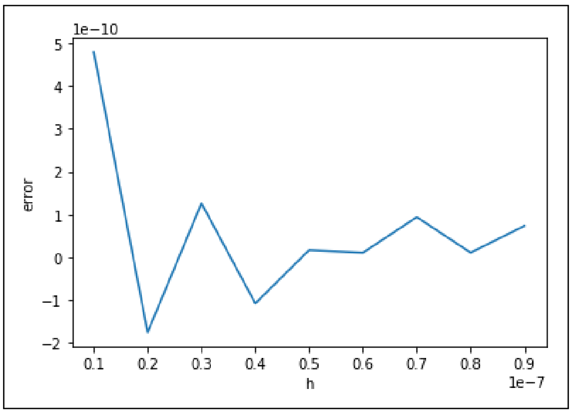

图 5: 误差随步长变化曲线

从图上可见, 当步长从 1×10^{-7} 减小到 1×10^{-8} 时, 误差一直处于波动状态, 此时误差已经开始逐渐增加, 故当 $h=5\times10^{-7}$ 时, 可认为是误差逐渐增加的转折点.

Romberg Integral

龙贝格算法递推公式为:

$$R_n^k = \frac{4^n \cdot R_{n-1}^{k+1} - R_{n-1}^k}{4^n - 1} \tag{4}$$

龙贝格积分算法实现如下, 这里我们取 k 不超过 10 阶:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from math import *

def f(x):
    if x==0:
        return 0
    else :
        return sqrt(x)*log(x)

x = np.linspace(0,1,1000)
h=1
r=h/2*(f(1)+f(0))
k=np.arange(1,11)
H=np.zeros_like(k, dtype=float)
ans=np.zeros_like(k, dtype=float)
y0=np.zeros_like(k, dtype=float)
y=np.zeros_like(k, dtype=float)
y0[0]=r
H[0]=h

for i in range(1,len(k)):
    n = k[i]
    X = 0
    for j in range(1,2**(n-2)+1):
        X+=f((j-0.5)*h)
    y[0]=0.5*(y0[0]+X*h)
    for j in range(2, n+1):
        y[j-1]=y[j-2]+(y[j-2]-y0[j-1])/(pow(4,j-1)-1)
    h*=0.5
    for j in range(n):
        y0[j]=y[j]
    H[i]=h
    ans[i]=y0[n-1]
```

由此得到积分结果随 k 的变化图像为:

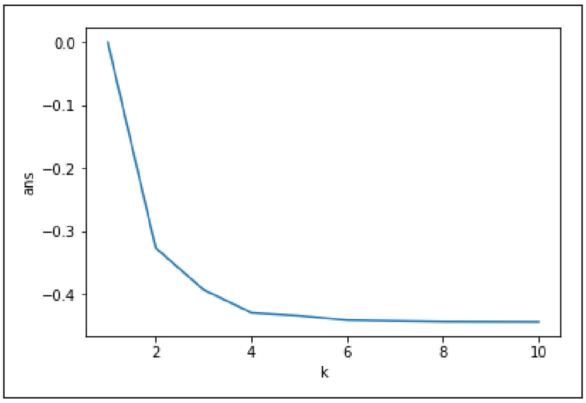

图 6: 积分值随 k 变化曲线

可见在 $k=6$ 阶以上时, 龙贝格方法已经可以得到比较精确的积分值了, 我们来观察一下误差与步长的变化情况, 如下图所示:

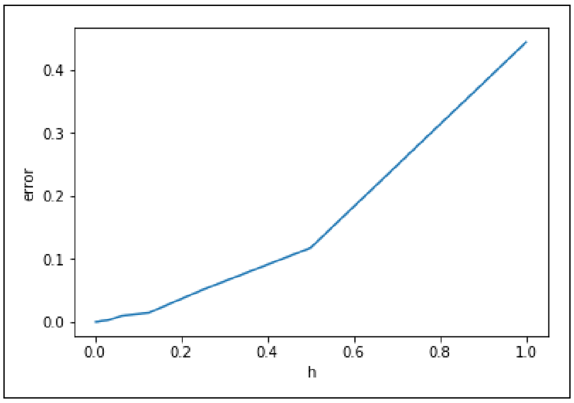

图 7: 误差随步长变化曲线