MonteCarloMethod

February 11, 2015

An Introduction to the Monte Carlo Method

This set of notes introduces some basic concepts regarding statistical limit theory as well as the Monte Carlo method.

Statistical Limit Theory

Limiting Distribution

Consider a sequence of random variables $Y_1, Y_2, \ldots$ with a corresponding sequence of CDFs $G_1(y), G_2(y), \ldots$ so that for each $n = 1, 2, \ldots$

$$G_n(y) = P[Y_n \leq y]$$

Definition
If $Y_n \sim G_n(y)$ for each $n = 1, 2, \ldots$, and if for some CDF $G(y)$

$$\lim_{n\to\infty} G_n(y) = G(y)$$

for all values y at which $G(y)$ is continous, then the sequence $Y_1, Y_2, \ldots$ is said to **converge in distribution** to $Y \sim G(y)$, denoted by $Y_n \xrightarrow{d} Y$. The distribution corresponding to the CDF $G(y)$ is called the **limiting distribution** of Y_n.

Basic Convergence Concepts

Definition
A sequence of random variables, $Y_1, Y_2, \ldots$, is said to **converge in probability** to the random variable Y if

$$\lim_{n\to\infty} P[|Y_n - Y| < \varepsilon] = 1$$

for every $\varepsilon > 0$.
Definition
The sequence **converges almost surely** to Y if

$$P[lim_{n\to\infty}|Y_n - Y| < \varepsilon] = 1$$

for every $\varepsilon > 0$.

The Law of Large Numbers

Definition
The law which states that the larger a sample, the nearer its mean is to that of the parent population from which the sample is drawn. More formally: for every $\varepsilon > 0$, the probability

$$\{|\bar{Y} - Y| > \varepsilon\} \to 0 \quad \text{as} \quad n \to \infty$$

where n is the sample size, $\bar{Y}$ is the sample mean, and μ is the parent mean.
More rigorous definitions are the following:
For i.i.d sequences of one-dimensional random variables $Y_1, Y_2, \ldots$, let $\bar{Y}_n = \frac{1}{n}\sum_{i=1}^{n} Y_i$.
The *weak law of large numbers* states that $\bar{Y}_n$ converges in probability to $\mu = E\{Y_i\}$ if $E\{|Y_i|\} < \infty$.
The *strong law of large numbers* states that $\bar{Y}_n$ converges almost surely to μ if $E\{|Y_i|\} < \infty$.
Both results hold under the more stringent but easily checked condition that $var\{Y_i\} = \sigma^2 < \infty$.

Using Simulation to Check the Law of Large Numbers

We can use simulation to check the Law of Large Numbers. Consider a fair die with six sides and outcomes $Y = \{1, 2, 3, 4, 5, 6\}$, each with $P[Y_i = y] = \frac{1}{6}$. The true mean is

$$\mu = E\{Y\} = \frac{1}{6}[1 + 2 + 3 + 4 + 5 + 6] = 3.5$$

We can verify this in Python:

```
In [11]: %matplotlib inline
```

```
In [12]: import numpy as np
         import matplotlib.pyplot as plt

         x = np.arange(1,7)
         mu = (1/6) * x.sum()
```

```
In [13]: mu
```

```
Out[13]: 3.5
```

Now let's simulate some rolls of the die and collect some data. We will let our sample size increase and plot the estimated mean.
We can simulate a single roll of the die as follows:

```
In [14]: np.random.randint(1,7)
```

```
Out[14]: 2
```

We can also simulate many at once as follows:

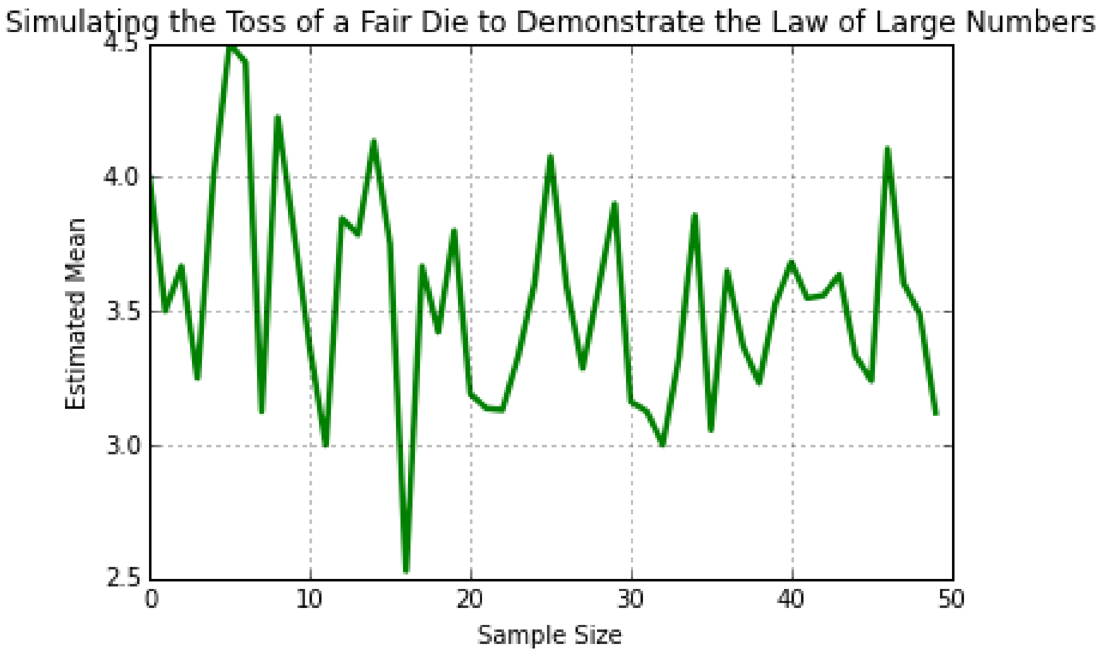
```
In [15]: np.random.randint(1,7, size=100)
```

```
Out[15]: array([4, 2, 6, 6, 3, 5, 4, 5, 3, 4, 3, 4, 4, 4, 3, 4, 4, 1, 5, 5, 3, 1, 1,
                4, 3, 3, 5, 2, 6, 4, 5, 1, 5, 5, 6, 1, 5, 2, 6, 6, 6, 1, 2, 2, 1, 4,
                4, 1, 6, 2, 1, 4, 3, 6, 3, 3, 3, 6, 1, 4, 3, 4, 2, 1, 5, 6, 5, 3, 4,
                1, 1, 4, 3, 5, 6, 5, 3, 3, 2, 6, 4, 5, 1, 4, 2, 2, 6, 2, 2, 1, 3, 6,
                5, 5, 1, 6, 4, 5, 6, 3])
```

```
In [25]: m = 50
         sizes = np.arange(1,m + 1)
         means = np.zeros((m,))

         for i in range(len(sizes)):
             y = np.random.randint(1,7, size=sizes[i])
             means[i] = y.mean()

         plt.plot(means, 'g', lw = 2.5)
```

```
        plt.grid(True)
        plt.title("Simulating the Toss of a Fair Die to Demonstrate the Law of Large Numbers")
        plt.xlabel("Sample Size")
        plt.ylabel("Estimated Mean")
```

```
Out[25]: <matplotlib.text.Text at 0x7f8101b93ba8>
```



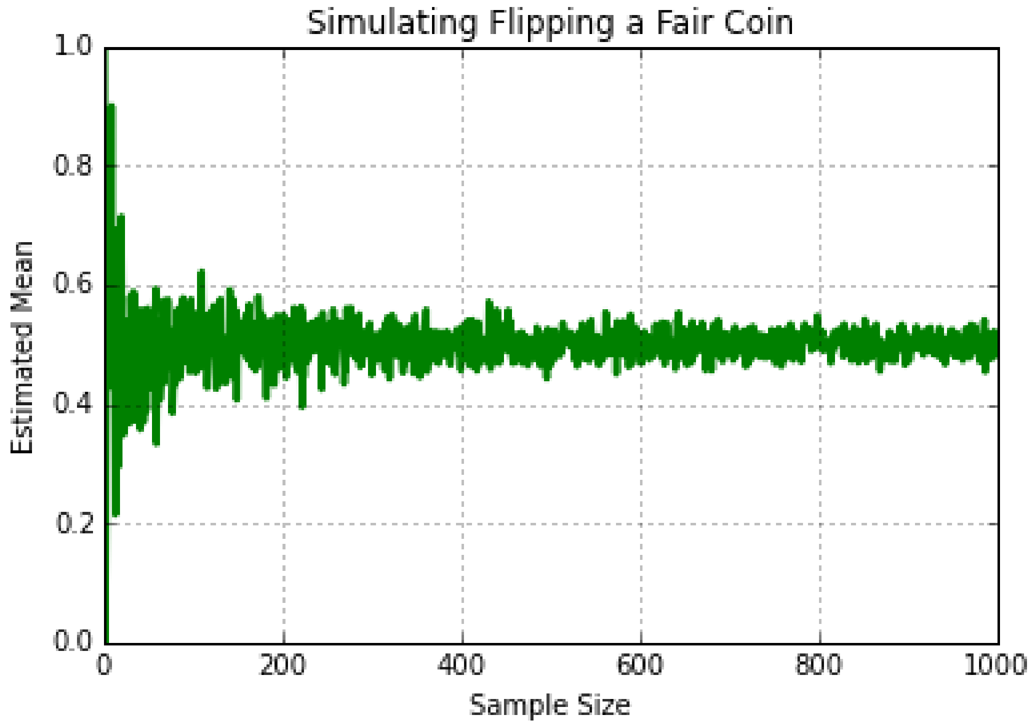
We can do a similar simulation for the flipping of a fair coin. We can simulate the flip of a coin with the Binomial distribution as follows:

```
In [23]: np.random.binomial(1, 0.5, 100)
```

```
Out[23]: array([1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0,
                0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1,
                1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1,
                0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0,
                1, 1, 1, 0, 0, 1, 0, 1])
```

```
In [30]: m = 1000
         sizes = np.arange(1,m + 1)
         means = np.zeros((m,))

         for i in range(len(sizes)):
             y = np.random.binomial(1, 0.5, sizes[i])
             means[i] = y.mean()

         plt.plot(means, 'g', lw = 2.5)
         plt.grid(True)
         plt.title("Simulating Flipping a Fair Coin")
         plt.xlabel("Sample Size")
         plt.ylabel("Estimated Mean")
```

```
Out[30]: <matplotlib.text.Text at 0x7f81019f45f8>
```



The Central Limit Theorem

The theorem that states that, if samples of size n are taken from a parent population with mean μ and standard deviation σ, then the distribution of their means will be approximately normal with:

$$\text{Mean} = \mu$$

and

$$\text{Standard deviation} = \frac{\sigma}{\sqrt{n}}$$

As the sample size n increases, this distribution approaches the normal distribution with increasing accuracy. Thus in the limit, as $n \to \infty$, the distribution of the sample means $\to$ Normal, mean μ, standard deviation $\sigma/\sqrt{n}$.
If the parent population is itself normal, the distribution of the sample means will be normal, whatever the sample size. If the parent population is of finite size N, two possibilties arise:

1. If the sampling is carried out with replacement, the theorem stands as stated;
2. If there is no replacement, the standard deviation of the sample mean is:

$$\frac{\sigma}{\sqrt{n}}\sqrt{\frac{N-n}{N-1}}$$

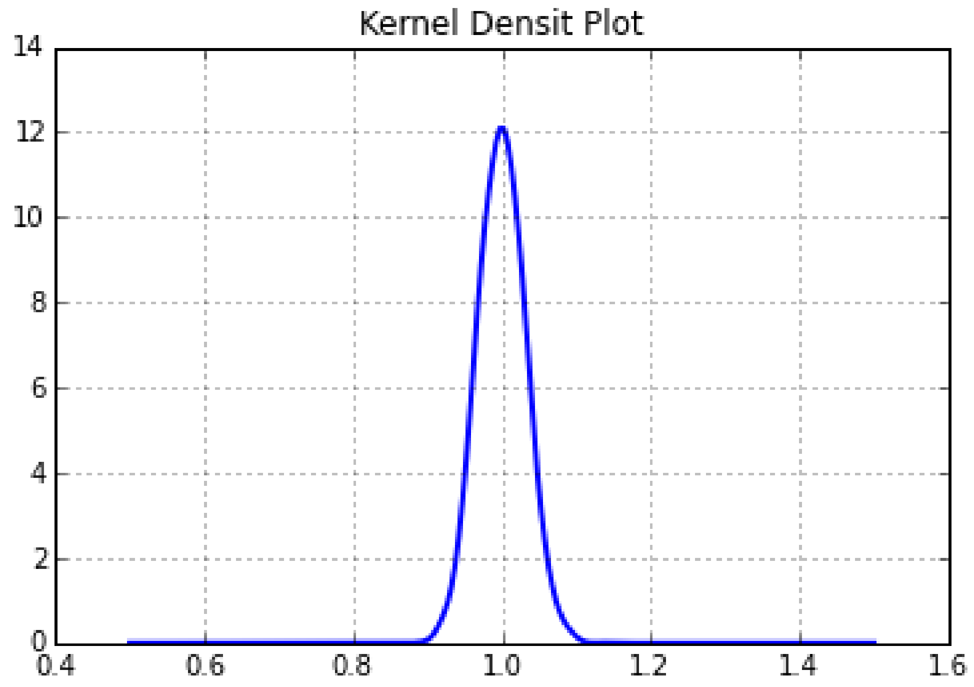
The factor $\sqrt{\frac{N-n}{N-1}}$ is called the **finite population correction.**

The central limite theorem provides the basis for much of sampling theory; it can be summed up, as follows. If n is not small, the sampling distribution of the means is approximately normal, has mean $= \mu$ (the parent mean), and has standard deviation $\sigma/\sqrt{n}$ (where σ is the parent standard deviation).
We can use simulation to build intuition for the central limit theorem as well. Consider the mean of a sample from an exponential distribution. Recall that the density of the exponential distribution is the following:

$$f(x) = \frac{1}{\theta} e^{-x/\theta}$$

for $\theta > 0$ and $x > 0$.
In `Python` we can simulate from the exponential distribution as follows:

```
In [56]: np.random.exponential(size=100)

Out[56]: array([ 2.11002182,  0.12337033,  1.08528294,  0.92795529,  3.21330863,
                 0.4153186 ,  3.11930452,  0.29012437,  0.43103097,  1.44610688,
                 1.07504128,  0.16066788,  0.31372567,  0.43684893,  0.02001204,
                 0.41539643,  0.17038639,  0.76048346,  0.13219559,  0.63284794,
                 2.82377058,  2.14737159,  0.75740743,  2.26059043,  1.98769895,
                 2.8080418 ,  1.27726222,  0.62044994,  1.69881708,  0.30315011,
                 1.80323159,  2.34383145,  0.52126866,  0.09651965,  0.46522634,
                 2.24878567,  0.62223617,  0.04700664,  0.35589569,  4.05364838,
                 1.29517861,  0.42479826,  0.37288011,  1.2141579 ,  1.35677263,
                 0.77510795,  2.0336761 ,  0.53522305,  0.26501789,  0.8938838 ,
                 0.02864191,  0.47618018,  0.75065438,  3.30501218,  0.85148293,
                 0.66879873,  0.01237078,  0.55991656,  2.7196078 ,  1.31449542,
                 1.96469666,  0.90228788,  0.08598953,  0.93728401,  0.470582  ,
                 2.6790866 ,  0.72445039,  0.59489476,  0.65904993,  1.8416647 ,
                 0.49705512,  0.36794448,  0.47815699,  0.75358261,  0.08600745,
                 0.58108817,  0.43186646,  1.82221527,  0.34850515,  0.36210523,
                 0.4755703 ,  2.11630464,  1.59907602,  0.14134817,  3.72827411,
                 1.6691565 ,  0.06884024,  0.3489224 ,  1.98527833,  0.85139388,
                 1.09372412,  0.68423446,  1.98711464,  0.14502445,  0.03127346,
                 0.08492938,  1.32845252,  1.9853265 ,  0.49275984,  0.34233787])

In [43]: %matplotlib inline

In [70]: import numpy as np
         import matplotlib.pyplot as plt
         from scipy.stats import gaussian_kde

         m = 10000
         n = 1000 # start at 10 and move up to 10000

         means = np.zeros((m,))

         for i in range(m):
             x = np.random.exponential(size=n)
             means[i] = x.mean()

         density = gaussian_kde(means)
         xs = np.linspace(0.5,1.5,200)
         density.covariance_factor = lambda : .25
         density._compute_covariance()
         plt.plot(xs,density(xs), lw = 2)
         plt.title("Kernel Densit Plot")
         plt.grid(True)
         plt.show()
```



The Monte Carlo Method

Now with that statistical limit theory as a background we can turn our attention to introducting the Monte Carlo Method.
Many quantities of interest in inferential statistical analyses can be expressed as the expectation of a function of a random variable, say $E\{h(Y)\}$. Let f denote the density of Y, and μ denote the expectation of $h(Y)$ with respect to f. When an i.i.d random sample $Y_1, \ldots, Y_n$ is obtained from f, we can approximate μ by a sample average:

$$\hat{\mu}_{MC} = \frac{1}{n}\sum_{i=1}^{n} h(Y_i) \rightarrow \int h(Y)f(Y)dy = \mu$$

as $n \rightarrow \infty$, by the strong law of large numbers. Further, let $v(y) = [h(Y) - \mu]^2$, and assume that $H(Y)^2$ has finite expectation under f. Then the sampling variance of $\hat{\mu}_{MC}$ is $\sigma^2/n = E\{v(Y)/n\}$, where the expectation is taken with respect to f. A similar Monte Carlo approach can be used to estimate σ^2 by

$$v\hat{a}r\{\hat{\mu}_{MC}\} = \frac{1}{n-1}\sum_{i=1}^{n}[h(Y_i) - \hat{\mu}_{MC}]^2$$

When σ^2 exists, the central limit theorem implies that $\hat{\mu}_{MC}$ has an approximate normal distribution for large n, so approximate confidence bounds and statistical inference for μ follow.
Monte Carlo integration offers slower convergence than other numerical methods for integration, so why should we use it? It turns out that Monte Carlo integration is less subject to the curse of dimensionality as other methods, so the Monte Carlo method is especially helpful for multidimensional problems. Also, some methods like quadrature require smoothness, while Monte Carlo ignores smoothness alrogether.

Approximating a Deterministic Definite Integral with the Monte Carlo Method

Let's consider the simple function:

$$f(x) = 4 - 4x^2$$

and the following definite integral:

$$\int_0^1 (4 - 4x^2)dx$$

This is a simple function to integrate and we can use analytical methods from calculus to solve it as follows:

$$\int_0^1 (4 - 4x^2)dx = (4x - \frac{4}{3}x^3) \mid_0^1 = (4(1) - \frac{4}{3}(1)^3) - (4(0) - \frac{4}{3}(0)^3) = \frac{8}{3} \approx 2.6667$$

A useful tool if your calculus skills are rusty is ***Wolfram Alpha***.
To use the Monte Carlo method to solve this integral numerically, we turn it into an expectation over a random variable:

$$\hat{\mu}_{MC} = \frac{1}{n}\sum_{i=1}^{n}(4 - 4X_i^2)$$

where the X_i are random draws from a uniform distribution on the interval $[0, 1]$ for $i = 1, \ldots, n$.
Let's see how we can accomplish this in `Python`:

```
In [39]: import numpy as np

         def f(x):
             y = 4 - 4 * x ** 2
             return y

         def main():
             # Generate random values between 0 and 1
             u = np.random.uniform(0, 1, size = 10**6)

             # Evaluate the function for each value u
             y = f(u)

             # Approximate the integral with the mean
             muhat = y.mean()

             # Print the answer
             print("The Monte Carlo estimate of the integral is: %s" % round(muhat, 4))

         if __name__ == "__main__":
             main()
```

```
The Monte Carlo estimate of the integral is: 2.6674
```

Using the Monte Carlo Method for Option Pricing

We saw that we could we could write the single-period Binomial option pricing model as:

$$C_0 = e^{-rT}[C_u p^* - C_d(1 - p^*)]$$

which we said was the risk-neutral interpretation of the model. We also saw that we could think of the model as containing two terms:

1. An expected cash flow from holding the option at expiration: $[C_u p^* - C_d(1-p^*)]$.
2. A discount factor to bring the expect cash flow to present value: e^{-rT}.

We can write this generically as:

$$C_0 = e^{-rT} E\{CF_T\}$$

where CF_T is the cash flow from holding the option at expiration. As we will see this holds even for more complex price dynamics beyond the Binomial tree. We will see that if we can simulate from a given stock price process (whatever it be), that we can price the option by simulating many stock price paths, applying the option payoff function, taking a sample average and discounting to present value:

$$C_0 = \frac{1}{n} \sum_{i=1}^{n} C_{0,i}$$

where $C_{0,i} = e^{-rT} C_{T,i}$ and $C_{T,i} = \max(S_{T,i} - K, 0)$, and $S_{T,i}$ is the i^{th} terminal simulated stock price for $i = 1, \ldots, n$ for a call option. A put option would be similar, but would apply the put payoff function instead. As we will see this is a very simple way to price options and will apply even for very complex stock price dynamics such as jumps in price and stochastic volatility, which seem to be required by the data.

```
In [ ]:
```