hw2 Problem2

January 13, 2019

0.0.1 Import Packages

```
In [1]: import numpy as np
        import scipy as sp
        from scipy.stats import norm
        from scipy.integrate import quad
```

0.0.2 14.1

```
In [2]: def integrate(g,a,b,N,method='midpoint'):
            if method not in ['midpoint','trapezoid', 'Simpsons']:
                raise ValueError
            else:
                if method == 'midpoint':
                    counter = 0
                    for i in range(N):
                        counter += g(a+(2*i+1)*(b-a)/(2*N))
                    return (b-a)*counter/N
                if method == 'trapezoid':
                    counter = g(a)+g(b)
                    for i in range(1,N):
                        counter += 2*g(a+i*(b-a)/N)
                    return (b-a)*counter/(2*N)
                if method == 'Simpsons':
                    counter = g(a)+g(b)+4*g(a+(2*N-1)*(b-a)/(2*N))
                    for i in range(1,N):
                        counter += 4*g(a+(2*i-1)*(b-a)/(2*N))
                        counter += 2*g(a+2*i*(b-a)/(2*N))
                    return (b-a)*counter/(2*N)
        integrate(lambda x: 0.1*x**4-1.5*x**3+0.53*x**2+2*x+1, -10, 10, 10000)
```

```
Out[2]: 4373.333196466632
```

0.0.3 14.2

```
In [3]: def N_C(N, mu=0, sigma=1, k=3):
            Z = np.linspace(mu-k*sigma, mu+k*sigma, N)
            weight = np.zeros(N)
```

```
            weight[0] = norm.cdf((Z[0]+Z[1])/2, loc=mu, scale=sigma)
            for i in range(1,N-1):
                weight[i] = quad(lambda x:norm.pdf(x, loc=mu, scale=sigma),(Z[i-1]+Z[i])/2,(Z[i
            weight[N-1] = 1-norm.cdf((Z[N-2]+Z[N-1])/2, loc=mu, scale=sigma)
            return Z, weight
        Z, weight = N_C(11)
        print('Z =',Z)
        print('weight = ',weight)
```

```
Z = [-3.  -2.4 -1.8 -1.2 -0.6  0.   0.6  1.2  1.8  2.4  3. ]
weight =  [0.00346697 0.01439745 0.04894278 0.11725292 0.19802845 0.23582284
 0.19802845 0.11725292 0.04894278 0.01439745 0.00346697]
```

0.0.4 14.3

```
In [4]: def new_N_C(N, mu=0, sigma=1, k=3):
            Z = np.linspace(mu-k*sigma, mu+k*sigma, N)
            A = np.e**Z
            weight = np.zeros(N)
            weight[0] = norm.cdf((Z[0]+Z[1])/2, loc=mu, scale=sigma)
            for i in range(1,N-1):
                weight[i] = quad(lambda x:norm.pdf(x, loc=mu, scale=sigma),(Z[i-1]+Z[i])/2,(Z[i
            weight[N-1] = 1-norm.cdf((Z[N-2]+Z[N-1])/2, loc=mu, scale=sigma)
            return A, weight
        A, weight = new_N_C(11)
        print('A =',A)
        print('weight = ',weight)
```

```
A = [ 0.04978707  0.09071795  0.16529889  0.30119421  0.54881164  1.
  1.8221188   3.32011692  6.04964746 11.02317638 20.08553692]
weight =  [0.00346697 0.01439745 0.04894278 0.11725292 0.19802845 0.23582284
 0.19802845 0.11725292 0.04894278 0.01439745 0.00346697]
```

0.0.5 14.4

```
In [5]: Myresult = new_N_C(1001, mu=10.5, sigma=0.8, k=10)[0]@new_N_C(1001, mu=10.5, sigma=0.8
        Exactresult = np.e**(10.5+0.5*(0.8**2))
        print('The difference between my result and the exact expectation is', Myresult[0]-Exac
```

```
The difference between my result and the exact expectation is 0.5334533017885406
```

0.0.6 14.5

```
In [6]: def Gaussian(g,a,b,N=3):
            init_weight = [1/N for i in range(N)]
            init_x = [a+i*(b-a)/(N-1) for i in range(N)]
```

```
        init = init_weight+init_x
        def func(x):
            result = []
            for i in range(2*N):
                weight = x[:N]
                node = x[N:]
                result.append((b**(i+1)-a**(i+1))/(i+1)-sum(weight[k]*(node[k]**i) for k in
            return tuple(k for k in result)
        Vector = list(k for k in sp.optimize.root(func, init)['x'])
        weight = Vector[:N]
        node = Vector[N:]
        counter = 0
        for i in range(N):
            counter += weight[i]*g(node[i])
        return counter
    Gauss = Gaussian(lambda x: 0.1*x**4-1.5*x**3+0.53*x**2+2*x+1, -10, 10)
    Newton = integrate(lambda x: 0.1*x**4-1.5*x**3+0.53*x**2+2*x+1, -10, 10, 10000)
    Exact = 0.02*(10**5-(-10)**5)+0.53/3*(10**3-(-10)**3)+20
    print("The result of Gaussian approximate is", Gauss, 'and the absolute error is', abs
    print("The result of Newton-Cotes approximate is", Newton, 'and the absolute error is'
```

```
The result of Gaussian approximate is 4373.333333189601 and the absolute error is 1.4373199519
The result of Newton-Cotes approximate is 4373.333196466632 and the absolute error is 0.0001368
```

0.0.7 14.6

```
In [7]: Quad = quad(lambda x: 0.1*x**4-1.5*x**3+0.53*x**2+2*x+1, -10, 10)[0]
        print("The result of Python Gaussian approximate is", Quad, 'and the absolute error is
```

```
The result of Python Gaussian approximate is 4373.333333333334 and the absolute error is 9.094
```

0.0.8 14.7

```
In [8]: def M_C(N, func=None, omega=[-1,1,-1,1]):
            counter = 0
            x_1 = np.random.uniform(omega[0],omega[1],size=N)
            x_2 = np.random.uniform(omega[2],omega[3],size=N)
            def g(X):
                x,y = X[0], X[1]
                if x**2+y**2<=1:
                    return 1
                else:
                    return 0
            for i in range(N):
                X = (x_1[i],x_2[i])
                if func is None:
                    counter += g(X)
```

```
            else:
                counter += func(X)
        return 4*counter/N
    judge = False
    min_N = 0
    while judge is False:
        min_N += 1
        judge = (round(M_C(min_N),4)==3.1415)
    print("The smallest number of random draws N that matches the true value of pi to the 4
```

```
The smallest number of random draws N that matches the true value of pi to the 4th decimal 3.14
```

0.0.9 14.8

```
In [9]: def equidistribution(n,d,Type='weyl'):
            def prime_list(d):
                if d == 1:
                    return [2]
                List = [2,3]
                i = 3
                while len(List)<d:
                    i+=1
                    for k in range(2, np.sqrt(i)//1):
                        if i%k == 0:
                            break
                    else:
                        List.append(i)
                return List
            def rational_list(d):
                return [1/(i+1) for i in range(d)]
            def cut(x):
                return x-x//1
            if Type == 'weyl':
                return tuple(cut(n*np.sqrt(prime_list(d)[i])) for i in range(d))
            elif Type == 'haber':
                return tuple(cut(n*(n+1)*0.5*np.sqrt(prime_list(d)[i])) for i in range(d))
            elif Type == 'nie':
                return tuple(cut(n*(2**(i/(n+1)))) for i in range(d))
            elif Type == 'baker':
                return tuple(cut(n*(np.e**(rational_list(d)[i]))) for i in range(d))
```

0.0.10 14.9

```
In [10]: def quasi_M_C(N, func=None, omega=[-1,1,-1,1]):
             counter = 0
             x = []
             for k in range(N):
```

```
            x.append((2*equidistribution(k,2)[0]-1,2*equidistribution(k,2)[1]-1))
        def g(X):
            x,y = X[0], X[1]
            if x**2+y**2<=1:
                return 1
            else:
                return 0
        for i in range(N):
            X = x[i]
            if func is None:
                counter += g(X)
            else:
                counter += func(X)
        return 4*counter/N
    judge = False
    min_N1 = 0
    while judge is False:
        min_N1 += 1
        judge = (round(M_C(min_N1),4)==3.1415)
    print("The smallest number of random draws N for former method that matches the true v
    judge = False
    min_N2 = 0
    while judge is False:
        min_N2 += 1
        judge = (round(quasi_M_C(min_N2),4)==3.1415)
    print("The smallest number of random draws N for new method that matches the true valu
```

```
The smallest number of random draws N for former method that matches the true value of pi to th
The smallest number of random draws N for new method that matches the true value of pi to the 4
```