**Numerical Methods and Statistics Using Python For Astrophysics and Cosmology Students**

A.1 Introduction

A.2 Numerical Differentiation and Integration

A.3 Runge-Kutta Methods for ODEs

A.4 Further Reading

**A.1 Introduction**

This section covers some basic material related to numerical computing and then we will provide some list of relevant material for further exploration. This section is only intended to provide a starting point with sample codes and is in no way an alternate to a full course on numerical computing.

**A.2 Numerical Differentiation and Integration**

In this section, we will cover some basic methods for differentiating and integrating functions of the form y=f(x).

To cover a detailed mathematical and computational background of the related topics please consult:

* **Numerical Methods** in Engineering with **Python (**Jaan **Kiusalaas)**
* **Numerical Methods** in Engineering with **Matlab (**Jaan **Kiusalaas)**
* **Advanced engineering mathematics** (**Erwin Kreyszig)**

**In order to get the first order derivative of the function y=f(x), we can use three basic techniques:**

1. **Forward difference method**

In order to calculate the derivative f’(x) of a function f(x) at some value ‘x’ using the forward difference method, we will first calculate the value of that function f(x+h) at a future interval ‘x+h’ where ‘h’ denotes the interval between ‘x’ and the next point ‘x+h’. We will then find the difference between f(x+h) and f(x), and then divide it with the interval to get the derivative.

The relation will now become:

Here, E(h) represents an error function of the order of ‘h’.

1. **Backward difference method**

In order to calculate the derivative f’(x) of a function f(x) at some value ‘x’ using the backward difference method, we will first calculate the value of that function f(x+h) at a previous interval ‘x-h’ where ‘h’ denotes the interval between ‘x’ and the previous point ‘x-h’. We will then find the difference between f(x) and f(x-h), and then divide it with the interval to get the derivative.

The relation will now become:

f'(x)= (f(x)-f(x-h))/h +E(h)

Here, E(h) represents an error function of the order of ‘h’.

1. **Central difference method**

In order to calculate the derivative f’(x) of a function f(x) at some value ‘x’ using the central difference method, we will first calculate the values of that function at points ‘x+h’ and ‘x-h’. We will then find the difference between f(x+h) and f(x-h), and then divide it with the interval to get the derivative.

The relation will now become:

Here, represents an error function of the order of, which shows that central difference method is more accurate than both central and forward methods.

**Here are some Python examples related to numerical differentiation:**

def fun(x):

y=x\*\*2+3\*x+1

return y

def central\_difference(x,h):

l=x+h

m=x-h

f=(fun(l)- fun(m))/(2\*h)

return f

def backward\_difference(x,h):

m=x-h

g=(fun(x)- fun(m))/(h)

return g

def forward\_difference(x,h):

l=x+h

i=(fun(l)- fun(x))/(h)

return i

print('central difference', central\_difference(1.1,0.2))

print('backward difference',backward\_difference(1.1,0.2))

print('forward difference', forward\_difference(1.1,0.2))

**Numerical Integration**

In order to numerically calculate the integral of a function y=f(x) over the interval [a,b]: **,** we basically break the continuous integral into discrete sums with some weights, W, in a form similar to:

Here, ‘i’ denotes the sample index.

Most common and simplest approaches are the Newton-Cotes class of methods known as Trapezoid Rules and Simpson’s Rules.

Trapezoid rule is the most simplest rule, it calculates the area under the curve between interval [a,b] as:

Simpson’s rule provides a much better approximation of the area under the curve as compared to the Trapezoidal rule, using the approximation:

Sample Python code to implement this is:

import sympy.mpmath as smp

from math import sin

def fx(x):

return x\*\*2

def fsin(x):

return sin(x)

def int\_simpsons(f,a,b):

I=0.

h=(b-a)/2.

I=(f(a)+4\*(f((a+b)/2.))+f(b))\*(h/3.)

return I

print(int\_simpsons(fx,1.,3.))

print(int\_simpsons(fsin,1.,3.))

-------------------------------------------------------------------------------------------

**Task 1 : Find the integral of:**

**A) x^3 –2x (1<= x <=4)**

**B) A) x^4 + 3x^2 (1<= x <=4)**

**Using Analytical method, Trapezoid Rule and Simpson’s Rule. Compare Trapezoid and Simpson’s Rule results with Analytical Solution.**

**Task 2: Implement Trapezoid Rule in Python**

A more advanced version of Simpson’s rule is known as the composite Simpson’s rule which approximates the integral as:

Which can also be approximated as:

Where h=(b-a)/n and n is the order of the Simpson’s rule.

Here is a sample code which implements the composite Simpson’s rule.

import numpy as np

Simpson\_N=100 #Basically N/2 of composite Simpson's Rule

#Composite Simpson's Rule

def ssin(f,a,b):

fv = np.vectorize(f)

result=0.

N=Simpson\_N # N/2 of composite Simpson's Rule

i=np.arange(0,N)\*1.

j=np.arange(1,N)\*1.

h=(b-a)/N

result=(h/6.)\*(fv(a)+ 4\*np.sum(fv(a + (i+0.5)\*h)) + 2\*np.sum(fv(a + j\*h)) + fv(b) )

return result

print(int\_simpsons(fx,1.,3.))

print(int\_simpsons(fsin,1.,3.))

For further reading, we suggest you to go through the references provided in the initial part of this section.

**A.3 Runge-Kutta Methods for ODEs**

Ordinary differential equations (ODEs), play an important role in space flight dynamics. We can adopt various approaches like Taylor series methods, Runge-Kutta methods and various other methods.

To cover a detailed mathematical and computational background of the related topics please consult:

* **Numerical Methods** in Engineering with **Python (**Jaan **Kiusalaas)**
* **Numerical Methods** in Engineering with **Matlab (**Jaan **Kiusalaas)**
* **Advanced engineering mathematics** (**Erwin Kreyszig)**

Here, we will limit our discussion to solve the initial value problems for the first order ODEs of the form:

y’=dy/dx=f(x,y) , with initial conditions at y0=y(x0)

Probably the most common approach in physical sciences is the fourth order Runge-Kutta method. The method requires us to perform a series of operations which are:

K0=hF(x,y)

K1=hF(x+ h/2, y + K0/2)

K2=hF(x+ h/2, y + K1/2)

K3=hF(x+ h, y + K2)

Y(x+h)=y(x) + (K0+2K1+2K2+K3)/6

Here is a Python example related to Runge-Kutta 4 for solving ODEs:

def RK4(f, a, b, ya, n):

'''

f:function

a:starting point

b:ending point

ya:initial value at x=a => y(a)=ya

n:numbers of samples to determine the size of 'h' or step size or width or delta\_x

'''

x = np.linspace(a, b, n)

h = x[1] - x[0]

ylst = [] #empty list

yi = ya

for xi in x:

ylst.append(yi) #adding elements to the list of ys

k1 = h\*f(xi, yi) #observe h is with k1 if you are writing h outside then modify k2,k3,k4 accordingly

k2 = h\*f(xi + 0.5\*h, yi + 0.5 \* k1)

k3 = h\*f(xi + 0.5\*h, yi + 0.5 \* k2)

k4 = h\*f(xi + h, yi + k3)

yi = yi+ (1./6) \* (k1 + 2.\*k2 + 2.\*k3 +k4) #setting new yi for new xi as initial conditions for next step

y = np.array(ylst)

return x, y

**A.4 Further Reading**

In order to learn advanced methods like Numerical PDEs and their applications in space flight dynamics, please consult:

A Compendium of Partial Differential Equation Models: Method of Lines Analysis with Matlab, Cambridge University Press (Schiesser, W. E. and G. W. Griffiths)

* Applied CFD Techniques: An Introduction Based on Finite Element Methods, John Wiley & Sons (R. L¨ohner)
* Computational Methods for Fluid Dynamics, Springer (J. H. Ferziger and M. Peric)
* Computational Space Flight Mechanics, Springer (Claus Weiland)
* Numerical Analysis and Optimization:An Introduction to Mathematical Modelling and Numerical Simulation (Craig and Allairie)
* Numerical methods in finite element analysis, Prentice-Hall (K.J. Bathe)
* Ordinary and Partial Differential Equation Routines in C, C++, Fortran, Java, Maple and Matlab, CRC Press (Lee, H. J. and W. E. Schiesser)
* Spectral Methods Evolution to Complex Geometries and Applications to Fluid Dynamics, Springer-Verlag (Canuto, Claudio S., M. Y. Hussaini, A. Quarteroni, and T. A. Zang)
* Theoretical and Computational Aerodynamics, Wiley (Tapan K. Sengupta)