实验 #4
数值积分

刘扬 2011012162

实验目标

通过使用Simpson, Romberg, Gauss三种不同的求解方法对数值积分进行求解，掌握求解数值积分的基本方法。

实验原理

Simpson公式

复合的Simpson公式是根据复合梯形公式进行进一步细化，将原来的每个空间进行等分，
然后利用中点值进行对精度的进一步改造。对于一个函数f,我们要求它在$[a, b]$上的积分$\int_a^b f(x)dx$,
复合梯形公式是这样做的：我们将$[a, b]$分成n段 令$h = \frac{b-a}{n}$,然后有：

$$T(h) = \frac{h}{2}\left[\frac{1}{2}f(a) + \sum_{k=1}^{n-1} f(x_k) + \frac{1}{2}f(b)\right]$$

其余项为$O(h^2)$,二次项系数为1。
对于将$[a, b]$分成2n段，令$h = \frac{b-a}{n}$有：

$$T(\frac{h}{2}) = \frac{1}{4}h\left[f(a) + 2\sum_{k=1}^{n-1} f(x_k) + 2\sum_{k=0}^{n-1} f(x_{k+\frac{1}{2}})\right]$$

其精度为$O(h^2)$,其中二次项系数为$\frac{1}{4}$,且$x_{k+\frac{1}{2}} = x_k + \frac{1}{2}h, x_0 = a, x_n = b$

为了提高精度，我们进行错位相减法

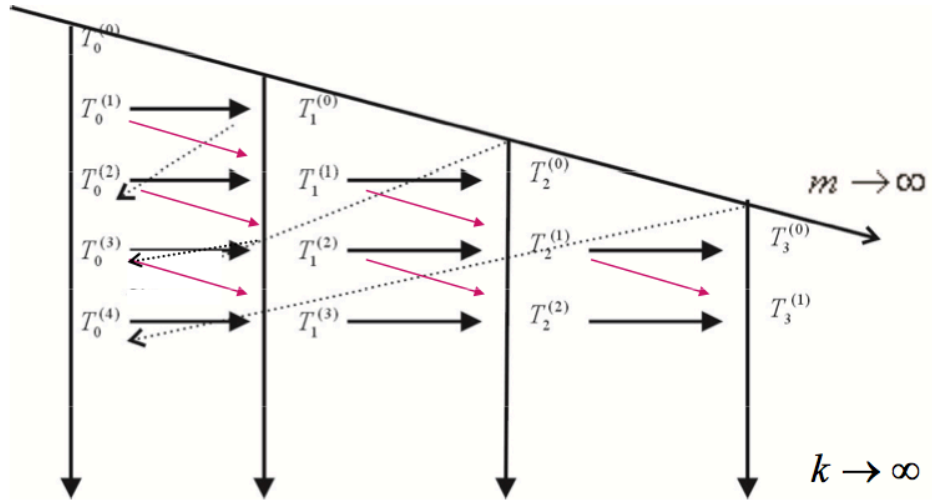
$$\begin{aligned} S(h) =& \frac{4T(\frac{h}{2}) - T(h)}{3} && (1) \\ =& \frac{h}{6}\left[f(a) + 4\sum_{k=0}^{n-1} f(x_{k+\frac{1}{2}}) + 2\sum_{k=1}^{n-1} f(x_k) + f(b)\right] && (2) \\ & && (3) \end{aligned}$$

这样余项为$O(h^4)$,通过使用Simpson公式可以提高计算的精度

Romberg外推

对Simpson公式进行进一步扩展，我们做出如下定义：
如果$h = \frac{(b-a)}{2^n}$,则$T_0^n = T(h)$
然后利用迭代公式进行计算：

$$T_n^k = \frac{4^n T_{n-1}^{k+1} - T_{n-1}^k}{4^{n-1}}$$



这样整个算法的流程可以描述如下:

计算T_0^0

对于某一个i，首先计算T_0^i

开始计算T_k^{i-k},直到T_i^0

如果$|T_i^0 - T_{i-1}^0|$满足要求，返回结果，否则i增1，继续到第二步

Gauss公式

直接利用题目中给出的公式进行计算

$$\int_a^b f(x)dx = \frac{h}{2}\left[f(x_{i+\frac{1}{2}} + \frac{h}{2\sqrt{3}}) + f(x_{i+\frac{1}{2}} - \frac{h}{2\sqrt{3}})\right] + \frac{(b-a)h^4}{4320} f^{(4)}(\xi), \xi \in (a,b)$$

算法设计与实现

具体的算法已经在上面给出，我们需要做的是针对上面的算法对h进行估算以保证精度

Simpson算法

Simpson公式的余项是$R_n(f) = -\frac{b-a}{180}(\frac{h}{2})^4 f^{(4)}(\xi), \xi \in (a,b)$ 对于$f(x) = \frac{1}{x}$,我们有$f^{(4)}(x) = \frac{24}{x^5}$,在$[1,2]$上，$f^{(4)}(x) \leq 24$ 所以我们有

$$|\frac{24h^4}{180 \times 16}| \leq \frac{1}{2} \times 10^{-8} h \leq 0.0278316$$

对于$f(x) = \frac{1}{x^2+1}$,有$f^{(4)}(x) = \frac{120x^4 - 240x^2 + 24}{(1+x^2)^5}$
在$[0,1]$上有$f^{(4)}(x) \leq 24$ 所以我们有

$$|\frac{24h^4}{180 \times 16}| \leq \frac{1}{2} \times 10^{-8} h \leq 0.0278316$$

所以都有$n \geq \frac{b-a}{h} = 36$

Gauss算法

由上面的推导，$f^{(4)}(x) \leq 24$，于是有:

$$|\frac{24h^4}{4320}| \leq \frac{1}{2} \times 10^{-8} h \leq 0.0308007$$

所以有$n \geq \frac{b-a}{h} = 33$ 按照上面的算法进行实现便可以求出解。

实验结果

实验结果为:

Listing 1: 实验结果

```
=============SIMPSON=============
For Problem 1, we have :  0.693147181722 ,answer is :  0.69314718056
delta is  1.16228071612e-09
For Problem 1, we have :  3.14159265359 ,answer is :  3.14159265359
delta is  2.84661183514e-13
=============ROMBERG=============
For Problem 1, we have :  0.69314718056 ,answer is :  0.69314718056
delta is  1.01030295241e-14
For Problem 1, we have :  3.14159265362 ,answer is :  3.14159265359
delta is  3.10014236504e-11
=============GAUSS=============
For Problem 1, we have :  0.693147179463 ,answer is :  0.69314718056
delta is  1.09727937847e-09
For Problem 1, we have :  3.14159265359 ,answer is :  3.14159265359
delta is  3.41060513165e-13
```

附:实验代码

Listing 2: simpson公式求解

```
#-*- coding:utf-8 -*-
import math
#for 1/x^2
def f_1(x):
     return 1.0/float(x)

def f_2(x):
     return 1.0/(float(x)*float(x)+1.0)

def cal(a, b, h, f):
     n = int((b - a) / h) + 1
     h = float((b-a))/float(n)
     #for the point a, b
     s = f(a)+f(b)
     now = a + h
     while abs(now - b)>0.0001:
          s += 2 * f(now) + 4 * f(now - 0.5 * h)
```

```
        now += h
    s += 4 * f(b - 0.5 * h)
    s = s * h / 6.0
    return s

if __name__=="__main__":
    h = 0.0278316
    f1 = cal(1, 2, h, f_1)
    f2 = 4.0 * cal(0, 1, h, f_2)
    y1 = math.log(2)
    y2 = math.pi
    print "============SIMPSON============"
    print "For Problem 1, we have : ", f1, ",answer is : ", y1
    print  "delta is ", abs(y1 - f1)
    print "For Problem 1, we have : ", f2, ",answer is : ", y2
    print "delta is ", abs(y2 - f2)
```

Listing 3: romberg外推求解

```
#-*- coding: utf-8-*-
import math

def f_1(x):
    return 1.0/float(x)
def f_2(x):
    return 1.0/(1.0+float(x)*float(x))

def cal(a, b, n, f):
    h = float(b - a) / float(n)
    s = f(a) + f(b)
    now = a + h
    while abs(b-now) > 0.0001:
        s += 2 * f(now)
        now += h
    s = s * h / 2.0
    return s

def romberg(a, b, n, e, f):
    t = []
    line = 0
    now_n = n
    while True:
        tmp_s = []
        tmp_s.append(cal(a, b, now_n, f))
        #romberg
        base = 1
        for i in range(line):
            base *= 4
            tmp_s.append((base * tmp_s[i] - t[line-1][i])/float(base-1))

        t.append(tmp_s)
        if line!=0:
            if abs(t[line][line] - t[line - 1][line - 1]) < e:
                break
```

```
        line += 1
        now_n *= 2

    return t[line][line]

if __name__=="__main__":
    f1 = romberg(1, 2, 10, 0.5 * 1e-8, f_1)
    f2 = 4.0 * romberg(0, 1, 10, 0.5 * 1e-8, f_2)
    y1 = math.log(2)
    y2 = math.pi
    t1 = abs(y1 - f1)
    t2 = abs(y2 - f2)
    print "=============ROMBERG============="
    print "For Problem 1, we have : ", f1, ",answer is : ", y1
    print "delta is ", t1
    print "For Problem 1, we have : ", f2, ",answer is : ", y2
    print "delta is ", t2
```

Listing 4: Gauss公式求解

```
#-*- coding:utf-8 -*-
import math
#for 1/x^2
def f_1(x):
    return 1.0/float(x)

def f_2(x):
    return 1.0/(float(x)*float(x)+1.0)

def gauss(a, b, e, f):
    n = int((b - a) / e) + 1
    h = float(b - a)/float(n)
    base = h/(2 * math.sqrt(3.0))
    now = a
    s = 0.0
    for i in range(n):
        s += (f(now + 0.5 * h + base) + f(now + 0.5 * h - base))
        now += h
    s = s * h /2.0
    return s
if __name__=="__main__":
    h = 0.0308007
    f1 = gauss(1, 2, h, f_1)
    y1 = math.log(2)
    f2 = 4.0 * gauss(0, 1, h, f_2)
    y2 = math.pi
    print "=============GAUSS============="
    print "For Problem 1, we have : ", f1, ",answer is : ", y1
    print  "delta is ", abs(y1 - f1)
    print "For Problem 1, we have : ", f2, ",answer is : ", y2
    print "delta is ", abs(y2 - f2)
```