Report

Homework nr.4

Student: Macrii Danu
Gr.: FAF-222
Teacher: Bostan Viorel

Chișinău, Spring 2023

Problem 1

My code:

```
def f(t, i):
    return t**(i-1) * exp(-t)


def composite_simpson(f, a, b, n, i):
    h = (b-a)/n
    integral = f(a, i)
    for j in range(1, n):
        if j % 2 == 1:
            integral += 4*f(a+j*h, i)
        else:
            integral += 2*f(a+j*h, i)

    integral += f(b, i)
    integral *= h/3
    return integral
```

Explaining:

Here I found that standard dart error of 10^{-6} can be achieved with Simpson's rule with n = 200 and an interval from 0 to 10. This error is achieved with inbuilt quadrature approximation with an interval from 0 to 45 , but saw the result that for bigger x quadrature becomes less and less efficient from 10^{-15} to 10^{-4}.

Results

x	Composite Simpson rule	Quad Aproximation	Gamma Function
1	1.000004272559753	1.0	1.0
2	0.9999872105072077	1.0	1.0
3	2.000025438381185	1.9999999999999998	2.0
4	5.999975121746226	5.9999999999999964	6.0
5	23.999998333308667	23.999999999999865	24.0
6	120.0000015986351	119.99999999999407	120.0
7	720.0000002013865	719.9999999997267	720.0
8	5039.999999812196	5039.999999987391	5040.0
9	40319.99999995062	40319.99999941779	40320.0
10	362879.99999957875	362879.99997309997	362880.0

Problem 2

My code:

```
def f(x):
    return 4/(1+x**2)

def trapezoidal(f, a, b, n):
    h = (b-a)/n
    integral = f(a)
    for j in range(1, n):
        integral += 2*f(a+j*h)
    integral += f(b)
    integral *= h/2
    return integral


def midpoint(f, a, b, n):
    h = (b-a)/n
    integral = 0
    for j in range(n+1):
        integral += h*f(a+j*h)
    return integral
```

Explaining:

After processing our result i found that all methods will , eventually , find the accurate pi, it only a matter of computation. Simpsons rule will achieve error or 0 when n=4096, trapezoidal rule for n= 4194304 and for midpoint some n further than other methods. As we see further n will get the error bigger , it is caused by Runge's phenomenon (at the ends of interval error function becomes bigger and bigger).

Result

Composite Trapezoical Rule				
Roots	I(n)	I(n)-I(n-1)	Ratio	Absolute Error
65536	3.1415926535509735	-	-	3.8819614189833374e-11
131072	3.14159265358012	2.914646302087931e-11	-	9.673151168954064e-12
262144	3.1415926535873995	7.279510327862226e-12	4.003904343582235	2.3936408410918375e-12
524288	3.1415926535891696	1.7701395904623496e-12	4.112393376818866	6.235012506294879e-13
1048576	3.141592653589666	4.9649173661237e-13	3.565295169946333	1.270095140171179e-13
2097152	3.1415926535897074	4.1300296516055823e-14	12.021505376344086	8.570921750106208e-14
4194304	3.141592653589793	8.570921750106208e-14	0.48186528497409326	0.0
8388608	3.1415926535896417	-1.5143442055887135e-13	-0.5659824046920822	1.5143442055887135e-13

Composite Midpoint Rule

Roots	I(n)	I(n)-I(n-1)	Ratio	Absolute Error
65536	3.141638429918161	-	-	-4.577632836788581e-05
131072	3.1416155417637133	-2.288815444773107e-05	-	-2.2888173920154742e-05
262144	3.1416040976791964	-1.1444084516920583e-05	1.9999987254454408	-1.1444089403234159e-05
524288	3.1415983756350676	-5.722044128741999e-06	1.9999993462889607	-5.72204527449216e-06
1048576	3.1415955146126153	-2.861022452282924e-06	1.9999997288298632	-2.861022822209236e-06
2097152	3.141594084101182	-1.4305114333090785e-06	1.9999997103586709	-1.4305113889001575e-06
4194304	3.1415933688455304	-7.1525565159547e-07	2.000000181918365	-7.152557373046875e-07
8388608	3.1415930112175103	-3.576280200867643e-07	1.9999989134574563	-3.576277172179232e-07

Composite Simpson's Rule

Roots	I(n)	I(n)-I(n-1)	Ratio	Absolute Error
16	3.1415926512248227	-	-	2.3649704417039175e-09
32	3.141592653552836	2.3280133376601952e-09	-	3.695710404372221e-11
64	3.141592653589215	3.637889989249743e-11	63.99350570082277	5.782041512247815e-13
128	3.1415926535897833	5.684341886080801e-13	63.9984375	9.769962616701378e-15
256	3.1415926535897936	1.021405182655144e-14	55.65217391304348	-4.440892098500626e-16
512	3.1415926535897936	0.0	-	-4.440892098500626e-16
1024	3.141592653589792	-1.7763568394002505e-15	-0.0	1.3322676295501878e-15
2048	3.1415926535897913	-4.440892098500626e-16	4.0	1.7763568394002505e-15
4096	3.141592653589793	1.7763568394002505e-15	-0.25	0.0
8192	3.1415926535897887	-4.440892098500626e-15	-0.4	4.440892098500626e-15

Problem 3

My code:

```
def cf(x):
    return cos((pi*x**2)/2)
def sf(x):
    return sin((pi*x**2)/2)


x_my = linspace(0, 10, 300)
for i in x_my:
    y_myc.append(simpson(cf, 0, i, 150))
    y_mys.append(simpson(sf, 0, i, 150))
    y_quads.append(integrate.quad(lambda x:sf(x), 0, i)[0])
    y_quadc.append(integrate.quad(lambda x:cf(x), 0, i)[0])

x = linspace(0, 10, 1000)
s, c = fresnel(x)
```

Explaining:

Here found that the most efficient d , yet accurate e and pleasing to the eye is if we use 300 points to plot the graph, also used 150 points to find the integral using Simpson rule to plot the graph quite the same as the original function. We can see from the graph below that error is not significant.

Result

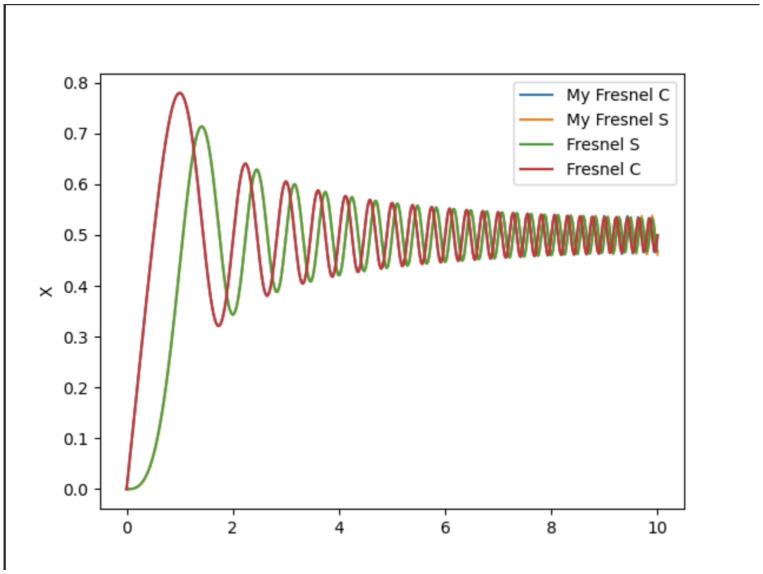

Figure 1: Original Fresnel graph over Fresnel generated with Simpson rule

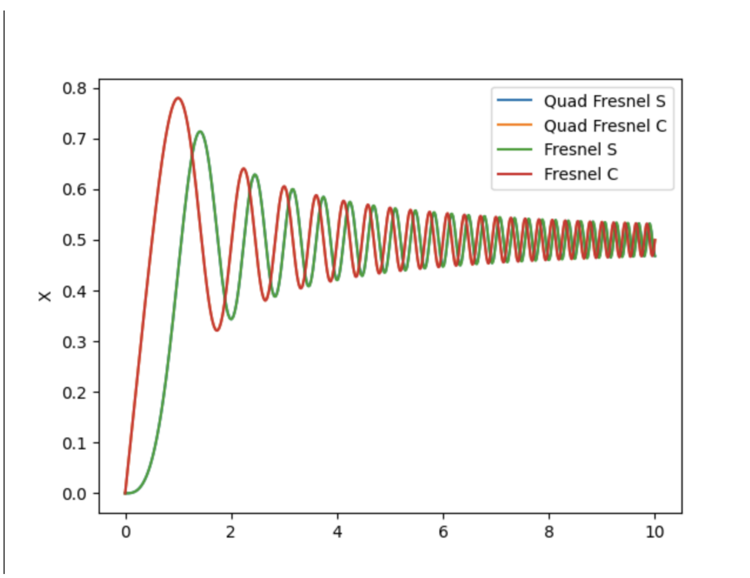

Figure 2: Original Fresnel graph over Quadrature Approximation Fresnel

Problem 4

My code:

```
a = 1
b = 5

def f(x):
    return 1/((x-1)**(5/2))

def t(t):
    return (a+b+t*(b-a))/2

arr1 = []
tegration = 0
for i in range(2, 7):
    x, w = roots_legendre(2**i)
    for j in range(2**i):
        tegration += w[j]*f(t(x[j]))
    tegration *= (b-a)/2
    arr1.append(tegration)


arr2 = []
arr3 = ['-']
arr4 = ['-', '-']
for i in range(2, 7):
    arr2.append(simpson(f, 1, 5, 2**i))
for i in range(2, 7):
    arr2.append(simpson(f, 1, 5, 2**i))
for i in range(1, len(arr2)):
    arr3.append(arr2[i]-arr2[i-1])
for i in range(2, len(arr3)):
    arr4.append(arr3[i-1]/arr3[i])
```

Explaining:

In this exercise we get an divergent integral because the power of the denominator is greater than 1. So the result are shown bellow. As we see because this integral isn't smooth enough on the interval Simpson rule ratio isn't 16 . Also this function isn't derivable enough C^4. In the code i changed so Simpsons method will not calculate $f(1)$, it will be automatically 0 because ratio will not change.Also we see that Simpson rule slower diverges than Gaussian quadrature.

Result

Roots	Gaussian quadrature DIY
4	17.903621822787922
8	154.9950608167236
16	1178.2764604116824
32	8984.108421298522
64	69758.24817111249

Composite Simpsons Rule			
Roots	I(n)	I(n)-I(n/2)	Ratio
4	1.5471345034110358	-	-
8	4.528563236665184	2.981428733254148	-
16	12.961097351569197	8.432534114904014	0.35356260557364905
32	36.81188937773342	23.850792026164225	0.3535536306573617
64	104.27211542802378	67.46022605029036	0.35355339616537745

```
InBuild Quad function result:  -0.08333333143865021
```

Appendix

1

```
from scipy.special import gamma
import scipy.integrate as integrate
from numpy import exp
from prettytable import PrettyTable
x = PrettyTable()



def f(t, i):
    return t**(i-1) * exp(-t)


def composite_simpson(f, a, b, n, i):
    h = (b-a)/n
    integral = f(a, i)
    for j in range(1, n):
        if j % 2 == 1:
            integral += 4*f(a+j*h, i)
        else:
            integral += 2*f(a+j*h, i)

    integral += f(b, i)
    integral *= h/3
    return integral


arr = []
for i in range(1, 11):
    arr.append(integrate.quad(lambda x: f(x, i), 0, 45)[0])

x.add_column('x', list(range(1, 11)))
x.add_column('Composite Simpson rule', [composite_simpson(f, 0, 50, 300, i) for i in range(1, 11
x.add_column('Quad Aproximation', arr)
x.add_column('Gamma Function', [gamma(i) for i in range(1, 11)])
print(x)
```

2

```
from numpy import pi
import scipy.integrate as integrate
import time
from prettytable import PrettyTable
table1 = PrettyTable()


def f(x):
    return 4/(1+x**2)


def simpson(f, a, b, n):
    h = (b-a)/n
    integral = f(a)
    for j in range(1, n):
        if j % 2 == 1:
            integral += 4*f(a+j*h)
        else:
            integral += 2*f(a+j*h)

    integral += f(b)
    integral *= h/3
    return integral
def trapezoidal(f, a, b, n):
    h = (b-a)/n
    integral = f(a)
    for j in range(1, n):
        integral += 2*f(a+j*h)
    integral += f(b)
    integral *= h/2
    return integral


def midpoint(f, a, b, n):
    h = (b-a)/n
    integral = 0
    for j in range(n+1):
        integral += h*f(a+j*h)
    return integral

arr2 = []
arr3 = ['-']
arr4 = ['-', '-']
for i in range(4, 14):
    arr2.append(simpson(f, 0, 1, 2**i))
for i in range(1, len(arr2)):
    arr3.append(arr2[i]-arr2[i-1])
```

```
for i in range(2, len(arr3)):
    if arr3[i] != 0:
        arr4.append(arr3[i-1]/arr3[i])
    else:
        arr4.append('-')
table1.add_column('Roots', [2**i for i in range(4, 14)])
table1.add_column('I(n)', arr2)
table1.add_column('I(n)-I(n-1)', arr3)
table1.add_column('Ratio', arr4)
table1.add_column('Absolute Error', [pi-i for i in arr2])
print(table1.get_string(title="Composite Simpson's Rule"))
table1.clear()

arr2 = []
arr3 = ['-']
arr4 = ['-', '-']
for i in range(16, 24):
    arr2.append(midpoint(f, 0, 1, 2**i))
for i in range(1, len(arr2)):
    arr3.append(arr2[i]-arr2[i-1])
for i in range(2, len(arr3)):
    arr4.append(arr3[i-1]/arr3[i])
table1.add_column('Roots', [2**i for i in range(16, 24)])
table1.add_column('I(n)', arr2)
table1.add_column('I(n)-I(n-1)', arr3)
table1.add_column('Ratio', arr4)
table1.add_column('Absolute Error', [pi-i for i in arr2])
print(table1.get_string(title="Composite Midpoint Rule"))


start = time.time()
simpson(f, 0, 1, 500)
final = time.time()
print('time for simpson method :', final-start, '\n')

start = time.time()
print('error for quad aproximation :', integrate.quad(lambda x: f(x), 0, 1, epsrel=1.49e-1)[0]-p
final = time.time()
print('time for quad :', final-start)
```

3

```
from numpy import pi, cos, sin, linspace
import scipy.integrate as integrate
from scipy.special import fresnel
import matplotlib.pyplot as plt


def cf(x):
    return cos((pi*x**2)/2)


def sf(x):
    return sin((pi*x**2)/2)


def simpson(f, a, b, n):
    h = (a+b)/n
    integral = f(a)
    for j in range(1, n):
        if j % 2 == 1:
            integral += 4*f(a+j*h)
        else:
            integral += 2*f(a+j*h)

    integral += f(b)
    integral *= h/3
    return integral


y_myc = []
y_mys = []
y_quads = []
y_quadc = []
x_my = linspace(0, 10, 300)
for i in x_my:
    y_myc.append(simpson(cf, 0, i, 150))
    y_mys.append(simpson(sf, 0, i, 150))
    y_quads.append(integrate.quad(lambda x:sf(x), 0, i)[0])
    y_quadc.append(integrate.quad(lambda x:cf(x), 0, i)[0])

x = linspace(0, 10, 1000)
s, c = fresnel(x)


plt.plot(x_my, y_myc, label='My Fresnel C')
plt.plot(x_my, y_mys, label='My Fresnel S')
# plt.plot(x_my, y_quads, label='Quad Fresnel S')
# plt.plot(x_my, y_quadc, label='Quad Fresnel C')
```

```
plt.plot(x, s, label='Fresnel S')
plt.plot(x, c, label='Fresnel C')

plt.ylabel('Y')
plt.ylabel('X')
plt.legend()
plt.show()
```

4

```
from scipy.special import roots_legendre
import scipy.integrate as integrate
from prettytable import PrettyTable
table1 = PrettyTable()
table2 = PrettyTable()
a = 1
b = 5


def f(x):
    return 1/((x-1)**(5/2))


def t(t):
    return (a+b+t*(b-a))/2


def simpson(f, a, b, n):
    h = (b-a)/n
    integral = 0
    for j in range(1, n):
        if j % 2 == 1:
            integral += 4*f(a+j*h)
        else:
            integral += 2*f(a+j*h)

    integral += f(b)
    integral *= h/3
    return integral

arr1 = []
tegration = 0
for i in range(2, 7):
    x, w = roots_legendre(2**i)
    for j in range(2**i):
        tegration += w[j]*f(t(x[j]))
    tegration *= (b-a)/2
    arr1.append(tegration)
table2.add_column('Roots', [2**i for i in range(2, 7)])
table2.add_column('Gaussian quadrature DIY',arr1)


arr2 = []
arr3 = ['-']
arr4 = ['-', '-']
for i in range(2, 7):
    arr2.append(simpson(f, 1, 5, 2**i))
```

```
for i in range(1, len(arr2)):
    arr3.append(arr2[i]-arr2[i-1])
for i in range(2, len(arr3)):
    arr4.append(arr3[i-1]/arr3[i])
table1.add_column('Roots', [2**i for i in range(2, 7)])
table1.add_column('I(n)', arr2)
table1.add_column('I(n)-I(n/2)', arr3)
table1.add_column('Ratio', arr4)

print(table2)
print(table1.get_string(title="Composite Simpson's Rule"))

print('Build in function result: ', integrate.quad(f, a, b)[0])
```