Monte Carlo Methods, Courant Institute, Spring 2015

http://www.math.nyu.edu/faculty/goodman/teaching/MonteCarlo15/

Always check the class message board on the NYU Classes site from home.nyu.edu before doing any work on the assignment.

Assignment 1. Part 1 due due February 10, Part 2 due February 17

Corrections: none yet.

Part 1: Set up a computing environment for the class. Programming assignments will use a combination of C/C++ and Python 2.7. The C/C++ code will do the main Monte Carlo computation. The Python code will do data analysis and visualization. The C/C++ code will write an ASCII format output file that contains all the information about the run that the analysis and visualization routines need. This file will be in the form of a Python module that can be read by the Python routines we create. This arrangement takes advantage of the strengths of both language systems. The C/C++ language and compilers create efficient code. Python comes with visualization tools and has more "productivity" (lines of debugged code per hour of programmer time).

Computing assignments in this class will ask students to program in both C++ and Python. We will use only basic features of these languages. There are excellent web resources to help you do this. You will have to learn enough about *make* to add new procedures to the `CPP_SOURCES` list. If you get stuck, post a question on the class message board.

(a) Setup a UNIX like programming environment. If you have a Mac or a Linux box, this just means using the terminal command line. If you have a Windows operating system, you can get a UNIX like operating system using cygwin. This is available for free download, but it's better to avoid cygwin if you have a choice. In your UNIX like environment, download the file `assignment_1.tar` to some directory. In that directory, type:

```
tar -xvf assignment_1.tar
```

Some files should appear. If this worked, the file `assignment_1.pdf` will look the same as `assignment_1_check.pdf` on the class web site.

(b) The C/C++ compilers come pre-installed on Linux and systems. On OSX (the Mac operating systems), they are part of the `xcode` package that you can download from Apple. You can check that your C/C++ compilers work by typing

```
make fTestExecutable
```

(c) There are several ways to get Python 2.7. It is installed on CIMS desktop machines, but if you just type `python ...`, you might get Python 2.6. You get version 2.7 by typing: `module load python-2.7`. Other systems come with installed versions of Python2.7 that may not be suitable for this class because they do not have the packages `numpy` and `matplotlib`. You can get a full Python installation, with these and many other packages, from `anaconda` (google it) or (what I use) `homebrew`. It has been frustrating for me to get Python working. Please report your frustrations on the class message board so others can help you get through this. If your Python is installed correctly, you should be able to type

```
python histogram.py
```

and create a .pdf file identical to the `assignment_1.pdf` that came in the *tarball* file `assignment_1.tar`.

(d) The final step is editing. We do not want to use an integrated build system such as the one that comes with `xcode` or the Microsoft compiler suite. These integrated systems are so different from each other that it would be impractical to post code in all formats. Instead, we use the UNIX `make` system. You need to find an editor you like. Popular choices are `emacs`, `xedit` (on Linux or cygwin systems), and the `xcode` editors on OSX. Check that you can edit by playing with the program. Change the bin size of the number of samples to see what happens. These are set in `main.cpp`. Change the wording of the plot title a little. This is set in `histogram.py`. Type

```
make fTest
```

to build and run the whole thing. The power of the `make` system is that you tell it how to build anything and it can re-build and re-run everything necessary from just one `make` command. Hand in two plots that show you have edited both `main.cpp` and `histogram.py`. Choose parameters you think are interesting.

Part 2: The actual assignment.

1. (*This exercise has two purposes. One is to understand why a sampler might not work well in high dimensions. Another is to understand why some functions have good Gaussian approximations. The analytical method is called Laplace's method.*) Let C_n be the cube in n dimensions, symmetric about the origin, whose side is length 2. This may be written $C_n = [-1,1]^n$. The n dimensional volume of C_n is $\text{vol}_n(C_n) = 2^n$. If $x \in \mathbb{R}^n$, then $x \in C_n$ if $|x_k| \leq 1$ for all k. You can generate a "random" point $X \in C_n$ by taking $X_k = 2U_k - 1$, where the U_k are independent standard uniforms. This X has uniform probability density inside C_n, which is $f(x) = 2^{-n} = 1/\text{vol}_n(C_n)$ if $x \in C_n$, and $f(x) = 0$ if $x \notin C_n$. Let B_n be the unit ball in n dimensions. We have $x \in B_n$ if $(x_1^2 + \cdots + x_n^2)^{1/2} \leq 1$.

Clearly $B_n \subset C_n$. We can generate X uniformly distributed in B_n by generating X uniformly distributed in C_n and accepting it if $X \in B_n$. The efficiency of this algorithm is the ratio of the volumes

$$Z_n = \frac{\mathrm{vol}_n(B_n)}{\mathrm{vol}_n(C_n)} \ .$$

This exercise derives an approximate formula for Z_n. The formula shows that shows that $Z_n \to 0$ as $n \to \infty$, exponentially. Therefore the sampling method impractical for large n. An exercise from the Week 1 notes suggests a different sampler that is practical for large n.

It is possible to find the large n behavior of $I(n)$ in (1) using a change of variables $r^2/2 = s$ to express it in terms of the Γ function, whose asymptotics are available on wikepedia – *Stirling's formula*. Please don't do it this way. The asymptotics of Γ are found using the method of this problem, so that approach is not actually easier.

(a) The *unit sphere* in n dimensions is $S_{n-1} = \{|x| = 1\}$. The "surface area" (or $n-1$ dimensional volume) of S_{n-1} is ω_{n-1}. Show that

$$\mathrm{vol}(B_n) = \frac{\omega_{n-1}}{n} \ .$$

You can do this by

$$\mathrm{vol}(B_n) = \int_{x \in B_n} dx$$

using polar coordinates, which involves $\omega_{n-1} r^{n-1} dr$.

(b) Show that

$$\omega_{n-1} = \frac{(2\pi)^{n/2}}{I(n)} \ ,$$

where

$$I(n) = \int_0^\infty r^{n-1} e^{-r^2/2} \, dr \ . \tag{1}$$

Hint: integrate

$$(2\pi)^{n/2} = \int_{x \in \mathbb{R}^n} e^{-|x|^2/2} \, dx$$

in polar coordinates.

(c) Write $I(n) = \int e^{-\phi(r)} \, dr$ and identify ϕ. Show that ϕ has a unique maximum value achieved at r_*. Calculate $\phi''(r_*)$, $\phi'''(r_*)$, and possibly one more. Let $q(r)$ be the quadratic Taylor approximation to $\phi(r)$ about r_*, which is

$$q(r) = \phi(r_*) + \tfrac{1}{2}\phi''(r_*)(r - r_*)^2 \ . \tag{2}$$

Write the formula for

$$J(n) = \int_{-\infty}^\infty e^{-q(r)} \, dr \ .$$

(d) $J(n)$ is an approximation of $I(n)$. The error is written $K(n) = I(n) - J(n)$. Show that

$$\frac{K(n)}{I(n)} \to 0 \quad \text{as} \quad n \to \infty \; .$$

Hint: there are two kinds of r values: those where the quadratic approximation (2) is accurate, and those where ϕ and q are much smaller than values that matter. For this exercise, you can take the "values that don't matter" set to be $|r - r_*| > n^p$ with $0 < p < \frac{1}{6}$. When $|r - r_*| = n^p$, then e^{-q} does not matter, and $q(r)$ is still relatively close to $\phi(r)$ (use ϕ''' to verify this).

(e) Write the large n asymptotic approximation of Z_n that shows that sampling uniformly in the ball by rejection from the cube is an exponentially bad idea.

2. (*Probability distributions usually depend on parameters. It may not be enough that a sampler "works" for each parameter value. It may need to be efficient uniformly over the parameter. This Exercise is an example of such a sampler. This exercise also demonstrates that some careful analysis can lead to good samplers.*) Let S_n for $n = 0, 1, \ldots$, be independent exponential random variables with rate parameter λ. Let these be the *inter-arrival* times for the arrival times T_n, which means that $T_0 = S_0$, and $T_n = T_{n-1} + S_n$ for $n > 0$. The sequence T_n is a *Poisson process* with *arrival rate* parameter λ. The goal is to find a sampler that samples T_n using an amount of work that is bounded as $n \to \infty$. A direct simulation of the Poisson process takes order n work, because you have to generate all the inter-arrival times from S_0 up to S_n. For simplicity, we take $\lambda = 1$.

(a) Show that the probability density for T_n is $f_n(t) = \frac{t^n}{n!}e^{-t}$ if $t \geq 0$. Hint: $T_n = T_{n-1} + S_n$, with S_n independent of T_{n-1}, allows you to find f_n from f_{n-1}.

(b) Determine the behavior of $f_n(t)$ for typical T_n values using the method of Exercise 1. Find the most likely value of T_n by maximizing f_n, then make a Gaussian approximation of f_n about this value, t_{n*}.

(c) You can find the mean and variance of T_n from the representation of T_n as a sum of independent S_k for $k \leq n$. You can estimate the mean and variance of T_n from the Gaussian approximation of part (2b). Show that these ways of getting the mean and variance give (approximately?) the same result.

(d) Explain why it is not a good idea to use the Gaussian approximation as a proposal distribution for rejection sampling of f_n.

(e) Explore using a *double exponential* as a proposal distribution. That is $g_n(t) = \frac{1}{Z}e^{-\alpha_n|t-t_{n*}|}$. Calculate the normalization constant Z. To find the optimal Z you need to solve the two maximization problems, one for $t > t_{n*}$ and one for $t < t_{n*}$. Do not worry about negative T values. Those are rare for large n, and can be rejected for any n.

(f) What formula for α_n is suggested by the Gaussian approximation? You can choose α_n so that the proposal distribution has the same or similar variance as the true distribution.

(g) Determine wether this α_n leads to a sampler whose efficiency does not go to zero as $n \to \infty$. If so, you are done. If not, can you adjust α_n to make the sampler uniformly efficient?

3. (*Programming exercise. Please read the material on the class web site on programming conventions. When you modify and re-use posted code, please keep the automation features, such as making plots automatically with computational parameters and legends. If you add a computational parameter, figure out how to make it appear in the plot. If you remove a parameter, make it disappear from the plot. Update the* `makefile` *to keep everything automated.*) Modify the code to sample the density $f(x) = \frac{\pi}{2}\sin(\pi x)$ for $x \in [0,1]$, and $f(x) = 0$ otherwise. Use rejection sampling with proposal distribution $g(x) = 6x(1-x)$ as described in the notes. Here is a suggested sequence of steps.

   (a) The proposal distribution is sampled using procedures presently in the file `f.cpp`. You need to copy this to `g.cpp` and change the names of the routines to be g instead of f. It should be clear how to do this. You also need to adjust `header.h`. If you do this correctly and run the code again, you should get the same plot, except that it will be called g.

   (b) Now modify `f.cpp` to do the rejection sampling using g as a trial. Change everything that needs changing, including the string that describes the distribution. Test it using the histogram procedure. Put both the f and g target distribution curves in the plot, so you can see that you have changed from f to g. Use a sample size that makes it clear that the empirical histogram represents f, not g.