Lecture12

March 14, 2019

1 Lecture 12: Numerical Integration and Monte Carlo

Overview: * Numerical integration. * Simple Monte Carlo integration. * Importance sampling.
Next Lecture: * Markov chain Monte Carlo and the Metropolis Algorithm. —

```
In [1]: %matplotlib notebook
        import numpy as np
        import math
        from scipy import integrate
```

1.1 Dart Board Estimate of π

The code in the cell below generates a set of random coordinates inside our unit square and calculates the magnitude of the vector defined by these coordinates.

- Run the code in the cell and call out the number generated for Prof. Plumb to plot on the board.

```
In [2]: x = 2 * (np.random.random([1, 2])) - 1
        print(np.sqrt(np.sum(x**2)))
```

```
1.0222317229234177
```

1.1.1 Functions to integrate, and some exact results

```
In [3]: def gaussian(x):
            return np.exp(-x**2)

        # A Function that is not well behaved
        def Fermi(x):
            num = 1/np.sqrt(x)
            den = np.exp(x)+1
            return num/den

        # area of a unit circle
        def sphere(x):
            """
```

```
        return 1 if point is inside radius, zero otherwise
        x is a multidimensional vector, must have dimension greater than 1
        """
        r = np.sum(x**2, axis = 1)
        a = (r<=1).astype(int)
        return 1.0*a

    # volume of a hypersphere in n dimensions
    hypersphere = lambda r, n: math.pi**(n / 2)/math.gamma(n / 2 + 1)*r**n

    # exact integral of a Gaussian
    analyticalIntegral = np.sqrt(np.pi)
```

In [4]:
```
# Numerically integrate a Gaussian

def riemannSum(f):
    width = 10.0
    n = 100
    dx = width/n
    x = np.arange(-0.5*width, 0.5*width, width/n)

    return np.sum(f(x)) * dx

def Simpson(f):
    width = 10.0
    n = 100
    dx = width/n
    x = np.arange(-0.5*width, 0.5*width, width/n)

    s = (f(-0.5*width)+f(0.5*width))
    return (2*f(x[2:-2:2]).sum() + 4*f(x[1:-2:2]).sum()+s) * dx/3

# using Scipy's built in integration schemes
scipyIntegral = integrate.quad(gaussian, -100.0, 100.0)
```

In [13]:
```
#np.random.seed(256)
def naiveMonteCarlo(f, limits = [-10,10],d = 1, n_points = 1000, NSamples =100):
    """
    Implement a mean value Monte-Carlo Integration in d dimensions

    f is function to integrate, must take an input vector x of dimension d

    limits define the range of integration, this function only works for integration 
    all dimensitons

    n_points are number of points to sample in domain
    NSamples number of time to repeat integration, decrease statistical noise
    """
```

```
        width = np.abs(limits[1] - limits[0])
        samples = np.zeros(NSamples)

        for i in range(NSamples):
            x = width * (np.random.random([n_points, d])) + limits[0]
            samples[i] = width**d * np.sum(f(x))/n_points

        return samples.mean(), samples.std()
```

```
In [14]: print("Analytical (exact) integral = ", analyticalIntegral)
         print("Riemann Sum = ", riemannSum(gaussian))
         print("Simpsons Rule = ", Simpson(gaussian))
         print("SciPy Integral = ", scipyIntegral)
         print("Naive Mean Value Monte Carlo = ", naiveMonteCarlo(gaussian))
```

```
Analytical (exact) integral =  1.7724538509055159
Riemann Sum =  1.7724538509025694
Simpsons Rule =  1.772453850891228
SciPy Integral =  (1.772453850905516, 1.976815268282025e-10)
Naive Mean Value Monte Carlo =  (1.7795263069564624, 0.1402938673272462)
```

1.1.2 Tasks

- Use the mean value method Monte Carlo method to estimate the value of π, (area of unit circle) to a higher accuracy than what was done in the demonstration.
- Use the Monte Carlo integrator to n-dimensions to find the volume of a hypersphere in 10 dimensions? Compare this results with Simpsons rule and the exact value.
- Can you confirm the error on the MC integration is independent of the number of dimensions?

```
In [16]: print("Naive Mean Value Monte Carlo = ", naiveMonteCarlo(gaussian, d = 10))
```

```
Naive Mean Value Monte Carlo =  (9076793553888.781, 211663373947.3855)
```

1.2 Importance sampling Monte Carlo

- Review the importance sampling method below. Do you understand all of the steps?
- Can you modify the method and integrate a 4 dimensional Gaussian function?

```
In [7]: def p_normal(stdev, x): # normal distribution
            s = 1.0 / stdev
            s2 = s**2
            return np.exp(-s2 * x**2) * s / np.sqrt(np.pi)

        def importanceSampledMonteCarlo(f, p, NSamples=10):
            n = 10000
```

```
        stdev = 1.0
        samples = np.zeros(NSamples)

        for i in range(NSamples):
            # sample random values from a normal distribution
            x = np.random.normal(loc = 0.0, scale = np.sqrt(0.5) * stdev, size = n)
            samples[i] = (f(x) / p(stdev, x)).mean()

        return samples.mean(), samples.std()
```

In [8]:
```
print("Naive Mean Value Monte Carlo = ", naiveMonteCarlo(gaussian))
print("Importance Sampled Monte Carlo = ", importanceSampledMonteCarlo(gaussian, p_norm
```

```
Naive Mean Value Monte Carlo =  (1.7825844596442761, 0.13423238476906396)
Importance Sampled Monte Carlo =  (1.7724538509055165, 0.0)
```

In [ ]:

In [ ]: