计算物理第一次作业

梁旭民*

Cuiying Hornors College, Lanzhou University
liangxm15@lzu.edu.cn

Abstract

本次计算物理作业有两个问题：第一个问题是求解数值积分，让我们学会处理被积函数在上下限取值处发散的问题，并简单体会不同数值积分方法精确度的优劣；第二个问题是运用二分法、*Newton*法、弦切法求解一个有无穷个解的超越方程并比较不同数值求解方程根的方法的收敛性快慢

I. 计算定积分

A. 问题描述

利用学到的积分方法计算以下积分，并讨论各种方案的计算精度。根据自己的程序，选择符合自己要求的积分方法。

$$\int_0^1 t^{-\frac{2}{3}}(1-t)^{-\frac{1}{3}}dt = \frac{2\pi}{\sqrt{3}}$$

B. 问题分析

观察被积函数我们发现：当$t=0$或$t=1$时，该被积函数会发散，因此我们首先将积分分段

$$\int_0^1 t^{-\frac{2}{3}}(1-t)^{-\frac{1}{3}}dt = \int_0^{\frac{1}{2}} t^{-\frac{2}{3}}(1-t)^{-\frac{1}{3}}dt + \int_{\frac{1}{2}}^1 t^{-\frac{2}{3}}(1-t)^{-\frac{1}{3}}dt$$

为了避免积分在奇点处发散，我们分别做变量替换：

1. 第一部分积分

令$x=t^{\frac{1}{3}}$，有$t=x^3$及$dt=3x^2dx$，则第一部分积分变为

$$\begin{aligned}&\int_0^{\frac{1}{2}} t^{-\frac{2}{3}}(1-t)^{\frac{1}{3}}dt\\ =&\int_0^{(\frac{1}{2})^{\frac{1}{3}}} x^{-2}(1-x^3)^{-\frac{1}{3}}3x^2dx\\ =&\int_0^{(\frac{1}{2})^{\frac{1}{3}}} 3(1-x^3)^{-\frac{1}{3}}dx\end{aligned}$$

2. 第二部分积分

令$x=(1-t)^{\frac{2}{3}}$，则第二部分积分变为

$$\begin{aligned}&\int_{\frac{1}{2}}^1 t^{-\frac{2}{3}}(1-t)^{-\frac{1}{3}}dt\\ =&-\int_{\frac{1}{2}}^1 t^{-\frac{2}{3}}(1-t)^{-\frac{1}{3}}d(1-t)\\ =&-\int_0^{(\frac{1}{2})^{\frac{2}{3}}} (1-x^{\frac{3}{2}})^{-\frac{2}{3}}x^{-\frac{1}{2}}(-\frac{3}{2})x^{\frac{1}{2}}dx\\ =&\int_0^{(\frac{1}{2})^{\frac{2}{3}}} \frac{3}{2}(1-x^{\frac{3}{2}})^{-\frac{2}{3}}dx\end{aligned}$$

因此整个积分变成

$$\begin{aligned}\int_0^1 t^{-\frac{2}{3}}(1-t)^{-\frac{1}{3}}dt =& \int_0^{(\frac{1}{2})^{\frac{1}{3}}} 3(1-x^3)^{-\frac{1}{3}}dx\\ &+\int_0^{(\frac{1}{2})^{\frac{2}{3}}} \frac{3}{2}\left(1-x^{\frac{3}{2}}\right)^{-\frac{2}{3}}dx\end{aligned}$$

接下来，我们分别用矩形法、梯形法和Simpson法进行计算，并对相同步长下的计算精度及计算速度进行简单分析。

*指导老师：齐新老师

C. 数据分析

在不同的步长下得到矩形法、梯形法、Simpson法得到积分结果与标准值$\frac{2\pi}{\sqrt{3}}$之间的差距见Table 1. 可以发现在步长相同的情况下，Simpson方法计算的结果精度远大于矩形法和梯形法，这可能是用抛物线拟合曲线时符合较好。同时可以得到结论梯形法的精度比矩形法高。

Table 1: 不同步长下不同方法数值积分结果对比

步长	Rectangle		Trapezoid		Simpson	
N	I	$I-I_0$	I	$I-I_0$	I	$I-I_0$
100	3.621660	-0.005832	3.627536	0.000037	3.627599	0.000001
1000	3.627012	-0.000587	3.627599	0.000000	3.627599	0.000000
10000	3.627540	-0.000059	3.627599	0.000000	3.627599	0.000000
100000	3.627593	-0.000006	3.627599	0.000000	3.627599	0.000000
1000000	3.627598	-0.000001	3.627599	0.000000	3.627599	0.000000

II. 求解方程

A. 问题描述

求解非线性方程的根：

$$E_i - V(r) = 0$$

其中，$E_i = -0.40705$
V的表达式为

$$V_{et}(x) = -\frac{1}{x}e^{-\mu x}\cos(\mu x) \quad \mu = 0.1$$

B. 问题分析

用Python绘制函数图像$y(x) = -0.40705 + \frac{1}{x}e^{0.1x}\cos(0.1x)$，见Figure 1.

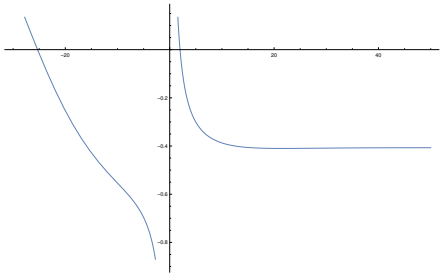

Figure 1: 区间$[-30, 50]$中的函数$f(x)$图像

该函数零点有无穷多个，其中一个在$x = 2$附近，因此我在分别用二分法，Newton法或弦切法讨论解的收敛性时，为了运算时间的缩小，选定x的取值区间[1.5,2.0]，为了比较方便我用Mathematica求解出方程的其中一个根$x_0 = 1.9767881587790763$并作为标准值。

C. 数据分析

我们选取1×10^{-11}作为可容忍误差，分别用二分法、Newton法和弦切法求解方程的[1.5,2.0]区间内的根及收敛补数与标准值的绝对误差见Table 2. 可以看出Newton法寻找方程解得收敛速度和弦切法差不多，而二分法要比这两种算法慢，同时由于Newton法需要求解函数的导数，因此相比而言弦切法这种算法较优。

Table 2: 方程数值解

方法	二分法	Newton法	弦切法
方程数值解	1.9767881587758893	1.9767881587790763	1.9767881587790763
绝对误差($\times 10^{-12}$)	5.889289056426605	9.0762952709155797	9.0762952709155797
收敛步数	28	5	5

Appendices

A. 求解上下限发散的数值积分

这是运用矩形法、梯形法、Simpson法求解上下限发散的数值积分Python代码。

Input Python source:

```
"""

积分上下限发散的积分

"""
import numpy as np
#积分上下限
a1=0;b1=(0.5)**(1.0/3)
a2=0;b2=(0.5)**(2.0/3)
#积分步数
N=[1000]
#积分标准值
I0=2*3.14159265358979323846264338327950288/np.sqrt(3)
#第一部分被积函数
def f1(x):
    return(3*pow((1-x*x*x),-1.0/3))
#第二部分被积函数
def f2(x):
    return(3.0/2*pow(1-pow(x,3.0/2),-2.0/3))
#矩形法积分
def rectangle(f,a,b):
    I=0;x=a;h=(b-a)/(int)(N[0])
    for i in range((int)(N[0])):
        I=I+f(x)*h
        x=x+h
    return(I)
#梯形法积分
def trapezoid(f,a,b):
    I=0;x=a;h=(b-a)/(int)(N[0])
    for i in range((int)(N[0])):
        I=I+1.0/2*(f(x)+f(x+h))*h
        x=x+h
    return(I)
#Simpson法积分
def simpson(f,a,b):
    h=(b-a)/(2*(int)(N[0]));s1=0.0;s2=0.0
    for i in range(1,(int)(N[0])+1):
        x=a+(2*i-1)*h
        s1=s1+f(x)
    for i in range(1,(int)(N[0])):
        x=a+2*i*h
        s2=s2+f(x)
    return((f(a)+4.0*s1+2.0*s2+f(b))*h/3.0)
#主函数
I1=rectangle(f1,a1,b1)+rectangle(f2,a2,b2)#矩形法求解
I2=trapezoid(f1,a1,b1)+trapezoid(f2,a2,b2)#梯形法求解
I3=simpson(f1,a1,b1)+simpson(f2,a2,b2)#Simpson法求解
#输出结果
print(I1,I1-I0,'\n')
print(I2,I2-I0,'\n')
print(I3,I3-I0,'\n')
```

B. 求解非线性方程

这是运用二分法、Newton法和弦切法求解非线性方程Python代码。

Input Python source:

```
"""

数值求解方程

"""
import numpy as np
#初始猜测位置
a=1.5;b=2.0
#可容忍误差
tolx=[0.00000000001]
#求解方程对应函数
def f(x):
    return(-0.40705+1.0/x*np.exp(-0.1*x)*np.cos(0.1*x))
#求解方程对应函数的导数
def dif(x):
    return(-((np.exp(-0.1*x)*np.cos(0.1*x))/(x*x))-(0.1*np.exp(-0.1*x)*np.sin(0.1*x))/
x-(0.1*np.exp(-0.1*x)*np.cos(0.1*x))/x)
#二分法
def dichotomy(f0,a,b):
    i=0#收敛步数初始化
    while(abs(b-a)>tolx[0]):
        i=i+1
        if(f0((a+b)/2.0)==0):
            a=(a+b)/2.0
        else:
            if(f0((a+b)/2.0)*f(a)<0):
                b=(a+b)/2.0
            if(f0((a+b)/2.0)*f(b)<0):
                a=(a+b)/2.0
    return(i,a,a-1.97678815877)
#Newton法
def newton(f0,dif0,a,b):
    i=0#收敛步数初始化
    while(abs(a-1.97678815877)>tolx[0]):
        i=i+1
        a=a-f0(a)/dif0(a)
    return(i,a,a-1.97678815877)
#弦切法
def stringcut(f0,a,b):
    i=0#收敛步数初始化
    c=b-f0(b)*((b-a)/(f0(b)-f0(a)))
    while(abs(c-a)>tolx[0]):
        i=i+1
        c=b-f0(b)*((b-a)/(f0(b)-f0(a)))
        a=b
        b=c
    return(i,c,c-1.97678815877)
#主函数
print(dichotomy(f,a,b))
print(newton(f,dif,a,b))
print(stringcut(f,a,b))
```

参考文献

[1] Philip J. Davis and Philip Rabinowitz, Methods of Numerical Integration.

[2] George E. Forsythe, Michael A. Malcolm, and Cleve B. Moler, Computer Methods for Mathematical Computations. Englewood Cliffs, NJ: Prentice-Hall, 1977. (See Chapter 5.)

[3] Press, W.H.; Teukolsky, S.A.; Vetterling, W.T.; Flannery, B.P. (2007), "Chapter 4. Integration of Functions", Numerical Recipes: The Art of Scientific Computing (3rd ed.), New York: Cambridge University Press, ISBN 978-0-521-88068-8.

[4] Josef Stoer and Roland Bulirsch, Introduction to Numerical Analysis. New York: Springer-Verlag, 1980. (See Chapter 3.)

[5] Boyer, C. B., A History of Mathematics, 2nd ed. rev. by Uta C. Merzbach, New York: Wiley, 1989 ISBN 0-471-09763-2 (1991 pbk ed. ISBN 0-471-54397-7).

[6] Eves, Howard, An Introduction to the History of Mathematics, Saunders, 1990, ISBN 0-03-029558-0.

[7] Wallis, John (1685). A Treatise of Algebra, both Historical and Practical. Shewing the Original, Progress, and Advancement thereof, from time to time, and by what Steps it hath attained to the Heighth at which it now is. Oxford: Richard Davis. doi:10.3931/e-rara-8842.

[8] Raphson, Joseph (1697). Analysis Æequationum Universalis seu ad Æequationes Algebraicas Resolvendas Methodus Generalis, & Expedita, Ex nova Infinitarum Serierum Methodo, Deducta ac Demonstrata (in Latin) (secunda ed.). London. doi:10.3931/e-rara-13516.

[9] Ryaben'kii, Victor S.; Tsynkov, Semyon V. (2006), A Theoretical Introduction to Numerical Analysis, CRC Press, p. 243, ISBN 9781584886075.

[10] Kaw, Autar; Kalu, Egwu (2008), Numerical Methods with Applications (1st ed.).

[11] Allen, Myron B.; Isaacson, Eli L. (1998). Numerical analysis for applied science. John Wiley & Sons. pp. 188–195. ISBN 978-0-471-55266-6.