Sampling Estimates of Expectations

Contents


```
import numpy as np
np.set_printoptions(precision=3)
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(rc={"figure.dpi":100, 'savefig.dpi':300})
sns.set_context('notebook')
sns.set_style("ticks")
```

Objectives

- To use the law of large numbers to estimate expectations

The law of large numbers

The strong law of large numbers

Take an infinite series of independent random variables $X_1, X_2, \dots$ with the same distribution, any distribution. Such a sequence of random variables is typically called an iid sequence for *independnet identically distributed*. Let $\mu = \mathbb{E}[X_i]$ be the common mean of these random variables. The *strong law of larger numbers* states the sampling average,

$$\bar{X}_N = \frac{X_1 + \dots X_N}{N} = \frac{1}{N}\sum_{i=1}^N X_i,$$

converges almost surely to μ as the number of samples N goes to infinity. Mathematically, we write:

$$\bar{X}_N = \frac{1}{N}\sum_{i=1}^N X_i \rightarrow \mu \text{ a.s.}$$

The a.s. (almost surely) is a technical term from measure theory which means that the probability of this convergence happening is one.

Demonstration with a synthetic example

Let's demonstrate the law of large numbers. We are going to take a Beta random variable:

$$X \sim \text{Beta}(\alpha, \beta),$$

where α and β is positive numbers. We know that the expectation of the Beta is (see wiki):

$$\mathbb{E}[X] = \frac{\alpha}{\alpha + \beta}.$$

Let's test if the law of large numbers holds:

```
import scipy.stats as st

# Create a Beta:
alpha = 2.0
beta = 3.0
X = st.beta(alpha, beta)

# The number of samples to take
N = 5
x_samples = X.rvs(N)

# Find the real mean (expectation):
mu = X.mean()
# Find the sampling estimate of the mean:
x_bar = x_samples.mean()

# Print the results
print(f"E[X] = {mu:.4f}")
print(f"The law of large numbers with N={N:d} samples estimates it as: {x_bar:.4f}")
```

```
E[X] = 0.4000
The law of large numbers with N=5 samples estimates it as: 0.5245
```

Questions

- Increase the number of samples N until you get closer to the correct answer.

The Monte Carlo method for estimating integrals

Now we will use the strong law of large numbers to estimate integrals. In particular, we will start with this integral:

$$I = \mathbb{E}[g(X)] = \int g(x)p(x)dx,$$

where $X \sim p(x)$ and $g(x)$ is a function of x. Let $X_1, X_2, \dots$ be independent copies of X. Then consider the random variables $Y_1 = g(X_1), Y_2 = g(X_2), \dots$ These random variables are also independent and identically distributed. So, the strong law of large number holds for them and we get that their sampling average converges to their mean:

$$\bar{I}_N = \frac{g(X_1)+\dots+g(X_N)}{N} = \frac{Y_1+\dots+Y_N}{N} \rightarrow I, \text{ a.s.}$$

This is the *Monte Carlo way for estimating integrals*.

Example: 1D expectation

Let's try it out with a test function in 1D (Example 3.4 of Robert & Casella (2004)). Assume that $X \sim \mathcal{U}([0,1])$ and pick:

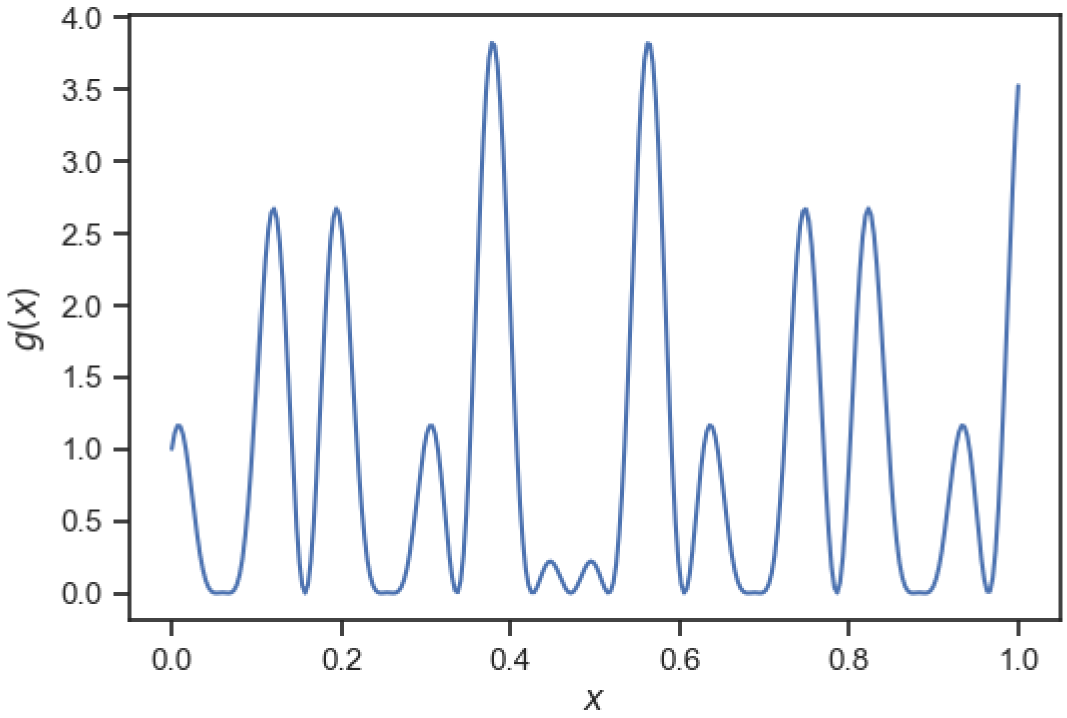
$$g(x) = (\cos(50x) + \sin(20x))^2.$$

The correct value for the expectation can be found analytically and it is:

$$\mathbb{E}[g(x)] = 0.965.$$

```
# Define the function
g = lambda x: (np.cos(50 * x) + np.sin(20 * x)) ** 2

# Let's visualize is first
fig, ax = plt.subplots()
x = np.linspace(0, 1, 300)
ax.plot(x, g(x))
ax.set_xlabel(r"$x$")
ax.set_ylabel(r"$g(x)$");
```



Let's take the samples:

```
N = 100
x_samples = np.random.rand(N)
y_samples = g(x_samples)
```

Evaluate the sample average for all sample sizes (see cumsum).

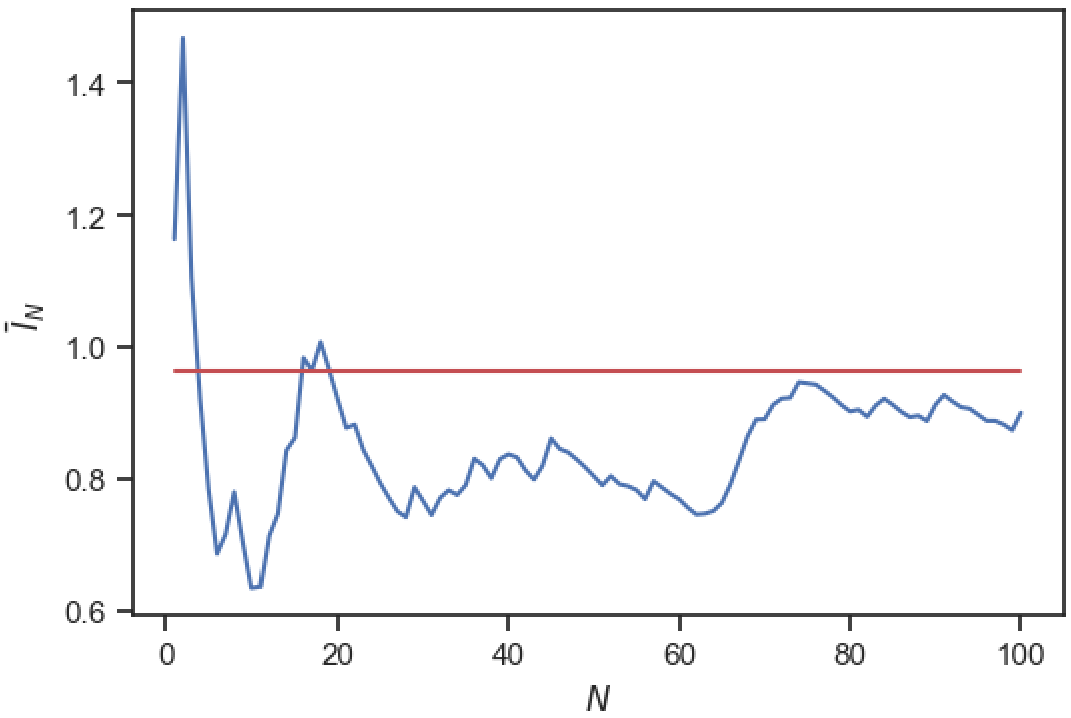
```
I_running = np.cumsum(y_samples) / np.arange(1, N + 1)
I_running
```

```
array([1.164, 1.468, 1.107, 0.923, 0.785, 0.687, 0.717, 0.781, 0.706,
       0.635, 0.637, 0.715, 0.748, 0.844, 0.863, 0.984, 0.965, 1.008,
       0.966, 0.921, 0.878, 0.883, 0.845, 0.82 , 0.794, 0.772, 0.752,
       0.743, 0.788, 0.767, 0.746, 0.772, 0.784, 0.776, 0.791, 0.831,
       0.821, 0.802, 0.831, 0.838, 0.833, 0.814, 0.8  , 0.821, 0.862,
       0.846, 0.841, 0.83 , 0.818, 0.805, 0.791, 0.805, 0.793, 0.79 ,
       0.784, 0.77 , 0.797, 0.788, 0.778, 0.77 , 0.757, 0.747, 0.748,
       0.752, 0.765, 0.793, 0.828, 0.865, 0.891, 0.891, 0.913, 0.923,
       0.924, 0.947, 0.946, 0.944, 0.934, 0.925, 0.913, 0.903, 0.906,
       0.895, 0.912, 0.922, 0.913, 0.903, 0.894, 0.897, 0.888, 0.913,
       0.928, 0.918, 0.91 , 0.907, 0.898, 0.889, 0.889, 0.883, 0.875,
       0.901])
```

Plot this:

```
fig, ax = plt.subplots()
ax.plot(np.arange(1, N+1), I_running)
ax.plot(np.arange(1, N+1), [0.965] * N, color="r")
ax.set_xlabel(r"$N$")
ax.set_ylabel(r"$\bar{I}_N$");
```

syntax for plotting at 0.965 for each discrete x-location?



Objective is to achieve convergence as cumulative sum consists of more samples from the rv

Questions

- Increase `N` until you get an answer that is close enough to the correct answer (the red line).
- Reduce `N` back to a small number, say 1,000. Run the code 2-3 times to observe that every time you get a slightly different answer...

By Ilias Bilionis (ibilion[at]purdue.edu)