

Question 2 Fulin Guo

January 21, 2019

Fulin Guo

1 Question 2

1.1 Exercise 2.1

```
In [1]: def g(x):
        g=0.1*(x**4)-1.5*x**3+0.53*x**2+2*x+1
        return g
        def ing(g,a,b,n,method='midpoint'):
            inter=0
            if method=='midpoint':
                for i in range(n+1):
                    inter+=((b-a)/n)*g(a+((2*i+1)*(b-a))/(2*n))
            elif method=='trapezoid':
                for i in range(1,n):
                    inter+=((b-a)/(2*n))*(2*g(a+i*(b-a)/n))
                    inter+=((b-a)/(2*n))*(g(a)+g(b))
            elif method=='Simpsons':
                for i in range(1,n):
                    if i%2==1:
                        inter+=((b-a)/(3*n))*(4*g((a+((i*(b-a)))/n)))
                    else:
                        inter+=((b-a)/(3*n))*(2*g((a+((i*(b-a)))/n)))
                inter+=((b-a)/(3*n))*(g(a)+g(b))
            return inter
        print('The numerical approximation of the intergral using midpoint:', ing(g,-10,10,1000))
        print('The numerical approximation of the intergral using trapezoid:',ing(g,-10,10,1000))
        print('The numerical approximation of the intergral using Simpsons:',ing(g,-10,10,1000))
        print('The difference between the midpoint method and the true value:',ing(g,-10,10,1000)-g(-10)+g(10))
        print('The difference between the trapezoid method and the true value:',ing(g,-10,10,1000)-g(-10)+g(10))
        print('The difference between the Simpsons method and the true value:',ing(g,-10,10,1000)-g(-10)+g(10))
```

The numerical approximation of the intergral using midpoint: 4373.324813312664

The numerical approximation of the intergral using trapezoid: 4373.333333360767

The numerical approximation of the intergral using Simpsons: 4373.333333333468

The difference between the midpoint method and the true value: -0.0051866873354811105

The difference between the trapezoid method and the true value: 0.0033333607671011123

The difference between the Simpsons method and the true value: 0.003333333467708144

1.2 Exercise 2.2

```
In [2]: import scipy.stats
import numpy as np
def f(mu,siga,n,k):
    w=np.array([0.0]*n)
    z=np.linspace(mu-k*siga,mu+k*siga,n)
    w[0]=scipy.stats.norm.cdf((z[0]+z[1])/2,mu,siga) # The i=1 case
    w[-1]=1-scipy.stats.norm.cdf((z[-2]+z[-1])/2,mu,siga) # The i=N case
    for i in range(1,n-1): # The 1<i<N case
        w[i]=scipy.stats.norm.cdf((z[i+1]+z[i])/2,mu,siga)-scipy.stats.norm.cdf((z[i]+z[i-1])/2,mu,siga)
    return [w,z]
f(0,1,11,3) # Set the mu=0, sita=1, N=11, and k=3

Out[2]: [array([0.00346697, 0.01439745, 0.04894278, 0.11725292, 0.19802845,
0.23582284, 0.19802845, 0.11725292, 0.04894278, 0.01439745,
0.00346697]),
array([-3. , -2.4, -1.8, -1.2, -0.6, 0. , 0.6, 1.2, 1.8, 2.4, 3. ])]
```

The two arrays above represent W_n and Z_n respectively.

1.3 Exercise 2.3

```
In [3]: import math
def lf(mu,siga,n,k):
    w=np.array([0.0]*n)
    z=np.linspace(mu-siga*k,mu+siga*k,n)
    w[0]=scipy.stats.norm.cdf((z[0]+z[1])/2,mu,siga)
    w[n-1]=1-scipy.stats.norm.cdf((z[n-2]+z[n-1])/2,mu,siga)
    for i in range(1,n-1):
        w[i]=scipy.stats.norm.cdf((z[i+1]+z[i])/2,mu,siga)-scipy.stats.norm.cdf((z[i]+z[i-1])/2,mu,siga)
    z=np.array([math.exp(i) for i in z])
    return [w,z]
lf(0,1,11,3)

Out[3]: [array([0.00346697, 0.01439745, 0.04894278, 0.11725292, 0.19802845,
0.23582284, 0.19802845, 0.11725292, 0.04894278, 0.01439745,
0.00346697]),
array([ 0.04978707, 0.09071795, 0.16529889, 0.30119421, 0.54881164,
1. , 1.8221188 , 3.32011692, 6.04964746, 11.02317638,
20.08553692])]
```

We could see from the above answer that the space nodes (the first array) are not evenly distributed, but the weights (the second array) are the same with those in Exercise 2.2

1.4 Exercies 2.4

```
In [21]: a=lf(10.5,0.8,11,3)
        sum=0
        for i in range(11):
            sum+=a[0][i]*a[1][i]
        print('The approximation of the average income in U.S. is:',sum)
        relative=abs((sum-math.exp(10.5+0.32)))/math.exp(10.5+0.32)
        print('The absolute error of the approximation is:',sum-math.exp(10.5+0.32))
        print('The relative error of the approximation is:',relative)
```

The approximation of the average income in U.S. is: 50352.45619276591

The absolute error of the approximation is: 341.3691842441549

The relative error of the approximation is: 0.006825870115280368

1.5 Exercise 3.1

```
In [5]: def g(x):
        x=0.1*(x**4)-1.5*x**3+0.53*x**2+2*x+1
        return x
        def func(x):
            return [x[0]+x[1]+x[2]-20,
                    x[0]*x[3]+x[1]*x[4]+x[2]*x[5]-0,
                    x[0]*(x[3]**2)+x[1]*(x[4]**2)+x[2]*(x[5]**2)-2000/3,
                    x[0]*(x[3]**3)+x[1]*(x[4]**3)+x[2]*(x[5]**3)-0,
                    x[0]*(x[3]**4)+x[1]*(x[4]**4)+x[2]*(x[5]**4)-((10**5)*2)/5,
                    x[0]*(x[3]**5)+x[1]*(x[4]**5)+x[2]*(x[5]**5)-0]
        s=scipy.optimize.root(func,[20/3,20/3,20/3,-10,0,10])
        print('The approximated intergral using Gaussian quadrature:', s.x[0]*g(s.x[3])+s.x[1]*g(s.x[4])+s.x[2]*g(s.x[5]))
        print('The absolute error to the answer in exercies 2.1:',s.x[0]*g(s.x[3])+s.x[1]*g(s.x[4])+s.x[2]*g(s.x[5]))
        print('The absolute error to the true value:',s.x[0]*g(s.x[3])+s.x[1]*g(s.x[4])+s.x[2]*g(s.x[5]))
```

The approximated intergral using Gaussian quadrature: 4373.333333588602

The absolute error to the answer in exercies 2.1: 0.00852027593737148

The absolute error to the true value: 0.0033335886018903693

1.6 Exercies 3.2

```
In [6]: from scipy import integrate
        print('The approximated interger using the scipy.integrate.quad is:',scipy.integrate.quad(g,-10,10)[0])
        print('The absloute error using this method is:',scipy.integrate.quad(g,-10,10)[1])
```

The approximated interger using the scipy.integrate.quad is: 4373.333333333334

The absloute error using this method is: 8.109531705284936e-11

1.7 Exercise 4.1

```
In [24]: import random
def f(g,oumu,n):
    ans=0
    random.seed(25)
    for i in range(n):
        x=random.uniform(oumu[0][0],oumu[0][1])
        y=random.uniform(oumu[1][0],oumu[1][1])
        ans+=g(x,y)
    return ans
def g(x,y):
    if x**2+y**2<1:
        return 1
    else:
        return 0
print('N=10000:',4*f(g,[[ -1,1],[ -1,1]],10000)/10000)
print('N=100000:',4*f(g,[[ -1,1],[ -1,1]],100000)/100000)
print('N=1000000:',4*f(g,[[ -1,1],[ -1,1]],1000000)/1000000)
print('N=990000:',4*f(g,[[ -1,1],[ -1,1]],990000)/990000)
print('N=999000:',4*f(g,[[ -1,1],[ -1,1]],999000)/999000)
```

N=10000: 3.126

N=100000: 3.14336

N=1000000: 3.141592

N=990000: 3.141737373737374

N=999000: 3.1415935935935937

Therefore, in order to match the true value to the 4th decimal, we need set N at least 999000 (i.e. about 1000000)

1.8 Exercies 4.2

```
In [16]: def prime(x):
    # Return True if x is a prime number. Otherwise, return False
    ans=True
    for i in range(2,int(math.sqrt(x))+1):
        if x%i==0:
            return False
        break
    return ans
def p(n):
    # Return the nth prime number
    a=[]
    i=2
    while len(a)<n:
        if prime(i)==True:
            a.append(i)
```

```

        i+=1
    return a[-1]
def quam(n,d,method='Weyl'):
    if method=='Weyl':
        return [n*(p(i)**(1/2))-int(n*(p(i)**(1/2))) for i in range(1,d+1)]
    elif method=='Haber':
        return [(((n*(n+1))/2))*((p(i))**(1/2))-int((((n*(n+1))/2))*((p(i))**(1/2)))]
    elif method=='Niederreiter':
        return [n*(2**(i/(n+1)))-int(n*(2**(i/(n+1)))) for i in range(1,d+1)]
    elif method=='Baker':
        return [n*math.exp(1/i)-int(n*math.exp(1/i)) for i in range(1,d+1)]

```

1.9 Exercise 4.3

```

In [27]: import random
random.seed(25)
def ff(g,n,method='Weyl'):
    ans=0
    for i in range(1,n+1):
        x=quam(i,2,method)[0]
        y=quam(i,2,method)[1]
        ans+=g(x,y)
    return ans
def g(x,y):
    if x**2+y**2<1:
        return 1
    else:
        return 0
print('N=10000',4*ff(g,10000)/10000)
print('N=100000',4*ff(g,100000)/100000)
print('N=1000000',4*ff(g,1000000)/1000000)

```

```

N=10000 3.1412
N=100000 3.14036
N=1000000 3.14148

```

We could from the above answer that in order to match the true value to the 4th decimal, we need set N at least about 1000000