# HomeWork 3 : Simple ( ?) Sampling Problems

Most of this homework has to be done on a computer, with the language of your choice. Present your result with graphs, plots, and data.

## 1 Sampling discrete distributions

We shall need to build a discrete distribution with $n_{item}$ elements, each of them having a probability randomly chosen. At first, we shall use a uniform distribution as follows : **First** : we create $n_{item}$ random number $r_i$, with $i = 0, \ldots, n_{items} - 1$,from a uniform random distribution between 0 and 1. **Then** we compute the sum $Z = \sum_i r_i$ of these numbers. **Finally** we write that $p_i = r_i/Z$. Now, we have made a distribution where each number $i$ from 0 to $n_{item} - 1$ has a probability $p_i$.

1. Consider $n_{item} = 5$. Write the "accept/reject" procedure to sample from the distribution. Sample many elements, make an histogram of the sampled data and compare with the theoretical proportions.
2. Repeat now this exercise with the "Tower sampling" procedure.
3. Now we wish to estimate numerically the time used by these two procedures. For many different values of $n_{item}$, compute the time needed to sample 100000 elements. How does this time grows with $n_{item}$ in the case of the "accept/reject" procedure ? And for the "Tower sampling" procedure ? Can you explain why ? As $n_{item}$ grows which one seem to be the fastest way to solve the problem ?
4. Repeat this exercise but now sample the $r_i$ from an exponential distribution (this can be done by writing $r_i = -\log random.uniform$). How does this time grows with $n_{item}$ in the case of the "accept/reject" procedure ? Why the difference [1] ?
5. Repeat this exercise but now sample the $r_i$ by writing $r_i = random.uniform^{-1/2}$ (or the equivalent formulation in your language). From which distribution are we now sampling ? Compute numerically how the time now grows with $n_{item}$ in the case of the "accept/reject" procedure ?
6. What do you conclude on these two methods to sample discrete distributions ?

## 2 Computing integrals

Consider the function $f(x) = 1 - e^{-x}$. We can to compute the one-dimensional integral $\int_0^1 f(x)dx$ exactly, but here we want to do it numerically.

1. First, we proceed much in the same way as when computed $\pi$ : we simply draw uniformly numbers in the rectangle $(0, 0), (1, 1 - e^{-1})$ and count the proportion of points below $f(x)$. Show how this allows to estimate the integrals (and an estimate of the error), and try it for different values of the numbers of points $N$.
2. Instead of drawing points uniformly in the rectangle, we can notice that $f(x) \leq x$. Using the transformation or the cumulative method show how to sample from $p_X(x) = 2x$ in $[0, 1[$. Explain why sampling $x$ from $p_X(x)$ and then $y$ uniformly between $[0, cp_X(x[$ (with $c = 2$) actually generate points with a uniform distribution under the line $x$. Use this to again compute the integral $\int_0^1 f(x)dx$ for different values of $N$. Why is the error lower (on average) ?

1. Clue : the largest number obtained by sampling $n$ number from an exponential distribution is $O(\log n)$.

3. Show that $f(x) \geq g(x) = (1 - exp(-1)) * x^{1/2}$. We now want to generate points filling uniformly the area below $g(x)$. Explain how you can easily generate random numbers $x$ from $p_x(x) = \frac{3}{2}\sqrt{x}$ and compute $c$ such that $g(x) = c * p_x(x)$. Show how to use this to improve again on the estimation of the integral.

This was a trivial example, but this method can be applied to much more involved computation. For instance, show that one can use it to compute the following integral :

$$I = \int dxdydz \mathrm{abs}(\cos(\sqrt{\mathrm{x}^2 + \mathrm{y}^4}))\tanh(\mathrm{x}^2 + \mathrm{y}^2 + \mathrm{z}^4)\frac{\mathrm{e}^{-\mathrm{x}^2/2 - \mathrm{y}^2/2 - \mathrm{z}^2/2}}{(2\pi)^{3/2}}$$

# 3 Sampling $\pi$ with MCMC

We come back to the problem dicussed in HW2 : estimating $\pi$ by sampling uniformly points in the square between $-1 < x < 1$ and $-1 < y < 1$, and counting how many hits are inside the unit circle. For each points $i = 1 \ldots N$, we again define the random variable $S_i = 0$ if the point is outside the circle, and $S_i = 4$ if the point is inside the circle.

As oppose to the previous HW, however, we now use the Markov Chain strategy to generate our $N$ points. The first one is at $(x_1 = 0, y_1 = 0)$ and each subsequent ones is then selected as follows : we propose a move $\delta_x = random.uniform(-stepsize, stepsize)$ and $\delta_y = random.uniform(-stepsize, stepsize)$ with $stepsize = 0.1$. Following the metropolis rule we discssed in class, the new position for the point $i + 1$ is then $(x_{i+1} = x_i + \delta_x, y_{i+1} = y_i + \delta_y)$ if it is inside the square, and $(x_{i+1} = x_i, y_{i+1} = y_i)$ otherwise.

1. Implement the MCMC and make a plot after 1000, 4000 and 20000 steps. Comments ?
2. We should think if $stepsize = 0.1$ is really a good choise. A good rule of thumb to select the *stepsize* is the empirical 1/2 rule : one should select it such that the accepting probability of the move, on a long run, is about 1/2. After few trials, you should be able to find the optimal value of *stepsize*. Repeat the plots of Q1 using the new "optimal" value.
3. We again consider the estimator $\hat{m} = S_i/N$ Is it still true that the error made by this estimator is the one given in HW2 (Q1.3) ? Compare the error made by your program with the one predicted by this formula.

Practitioners of MCMC have a way to solve this problem by using a "batch" method, also called the bunching method. To estimate the error of a Markov-chain simulation that has obtained many observables $S_1, S_2, \ldots, S_N$, we should do a "bunching" of the data, and work with a new set of observables defined as $(S_1 + S_2)/2, (S_3 + S_4)/2, \ldots, (S_{N-1} + S_N)/2$. This gives a new chain of data, with new observables (that are simply the mean value of two subsequent old ones), a new number of data points (that is simply half the old one), an unchanged mean value, but a new value of the variance and therefore a new estimate of the error, if using the naive formula of Q1.3 in HW2 to these points.

This bunching procedure can then be iterated, making pairs, then pairs of pairs, then pairs of pairs of pairs. The observed error is found to increase with the iterations, then exhibits a plateau (to see this you may have to run the program several times if you are unlucky). This plateau is an *excellent* estimation of the true error of a single Markov chain output.

1. Implement the bunching strategy to analyze your data and the predicted error, with both the optimal value of *stepsize* and the 0.1 one, and check that it predicts the error accurately.
2. Why does this procedure work according to you ? Gives a hand-waving argument explaining why the apparent error initially increases with the iterations, then saturates (more or less) to a plateau.