Perspectives PSet 2 Part 2

January 15, 2020

1 Macs 30150 PSet 2

1.1 Part 2

[1]:
```
import requests
from matplotlib import pyplot as plt
from scipy import optimize
import warnings
warnings.filterwarnings("ignore")
```

1.1.1 Excercise 2.1

[20]:
```
import numpy as np
g = lambda x: 0.1*x**4 - 1.5*x**3 + 0.53*x**2 + 2*x + 1

def integr_g(g, a, b, n, method):
    result = 0
    if method == 'midpoint':
        segs = np.linspace(a, b, n)
        mids = (segs[1:]+segs[:-1])/2
        g_m = g(mids)
        result = (b-a)/n * np.sum(g_m)
    elif method == 'trapezoid':
        segs = np.linspace(a, b, n)
        result = (b-a)/(2*n) * (g(segs[0]) + 2*np.sum(g(segs[1:-1])) +␣
 ↪g(segs[-1]))
    elif method == 'simpsons':
        segs = np.linspace(a, b, 2*n)
        result = (b-a)/n/6*(g(segs[0]) + 4*np.sum(g(segs[1::2])) + 2*np.
 ↪sum(g(segs[2::2])) + g(segs[-1]))
    return result
```

[21]:
```
true_value = 4373+1/3
result1 = integr_g(g, -10, 10, 200, 'midpoint')
result2 = integr_g(g, -10, 10, 200, "trapezoid")
result3 = integr_g(g, -10, 10, 200, "simpsons")
print("Midpoint: ", result1, "Difference is ",result1-true_value)
```

```
print("Trapezoid: ", result2, "Difference is ", result2-true_value)
print("Simpsons: ", result3, "Difference is ", result3-true_value)
nv = np.arange(20, 201)
```

```
Midpoint:  4351.122786491231 Difference is  -22.210546842102303
Trapezoid:  4352.154432093296 Difference is  -21.178901240036794
Simpsons:  4341.084377978026 Difference is  -32.24895535530686
```

[ ]:
```
y_mid, y_tra, y_sim=np.zeros(0), np.zeros(0), np.zeros(0)
for i in range(20,201,1):
    y_mid = np.append(y_mid, abs(integr_g(g, -10, 10, i,␣
 ↪'midpoint')-true_value))
    y_tra = np.append(y_tra, abs(integr_g(g, -10, 10, i,␣
 ↪'trapezoid')-true_value))
    y_tra = np.append(y_tra, abs(integr_g(g, -10, 10, i,␣
 ↪"simpsons")-true_value))
plt.plot(nv,y_mid,label="Midpoint")
plt.plot(nv,y_tra,label="Trapezoid")
plt.plot(nv,y_sim,label="Simpsons")
plt.legend()
plt.show()
```

[ ]:
```
ax=plt.gca()
ax.spines['bottom'].set_position('zero')
x = np.linspace(-10, 10,1000)
plt(x,g(x))
```

1.1.2 Excercise 2.2

[22]:
```
import scipy as sp
from scipy.stats import norm
from scipy.integrate import quad

def NC_app(u, s, N, k):
    z = np.linspace(u - k * s, u + k * s, N)
    w = np.zeros(N)
    w[0] = norm.cdf((z[0] + z[1])/2, loc=u, scale=s)
    w[N - 1] = 1 - norm.cdf((z[N - 2] + z[N - 1]) / 2, loc=u, scale=s)
    for i in range(1,N - 1):
        w[i] = quad(lambda x:norm.pdf(x, loc=u, scale=s),(z[i - 1] + z[i]) /␣
 ↪2,(z[i + 1] + z[i]) / 2)[0]
        app = np.sum(w*z)
    return z, w, app
print('Z : ',NC_app(5, 1.5, 11, 3)[0])
print('weight : ',NC_app(5, 1.5, 11, 3)[1])
print('approximation : ',NC_app(5, 1.5, 11, 3)[2])
```

```
Z :  [0.5 1.4 2.3 3.2 4.1 5.  5.9 6.8 7.7 8.6 9.5]
weight :  [0.00346697 0.01439745 0.04894278 0.11725292 0.19802845 0.23582284
 0.19802845 0.11725292 0.04894278 0.01439745 0.00346697]
approximation :  5.0
```

1.1.3 Excercise 2.3

[23]:
```
from scipy.stats import norm
import numpy as np

def nc_lapp(u, s, N, k):
    inp = NC_app(u, s, N, k)
    a = np.exp(inp[0])
    w = inp[1]
    app = np.sum(np.dot(a,w))
    return a, w, app
print("A : ", nc_lapp(5, 1.5, 11, 3)[0])
print("weights : ", nc_lapp(5, 1.5, 11, 3)[1])
print("approximation : ", nc_lapp(5, 1.5, 11, 3)[2])
```

```
A :  [1.64872127e+00 4.05519997e+00 9.97418245e+00 2.45325302e+01
 6.03402876e+01 1.48413159e+02 3.65037468e+02 8.97847292e+02
 2.20834799e+03 5.43165959e+03 1.33597268e+04]
weights :  [0.00346697 0.01439745 0.04894278 0.11725292 0.19802845 0.23582284
 0.19802845 0.11725292 0.04894278 0.01439745 0.00346697]
approximation :  460.5426522031043
```

1.1.4 Excercise 2.4

[24]:
```
app = nc_lapp(10.5, 0.8,11, 3)[2]
print("approximation : ", app)
expected = np.exp(10.5+0.8**2/2)
print('My approximation compare to the exact expected value:')
print(abs(app - expected))
```

```
approximation :  50352.4561927659
My approximation compare to the exact expected value:
341.3691842441476
```

1.1.5 Excercise 3.1

[29]:
```
def Gaus(g,a,b,n):
    w0= [1/n for i in range(n)]
    x0 = [a + i * (b - a) / (n - 1) for i in range(n)]
    iv = w0 + x0
    def func(x):
        output = []
        for i in range(2 * n):
```

```
            w, k  = x[:n], x[n:]
            sum_ = sum(w[j] * (k[j] ** i) for j in range(n))
            output.append((b ** (i + 1) - a ** (i + 1)) / (i + 1) - sum_)
        return tuple(output)
    vec = [l for l in sp.optimize.root(func, iv)['x']]
    w,s  = vec[:n], vec[n:]
    m = 0
    for i in range(n):
        m += w[i] * g(s[i])
    return n
f = lambda x: 0.1 * x ** 4 - 1.5 * x ** 3 + 0.53 * x ** 2 + 2 * x + 1
gaus = Gaus(f,-10,10,3)
mid = integr_g(g, -10, 10, 2000000, 'midpoint')
tra = integr_g(g, -10, 10, 2000000, "trapezoid")
sim = integr_g(g, -10, 10, 2000000, "simpsons")
print("Gaussian quadrature approximation: {:f}, Difference with mid point: {:
 ↪f}".format(gaus, gaus-mid))
print("Gaussian quadrature approximation: {:f}, Difference with trapezoid: {:
 ↪f}".format(gaus, gaus-tra))
print("Gaussian quadrature approximation: {:f}, Difference with simpsons: {:f}".
 ↪format(gaus, gaus-sim))
print("Gaussian quadrature approximation: {:f}, Difference with true value: {:
 ↪f}".format(gaus, gaus-4373.33))
```

```
Gaussian quadrature approximation: 3.000000, Difference with mid point:
-4370.331147
Gaussian quadrature approximation: 3.000000, Difference with trapezoid:
-4370.331147
Gaussian quadrature approximation: 3.000000, Difference with simpsons:
-4370.330110
Gaussian quadrature approximation: 3.000000, Difference with true value:
-4370.330000
```

1.1.6 Excercise 3.2

[30]:
```
q = quad(lambda x: 0.1 * x ** 4 - 1.5 * x ** 3 + 0.53 * x ** 2 + 2 * x + 1,␣
 ↪-10, 10)[0]
print("Gaussian quadrature approximation: {:f}".format(q))
print("The absolute error: {:f}".format(abs(q-(4373+1/3))))
```

```
Gaussian quadrature approximation: 4373.333333
The absolute error: 0.000000
```

1.1.7 Excercise 4.1

[144]:
```python
import numpy as np
np.random.seed(seed=25)
def mc(f, o, n):
    x_1 = np.random.uniform(o[0],o[1],n)
    x_2 = np.random.uniform(o[2],o[3],n)
    c = np.sum(f(x_1,x_2))
    area = (o[3]-o[2])*(o[1]-o[0])
    return area*c/n
f = lambda x,y: x**2+y**2<=1
i = 1
while round(mc(f,[-1,1,-1,1],i),4) != 3.1415:
    i = i + 1
else:
    print("The smallest number :", i)
```

```
The smallest number : 615
```

1.1.8 Excercise 4.2

[31]:
```python
# Imported function
import numpy as np

def isPrime(n):
    '''
    -------------------------------------------------------------------
    This function returns a boolean indicating whether an integer n is a
    prime number
    -------------------------------------------------------------------
    INPUTS:
    n = scalar, any scalar value

    OTHER FUNCTIONS AND FILES CALLED BY THIS FUNCTION: None

    OBJECTS CREATED WITHIN FUNCTION:
    i = integer in [2, sqrt(n)]

    FILES CREATED BY THIS FUNCTION: None

    RETURN: boolean
    -------------------------------------------------------------------
    '''
    for i in range(2, int(np.sqrt(n) + 1)):
        if n % i == 0:
            return False

    return True
```

```python
def primes_ascend(N, min_val=2):
    '''
    --------------------------------------------------------------------
    This function generates an ordered sequence of N consecutive prime
    numbers, the smallest of which is greater than or equal to 1 using
    the Sieve of Eratosthenes algorithm.
    (https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes)
    --------------------------------------------------------------------
    INPUTS:
    N       = integer, number of elements in sequence of consecutive
              prime numbers
    min_val = scalar >= 2, the smallest prime number in the consecutive
              sequence must be greater-than-or-equal-to this value

    OTHER FUNCTIONS AND FILES CALLED BY THIS FUNCTION:
        isPrime()

    OBJECTS CREATED WITHIN FUNCTION:
    primes_vec     = (N,) vector, consecutive prime numbers greater than
                     min_val
    MinIsEven      = boolean, =True if min_val is even, =False otherwise
    MinIsGrtrThn2  = boolean, =True if min_val is
                     greater-than-or-equal-to 2, =False otherwise
    curr_prime_ind = integer >= 0, running count of prime numbers found

    FILES CREATED BY THIS FUNCTION: None

    RETURN: primes_vec
    --------------------------------------------------------------------
    '''
    primes_vec = np.zeros(N, dtype=int)
    MinIsEven = 1 - min_val % 2
    MinIsGrtrThn2 = min_val > 2
    curr_prime_ind = 0
    if not MinIsGrtrThn2:
        i = 2
        curr_prime_ind += 1
        primes_vec[0] = i
    i = min(3, min_val + (MinIsEven * 1))
    while curr_prime_ind < N:
        if isPrime(i):
            curr_prime_ind += 1
            primes_vec[curr_prime_ind - 1] = i
        i += 2

    return primes_vec
```

[32]:
```python
def dig_x(x):
    return x % 1
def equi_distri(n, d, typ):
    p_v= primes_ascend(d)
    if typ == 'weyl':
        return dig_x(n*np.sqrt(p_v))
    elif typ == 'haber':
        return dig_x(n*(n+1)/2*np.sqrt(p_v))
    elif typ == 'nied':
        lis1 = [i/(d+1) for i in range(1,d+1)]
        return dig_x(n*np.power(2,lis1))
    elif typ == 'baker':
        lis2 = [1/i for i in range(1,d+1)]
        return dig_x(n*np.exp(lis2))
print(equi_distri(20,3,"weyl"))
print(equi_distri(20,3,"haber"))
print(equi_distri(20,3,"nied"))
print(equi_distri(20,3,"baker"))
```

```
[0.28427125 0.64101615 0.72135955]
[0.9848481  0.73066959 0.57427527]
[0.7841423  0.28427125 0.63585661]
[0.36563657 0.97442541 0.9122485 ]
```

1.1.9 Exercise 4.3

[33]:
```python
def quasi_mc(fcn, o, n, typ):
    x1, x2 = o[0][0], o[0][1]
    y1, y2 = o[1][0], o[1][1]
    x = [(x2-x1)*equi_distri(i,2,typ)[0]+x1 for i in range(1,n+1)]
    y = [(y2-y1)*equi_distri(i,2,typ)[1]+y1 for i in range(1,n+1)]
    x_ram = np.array(x)
    y_ram = np.array(y)
    cir = np.sum(fcn(x_ram, y_ram))
    a = abs(x1-x2)*abs(y1-y2)
    app = cir/n*a
    return app
fcn = lambda x,y: x**2+y**2<1
sqr = [[-1,1],[-1,1]]
```

[34]:
```python
i = 1
while round(quasi_mc(fcn, sqr, i, 'weyl'),4) != 3.1415:
    i = i + 1
else:
    print("The smallest number of weyl:", i)
```

```
The smallest number of weyl: 1244
```

[35]:

```python
i = 1
while round(quasi_mc(fcn, sqr, i, 'haber'),4) != 3.1415:
    i = i + 1
else:
    print("The smallest number of Haber:", i)
```

```
The smallest number of Haber: 2078
```

[ ]:

```python
i = 1
while round(quasi_mc(fcn, sqr, i, 'neid'),4) != 3.1415:
    i = i + 1
else:
    print("The smallest number of Neiderreiter:", i)
```

[36]:

```python
i = 1
while round(quasi_mc(fcn, sqr, i, 'baker'),4) != 3.1415:
    i = i + 1
else:
    print("The smallest number of Baker:", i)
```

```
The smallest number of Baker: 205
```

[ ]:

```python
quasi_mc(fcn, sqr, 1000, 'neid')
```

[ ]:

```python

```