IE531: Algorithms for Data Analytics
Spring, 2019
Programming Assignment 4: Working in an Unfair-World!
Due Date: March 20, 2020

1 Introduction

We have an unknown, possibly unfair, three-sided dice (cf. figure 1). You get one of three outcomes (say, $x_i \in \{1, 2, 3\}$) when you roll this dice. We have no clue as to what $Prob(x_i = 1), Prob(x_i = 2)$, and $Prob(x_i = 3)$ are. All we have to go on is that $Prob(x_i = 1) + Prob(x_i = 2) + Prob(x_i = 3) = 1$. We also know that by repeated tosses of this dice we can generate a string of i.i.d. outcomes $\{x_i\}_{i=1}^{\infty}$. Empirically estimating these probabilities are ruled-out, as there is no way to estimate the individual probabilities with *certainty* using a finite-set of outcomes.

We are asked to use this unknown, possibly unfair, three-sided dice as the only available source of "randomness," and to generate continuous-RVs that are from a unit-normal distribution. That is, we have to generate a set of Univariate Gaussian RVs $\{y_i\}_{i=1}^{\infty}$ such that $y_i \sim N(0, 1)$.

Figure 1: Three-sided-dice.

2 Discussion

We know that if we have a way of generating u.i.i.d. RVs we can generate the Unit-Normal RVs using the *Box-Muller Transform*. **We are not permitted to use** np.random.uniform() **for this part of the exercise. All we have is the unknown, possibly unfair, three-sided dice.** Essentially, we have to find a way of "converting" the outcomes of repeated-tosses of this three-sided dice into u.i.i.d. RVs.

See lectures in Echo360.org for the process of simulating a fair-coin (i.e. simulating a coin whose probability of "heads" and "tails" being equal) using

the unknown, possibly unfair, coin. You have to figure out how to modify this to handle the three-sided dice, in class. This method is an illustration of a large-class of "accept/reject" algorithms that find use in different forms-and-shapes in data analytics. The resulting simulated-fair-coin is used to generate u.i.i.d. RVs. Following this, the rest of the programming exercise should be straightforward.

3 Requirements

What I need from you:

1. Suppose $Prob(1) = p_1, Prob(2) = p_2$ and $Prob(3) = p_3$ are the probabilities of seeing a "1" or "2" or "3" when the (possibly unfair) 3-sided dice is tossed – I want your Python code to set these probabilities such that the 3-tuple (p_1, p_2, p_3) is uniformly distributed over the surface $p_1+p_2+p_3 = 1$ where $p_1, p2, p3 \geq 0$. That is, you should have an equal chance of picking any (p_1, p_2, p_3) from the surface $p_1 + p_2 + p_3 = 1$ where $p_1, p2, p3 \geq 0$ as the probabilities for the unfair 3-sided dice.

2. Python Code that uses a stream of i.i.d. discrete RVs $\{x_i\}_{i=1}^{\infty}$, where $x_i \in \{1, 2, 3\}$, where $Prob(x_i = 1) = p_1, Prob(x_i = 2) = p_2$, and $Prob(x_i = 3) = (1 - p_1 - p_2)$, where p_1 and p_2 are not known (the values of p_1 and p_2 are assigned randomly, as described above). Your code should produce

   (a) an unit-normal RV generator.

   Make sure your code is commented well enough to permit quick grading.

These files can be submitted on Compass directly. Please do not e-mail them to the TA or me.