Gaussian Quadrature

Arunima Sarkar

1 Singular integrals

The integral that have to be computed is:

$$J = \int_1^3 \frac{e^{-x}}{J_1(\sqrt{-x^2+4x-3})} dx \tag{1}$$

1.1 Graphing the integrand in python

The corresponding code is:

```
import matplotlib.pyplot as plt
import plotly.graph_objects as go

import scipy.special as sp
import math as m
import numpy as np
import seaborn as sns


def integrand(x):
    return np.exp(-x)/sp.jv(1,np.sqrt(-x**2 + 4*x - 3))

initial=1e-7
x=np.linspace(1+initial,3-initial,1000)
y=integrand(x)

fig = go.Figure()

fig.add_trace(go.Scatter(
    x=x,
    y=y
))

fig.update_layout(title="F(x) vs x",
    xaxis_title="Values in x",
```

```
    yaxis_title="f(x)",yaxis_type="log",
                    font=dict(
        family="Courier New, monospace",
        size=18,
        color="RebeccaPurple"
    ))
fig.show()
```

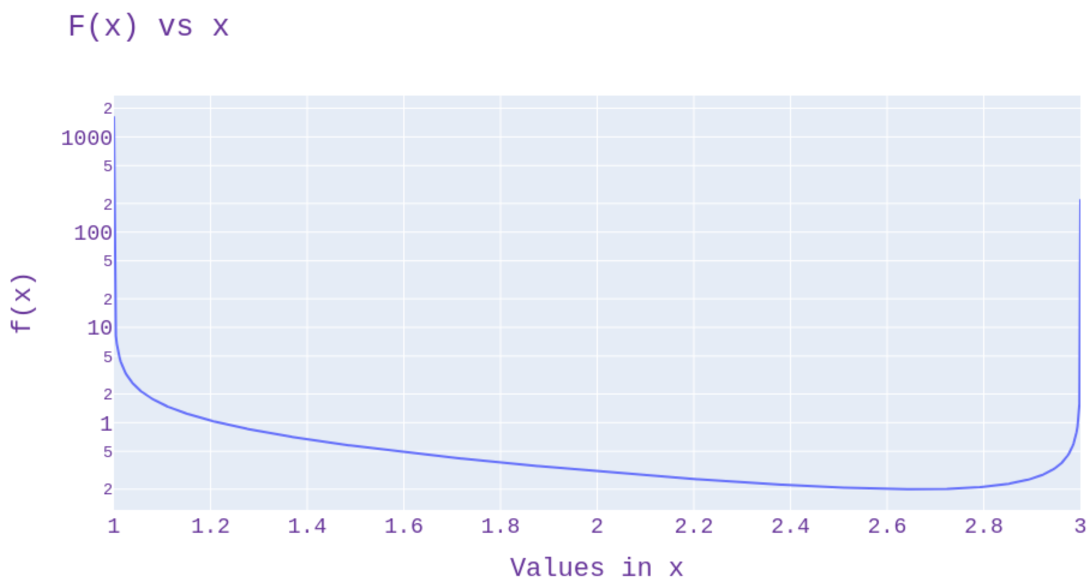

Figure 1

We can clearly see that there are two discontinuities at $x = 1$ and at $x = 3$.

1.2 Using quad

The code:

```
import scipy.integrate as si
ans=si.quad(integrand,1,3,full_output=0)
print(ans)
```

The value of the integral in eq(1) achieved is 1.140489938554265 and the error is $5.3339754835235545 \times 10^{-10}$. If we make the 4th argument any non-zero value then number of evaluation of the function is shown which in this case is 567.

1.3 Using Open Romberg

```
import sing_intg as rom
ans=rom.qromo(integrand,1,3,1e-4)
print(ans) # Requires 81 function calls and error is of order 6

import sing_intg as rom
ans=rom.qromo(integrand,1,3,1e-5)
print(ans) #Requires 81 function calls and error is of order 6

import sing_intg as rom
ans=rom.qromo(integrand,1,3,1e-6)
print(ans) #Requires 2187 function calls and error is of order 6

import sing_intg as rom
ans=rom.qromo(integrand,1,3,1e-7)
print(ans) #Requires 531441 function calls and error is of order 8
```

The number of function calls drastically increases with decreasing epsilon values in qromb.

1.4 Transforming the integrand

Here we are applying a transformation where x=t+2 on eq(1). The equation is:

$$J = \int_{-1}^{1} \frac{e^{-t-2}}{J_1(\sqrt{1-t^2})} dt = \int_{-1}^{0} \frac{e^{-t-2}}{J_1(\sqrt{1-t^2})} dt + \int_{0}^{1} \frac{e^{-t-2}}{J_1(\sqrt{1-t^2})} dt \tag{2}$$

According to Numerical recipes in C, the mentioned transformations on improper integrals are done,I also broke the integral from -1 to 0 and 0 to 1 and carried out the following transformation,

when singularity is in a,

$$\int_a^b f(x) = \frac{1}{1-\gamma} \int_0^{(b-a)^{(1-\gamma)}} t^{\frac{\gamma}{1-\gamma}} f(t^{\frac{1}{1-\gamma}} + a) dt, \qquad b > a \tag{3}$$

So our equation(2),

$$J = \int_{-1}^{0} \frac{e^{-t-2}}{J_1(\sqrt{1-t^2})} dt \tag{4}$$

upon transformation looks like,

$$\int_0^1 2mf(m^2-1)dm \tag{5}$$

when singularity is in b, the transformation is,

$$\int_a^b f(x) = \frac{1}{1-\gamma} \int_0^{(b-a)^{(1-\gamma)}} t^{\frac{\gamma}{1-\gamma}} f(b - t^{\frac{1}{1-\gamma}}) dt, \qquad b > a \tag{6}$$

So our equation(2),

$$J = \int_{0}^{1} \frac{e^{-t-2}}{J_1(\sqrt{1-t^2})} dt \tag{7}$$

upon transformation looks like,

$$\int_0^1 2mf(1-m^2)dm \tag{8}$$

Code

```
def ne_integrand_1(m):
    t=m**2-1
    return 2*(m*np.exp(-t-2)/sp.jv(1,np.sqrt(1-(t)**2)))

def ne_integrand_2(m):
    t=1-m**2
    return 2*(m*np.exp(-t-2)/sp.jv(1,np.sqrt(1-(t)**2)))
```

```
ans11=rom.qromo(ne_integrand_1,0,1,1e-5)

ans22=rom.qromo(ne_integrand_2,0,1,1e-5)

Integral=ans11[0]+ans22[0]
error=ans11[1]+ans22[1]
print(Integral)
print("Error is",error)
```

And upon applying romberg upon this we get a lower order error, $1.688269168388129e \times 10^{-11}$ and even the number of function calls are also less, which is 81.

2 Gaussian Quadratures

We have to just compute x_j and w_j, which are given by below formulas,

$$x_j = \cos\frac{\pi(j-\frac{1}{2})}{N} \tag{9}$$

$$w_j = \frac{\pi}{N} \tag{10}$$

Here N=20 is considered as exact value ans the code has looped over till it reaches 20 and error is found by checking their values.

Code:

```
def gq(x,w):
    s=0
    for i in range(len(x)):
        s+=new_integrand(x[i])*w[i]
    return s

def exact():
    x=[]
    w=[1]*20
    val=0
    for i in range(1,21):
        x.append(np.cos(np.pi*(i-0.5)/20))
        w[i-1]=w[i-1]*m.pi/20
    return gq(x,w)

errors=[]
for i in range(1,21):
```

```
xi=np.array(range(1,i+1))
xi=np.cos(np.pi*(xi-0.5)/i)
wi=np.full(i,np.pi/i)
errors.append(np.abs(gq(xi,wi)-exact()))
```

The graph for Number of points(N) Vs Order of error.

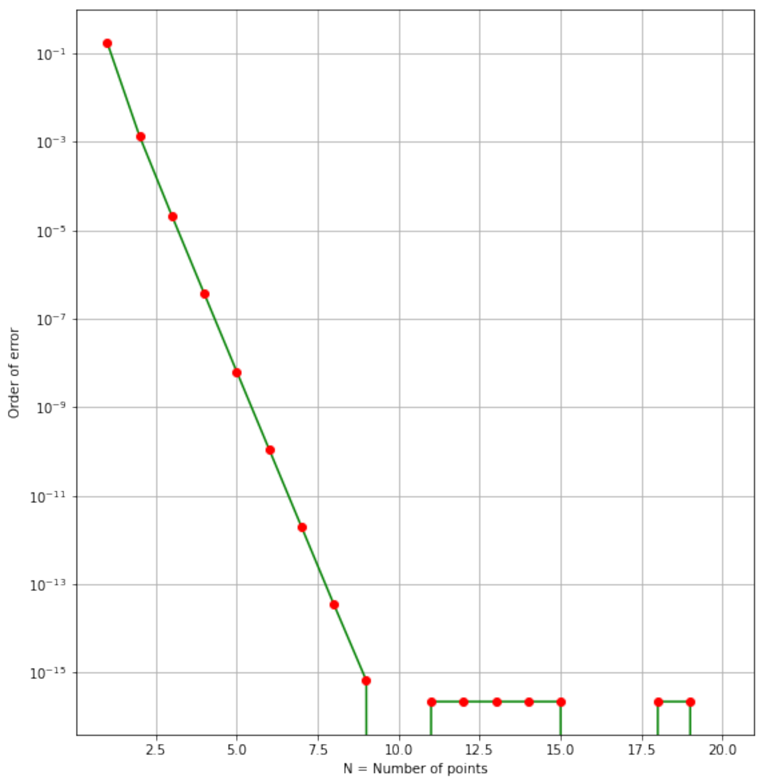

Figure 2

3 Using quad on integrals from Romberg assignment

3.1 Question 1, 2

Code

```
def f1(x):
    return ((sp.jv(3,2.7*x))**2)*x

def f2(x):
    return ((sp.kv(3,1.2*x))**2)*x

import scipy.integrate as si
ans1=si.quad(f1,0,1,full_output=0,epsabs=1e-12,epsrel=1e-12)
print(ans1) # Using quad function


>(0.009969186534269647, 1.106802042712923e-16)


ans2=si.quad(f2,1,np.inf,full_output=0,epsabs=1e-12,epsrel=1e-12)
print(ans2)

>(3.0924507786178372, 8.110039530734295e-13)
```

When we are breaking the functions and then taking the integral gives us a better answer compared to taking the whole integrand in one go. This behaviour is similar to the one seen in trapzd function for trapezoidal method.

3.2 Question 3

Code: Using Gauss-Legendre in f1

```
import gauss_quad as gq
x,w=gq.gauleg(0,1,20)
val=0
for i in range(len(x)):
    val+=f1(x[i])*w[i]
print(np.abs(val-ans1[0]))

>1.682681771697503e-16
```

Code: Using Gauss-Laguerre in f2

```
def func2(z):
    return f2(z+1)*m.exp(z) #shifting it by 1 such that the integral is from
                                zero to infinity and not from 1 to infinity

x,w=gq.gaulag(105,0) #Quad is giving 13th order accuracy for 105 eval, but
                       Gauss-Laguerre is giving 12th order.
val=0
for i in range(len(x)):
    val+=func2(x[i])*w[i]
print(val)
print(np.abs(val-ans2[0])) #alpha is zero


3.09245077861445
3.3870684035264276e-12
```

The alpha is zero here, however, according to wiki, this form of integration is analytically correct but numerically stable.

3.3 Question 4: Using Romberg to evaluate f1

f1 represents the function

$$I_1 = \int_0^1 J_v^2 (ku) \, udu \tag{11}$$

Code:

```
import romberg as r
integral=r.qromb(f1,0,1,1.e-12)
print(integral)

>(0.009969186534269642, -1.0555978304531426e-16, 129)
```

We can see that the error is of order 16.

3.4 Question 5: Using Romberg to evaluate f2

```
def modified_f2(x):
    A=20
    return (sp.kv(3,1.2*A*np.tan(x))**2)*(A**2)*np.tan(x)/(np.cos(x)**2)

#Visalizing the function after modification but converting Nan values to zero
x=np.array(range(1,20))
x=modified_f2(x)
for i in range(len(x)):
    if np.isnan(x[i]):
        x[i]=0
fig = plt.figure(figsize=(20,10))
fig.add_subplot(1, 2, 1)
plt.semilogy(range(1,20),x,"c")
plt.semilogy(range(1,20),x,"mo")
plt.grid()
```

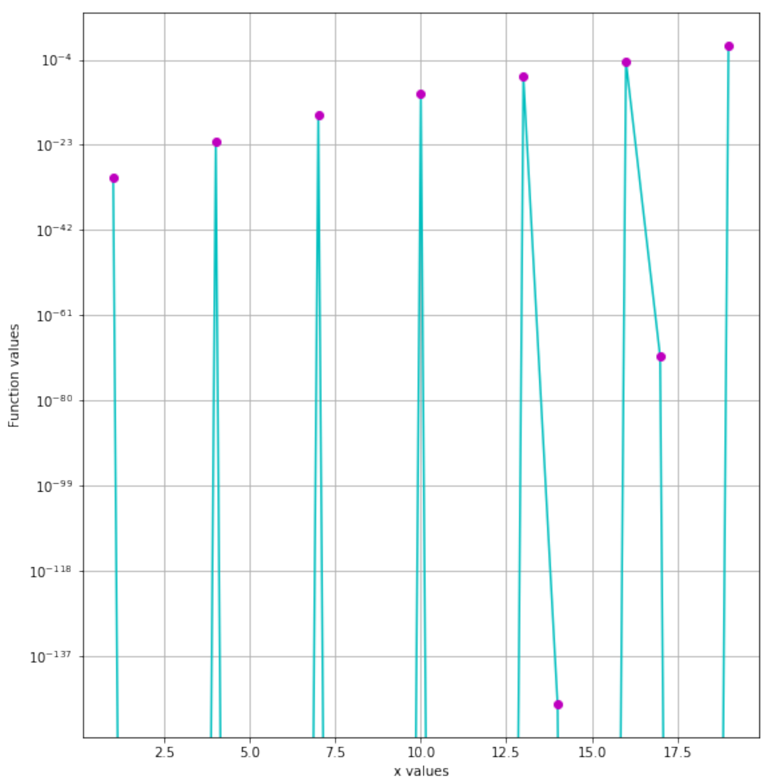

Figure 3

Code:

```
import romberg as r
integral=r.qromb(modified_f2 ,m.pi/4,m.pi/2,1.e-12) #Closed Romberg
print(integral)

>(1.1029478499177205e-21, -7.071826959966638e-36, 2049)


integral2=rom.qromo(modified_f2 ,m.pi/4,m.pi/2,1.e-12) #Open Rpmberg
print(integral2)

 >(1.1029478499176691e-21, 4.799818695639193e-34, 2187)
```

It can be observed that changing the values of A changes the order of accuracy.