hw2 Problem2

January 20, 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 2.1

```
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 2.2

```
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):
        func = lambda x:norm.pdf(x, loc=mu, scale=sigma)
        weight[i] = quad(func,(Z[i-1]+Z[i])/2,(Z[i+1]+Z[i])/2)[0]
    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 2.3

```
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):
                func = lambda x:norm.pdf(x, loc=mu, scale=sigma)
                weight[i] = quad(func,(Z[i-1]+Z[i])/2,(Z[i+1]+Z[i])/2)[0]
            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 2.4

```
In [5]: A, weight = new_N_C(1001, mu=10.5, sigma=0.8, k=10)
        Myresult = A@weight.reshape(-1,1)
        Exactresult = np.e**(10.5+0.5*(0.8**2))
        print('The difference between my result and the exact expectation is:')
        print(Myresult[0]-Exactresult)
```

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

0.0.6 3.1

```
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:]
                    Sum = sum(weight[k]*(node[k]**i) for k in range(N))
                    result.append((b**(i+1)-a**(i+1))/(i+1)-Sum)
                return tuple(k for k in result)
            Vector = [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)
        print('The absolute error of Gaussian approximate is', abs(Gauss-Exact))
        print("The result of Newton-Cotes approximate is", Newton)
        print('The absolute error of Newton-Cotes approximate is', abs(Newton-Exact))
```

```
The result of Gaussian approximate is 4373.333333189601
The absolute error of Gaussian approximate is 1.4373199519468471e-07
The result of Newton-Cotes approximate is 4373.333196466632
The absolute error of Newton-Cotes approximate is 0.00013686670081369812
```

0.0.7 3.2

```
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)
        print('The absolute error of Python Gaussian approximate is', abs(Quad-Exact))
```

```
The result of Python Gaussian approximate is 4373.333333333334
The absolute error of Python Gaussian approximate is 9.094947017729282e-13
```

0.1 4

```
In [8]: 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
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
```

0.1.1 4.1

```
In [9]: 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,y):
                if x**2+y**2<=1:
                    return 1
                else:
                    return 0
            for i in range(N):
                x,y = x_1[i],x_2[i]
                if func is None:
                    counter += g(x,y)
                else:
                    counter += func(x,y)
            return 4*counter/N
        np.random.seed(25)
        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 is", min_N)
```

```
The smallest number of random draws N is 615
```

0.1.2 4.2

```
In [10]: def equidistribution(n,d,Type='weyl'):
             prime_vector = primes_ascend(d)
             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_vector[i])) for i in range(d))
             elif Type == 'haber':
                 return tuple(cut(n*(n+1)*0.5*np.sqrt(prime_vector[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))
```

0.1.3 4.3

```
In [11]: 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
         print("The smallest number of random draws N for former method is", min_N)
         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 is", min_N2)
```

```
The smallest number of random draws N for former method is 615
```

```
The smallest number of random draws N for new method is 1230
```

Since the smallest number of random draws N for Monte Carlo integration that matches the true value of π to the 4th decimal 3.1415 is smaller, the rates of convergence of Monte Carlo integration is faster than that of quasi-Monte Carlo integration