Computational Physics Homework 7

Cristopher Cerda Puga

Problem 1.

Apply Gauss-Legendre and Gauss-Laguerre quadrature to two examples of your choice.

- Euler-Lagrange for $f(x) = e^{-x^2}$:

```python
import numpy as np
import mpmath as mp
from mpmath import *
from sympy import *

mp.dps = 40; mp.pretty = True

# define the legendre weights and roots

def legendre_weights_roots(n,dgt):
    x = Symbol('x')
    roots = Poly(legendre(n,x),x).all_roots()
    x_i = [rt.evalf(dgt) for rt in roots]
    w_i = [(2*(1-rt**2)/(n+1)**2/(-rt*legendre(n, rt)+ legendre(n+1,rt))**2).evalf(dgt) for rt in roots]
    return x_i, w_i

def f(x): #function to integrate
    x2 = mp.fmul(-x,x)
    return mp.exp(x2)


n=50 #number of points
dgt=50 #number of digits
gl=legendre_weights_roots(n,dgt) #legendre weights and roots

# calculates the integral numerically

integral = 0
for i in range(n):
    integral += f(gl[0][i])*gl[1][i] # change the the function you want to integrate

print(integral)
>>> 1.4936482656248540507989348722637060107 09 # OK
```

- Euler-Lagrange for $f(x) = \dfrac{1}{1+e^x}$:

```python
import numpy as np
import mpmath as mp
from mpmath import *
from sympy import *

mp.dps = 40; mp.pretty = True

# define the legendre weights and roots

def legendre_weights_roots(n,dgt):
    x = Symbol('x')
    roots = Poly(legendre(n,x),x).all_roots()
    x_i = [rt.evalf(dgt) for rt in roots]
    w_i = [(2*(1-rt**2)/(n+1)**2/(-rt*legendre(n, rt)+ legendre(n+1,rt))**2).evalf(dgt) for rt in roots]
```

```
    return x_i, w_i

def f(x): #function to integrate
    return mp.fdiv(1,1 + mp.exp(x))


n=50 #number of points
dgt=50 #number of digits
gl=legendre_weights_roots(n,dgt) #legendre weights and roots

# calculates the integral numerically

integral = 0
for i in range(n):
    integral += f(gl[0][i])*gl[1][i] # change the the function you want to integrate

print(integral)
>>> 1.0 # OK
```

- Euler-Laguerre for $f(x) = e^{-x} \ln x$:

```
import numpy as np
import mpmath as mp
from mpmath import *
from sympy import *

mp.dps = 100; mp.pretty = True

def laguerre_weigths_roots(n,dgt):
    x = Symbol('x')
    roots = Poly(laguerre(n,x),x).all_roots()
    x_i = [rt.evalf(dgt) for rt in roots]
    w_i = [(rt/((n+1)*laguerre(n+1,rt))**2).evalf(dgt) for rt in roots]
    return x_i, w_i

def f(x):
    return mp.fmul(mp.exp(-x),mp.ln(x))
def g(x):
    return mp.fdiv(mp.sin(x),x)

n= 61 # number of points
dgt = 10 # digits
gl = laguerre_weigths_roots(n,dgt)
ngl = np.array(gl,dtype = object)

integral = 0
for i in range(n):
    integral += f(ngl[0][i])*ngl[1][i]*mp.exp(ngl[0][i])

print(integral)
>>> -0.566882466 # OK
```

- Euler-Laguerre for $f(x) = x - \ln(e^x - x)$:

```
import numpy as np
import mpmath as mp
from mpmath import *
from sympy import *
```

```python

mp.dps = 100; mp.pretty = True

def laguerre_weigths_roots(n,dgt):
    x = Symbol('x')
    roots = Poly(laguerre(n,x),x).all_roots()
    x_i = [rt.evalf(dgt) for rt in roots]
    w_i = [(rt/((n+1)*laguerre(n+1,rt))**2).evalf(dgt) for rt in roots]
    return x_i , w_i

def f(x):
    return mp.fmul(mp.exp(-x),mp.ln(x))
def g(x):
    return x-mp.ln(mp.exp(x)-x)

n= 61 # number of points
dgt = 10 # digits
gl = laguerre_weigths_roots(n,dgt)
ngl = np.array(gl,dtype = object)

integral = 0
for i in range(n):
    integral += g(ngl[0][i])*ngl[1][i]*mp.exp(ngl[0][i])

print(integral)
>>> 1.15769475279977456 # OK
```

Problem 2.

Use the Montecarlo method (hit or miss) to estimate the volume of the intersection of two hyperspheres of unit radius separated by a distance L=1/2 in a space of dimension d (make sure that $for d = 2,3$ you get something in agreement with the exact result). Apart from $d = 2,3$ you should pick at least one case $d \geqslant 4$. Can you think of any improvement with respect to the way we did the calculation in class for the hyper-sphere?

After the fail in class, I corrected the code, here it is:

```python
import numpy as np
import matplotlib.pyplot as plt
from numba import jit

L = 0.5 # separation distance
d = 4 # dimension of the problem
num_samples = 10**4
trials = 10**3

@jit(nopython=True)
def generate_points(L,d):
    x_limit = 1 - L/2
    yzw_limit = np.sqrt(1-(L/2)**2)
    point = np.zeros(d)
    for i in range(1,d):
        point[0] = np.random.uniform(-x_limit,x_limit)
        point[i] = np.random.uniform(-yzw_limit,yzw_limit)

    return point

@jit(nopython=True)
def montecarlo_method(L,d,num_samples):
```

```python
    x_limit = 1 - L/2
    yzw_limit = np.sqrt(1-(L/2)**2)
    count = 0

    if d == 2:
        exact_area = 2*abs(x_limit)*2*abs(yzw_limit)
        for _ in range(num_samples):
            point = generate_points(L,d)
            if np.sqrt((point[0]+L/2)**2 + point[1]**2) < 1 and np.sqrt((point[0]-L/2)**2 + point[1]**2) < 1:
                count += 1
        return exact_area*count/num_samples
    if d == 3:
        exact_volume = 2**d*(abs(x_limit)*abs(yzw_limit)*abs(yzw_limit))
        for _ in range(num_samples):
            point = generate_points(L,d)
            if np.sqrt((point[0]+L/2)**2 + point[1]**2 + point[2]**2) < 1 and np.sqrt((point[0]-L/2)**2 + point[1]**2 + point[2]**2) < 1:
                count += 1
        return exact_volume*count/num_samples
    if d == 4:
        exact_volume = 2**d*(abs(x_limit)*abs(yzw_limit)*abs(yzw_limit)*abs(yzw_limit))
        for _ in range(num_samples):
            point = generate_points(L,d)
            if np.sqrt((point[0]+L/2)**2 + point[1]**2 + point[2]**2 + point[3]**2) < 1 and np.sqrt((point[0]-L/2)**2 + point[1]**2 + point[2]**2 + point[3]**2) < 1:
                count += 1
        return exact_volume*count/num_samples


@jit(nopython=True)
def run_trials(L,d,num_samples,trials):
    mc_values = np.zeros(trials)
    for i in range(trials):
        mc_values[i] = montecarlo_method(L,d,num_samples)
    return mc_values

approx = montecarlo_method(L,d,num_samples)
print(approx)

result = run_trials(L,d,num_samples,trials)
average = np.mean(result)
print(average)
plt.show()
```

With a more geometrical approach (the correct approach) we get the next results:

- For $d = 2$: Approximate area = 2.152593783591631
- For $d = 3$: Approximate volume = 2.65044825
- For $d = 4$: Approximate hypervolume = 2.9042117521663156