PHYS332: Homework 3 (due Sep 17, 9:30am)

Monte Carlo Integration

Note: Please read the complete homework instructions before you start.

Goal: The goal of the next three homeworks (starting with this) is to develop the necessary machinery for Monte Carlo integration (MCI). MCI in its simplest form can be described as finding the integral

$$\Phi_f = \int_a^b f(x)\,dx \tag{1}$$

by taking the average (or "calculating the expectation value $E[\phi]$")

$$\hat{\Phi}_f = \frac{b-a}{R}\sum_{r=1}^{R} f(x_r), \tag{2}$$

where $x_r \in [a,b]$. Writing $\Delta x \equiv (b-a)/R$, you can see why eq. 2 should converge to eq. 1 in the limit of $R \to \infty$.

Main Challenge: The main issue to understand is that eq. 2 can be *extremely* inefficient. Imagine, $f(x) = \exp(-(100x)^2)$ with $[a,b] = [-1,1]$. Most of the support points x_r will not contribute to the sum $\hat{\Phi}_f$: their calculation is a waste of time. How can we position (i.e. "sample") the x-axis cleverly, such that most of the x_r contribute significantly to $\hat{\Phi}_f$?

Answer: The goal of Monte Carlo methods is twofold[1]:

- to generate samples $\{x_r\}_{r=1}^R$ from a given **probability density function (PDF)** $P(x)$. An example for $P(x)$ would be the PDF[2] for the normal distribution, $P(x) = \exp(-x^2/2)/\sqrt{2\pi}$. In the main challenge as formulated above, this would address only part of the problem, namely how to cleverly place the x_r. Yet, this is a important application by itself – many scientifically relevant probability distributions can only be calculated efficiently by "clever" sampling. We will meet two methods that can achieve this goal, the **rejection method** (this homework), and the **Metropolis-Hastings algorithm** (next homework).

- to estimate the expectation values of a function $\phi(x)$ under the PDF $P(x)$,

$$\Phi_P = \langle\phi(x)\rangle_P \equiv \int P(x)\phi(x)\,dx. \tag{3}$$

  $P(x)$ can be as simple as the uniform PDF, $P(x) = 1/(b-a)\,\forall\,\{x|x \in [a,b]\}$, and 0 otherwise, in which case we recover eq. 2. The reason to introduce a more complicated $P(x)$ is the same as for sample generation: we want the integrand to be evaluated at "important" positions, hence the method doing this is called **importance sampling**.

[1] I strongly recommmend `http://jimbeck.caltech.edu/summerlectures/references/Stochastic%20simulation.pdf`

[2] Note the somewhat cumbersome language: It is useful to distinguish between the **PDF** and its (associated) **probability distribution**: The PDF is given by a function $P(x)$, while its distribution is a set of x_r. The literature sometimes uses shorthand language for this, not necessarily making things clearer.

Another way to see this is if we solve the sampling problem, we have found the "clever" choice for $\{x_r\}$ in the integration problem.

The Strength of Monte Carlo – Convergence and Variance: By the Law of Large Numbers, the average

$$\hat{\Phi} \equiv \frac{1}{R} \sum_r \phi(x_r) \xrightarrow[R\to\infty]{} \Phi, \tag{4}$$

if the x_r are drawn from $P(x)$, and by the Central Limit Theorem, the variance of $\hat{\Phi} \propto \sigma^2/R$, where

$$\sigma^2 = \int P(x)(\phi(x) - \Phi)^2 \, dx \tag{5}$$

is the variance of the function ϕ. The strength of MCI lies in the fact that eq. 5 holds *independently of the number of dimensions in the problem*, i.e. a single x_r could be a vector in $\mathbb{R}^N, N = 1000$. Yet, in this multi-dimensionality lies the problem: most often, such systems have $\phi(x)$ concentrated in a very small region of $\mathbb{R}^N$, therefore it is crucial to find "clever" choices of $\{x_r\}$ **and** to do so efficiently. The main answer to this problem is the Metropolis-Hastings algorithm and its relatives.

Rejection Sampling: Let's meet our first Monte Carlo technique. Imagine you would like your random support points $\{x_r\}$ to be distributed according to a Gaussian[3] centered on $x = 0$, with a width of $\sigma = 1$. Thus, the corresponding PDF that generates this distribution (also known as the normal distribution) is given by

$$N(\mu, \sigma) \overset{\mu=0,\sigma=1}{=} N(0,1) \equiv \frac{1}{\sqrt{2\pi}} e^{-x^2/2}. \tag{6}$$

Clearly, this would mean that within equally sized bins of Δx, more x_r should be found close to $x = 0$ than – say – close to $x = 10$. That's the whole point: putting the support points where they matter.

(3a) Normal distribution [5 pts]: Show that for $\{x_r\}_{r=1}^R$ drawn from $N(0,1)$, $\approx 68\%$ of the samples are located within $-1 \leq x \leq 1$, and $\approx 95\%$ are located within $-2 \leq x \leq 2$. *Hint: You can use Mathematica or similar tools – just remember to normalize your Gaussian appropriately.*

(3b) The algorithm [5 pts]: Rejection sampling does the following: Assuming that the **target density function**[4] $P(x)$ is too complicated to sample the x_r from directly, we find a **proposal distribution function** $Q(x)$ and a constant c such that $\forall x : cQ(x) > P(x)$. In the simplest case, $Q(x) = U(a,b)$, resulting in uniformly distributed x on the interval $a \leq x \leq b$, and $c = \max(P/Q)$. It is crucial to ensure that $\forall x : cQ(x) > P(x)$. Note that in Bayesian statistics, $Q(x)$ would provide the *prior* (i.e. the known, "easy to calculate" distribution), and $P(x)$ would be the *posterior* (i.e. generally unknown).

Now, generate two random numbers. The first, x, is drawn from $Q(x)$ (again, in the simplest case, uniformly distributed between $a \leq x \leq b$). Evaluate $Q(x)$. Then, pick a uniformly distributed

[3] In this case, the x_r can be directly determined, see *Box-Mueller method*, but we assume we don't know this.

[4] Note the subtlety in language. The *density function* $P(x)$ is the function that generates the *distribution* (of random variables) x. $P(x)$ is also called *probability density function*, or PDF.

random variable $0 \leq u \leq 1$. If $u \leq P(x)/(cQ(x))$, accept x as a new x_r, otherwise discard it. Repeat this process until you have R random numbers.

Show (analytically) for the simplest case of constant $Q(x)$, that the number of accepted x's divided by the number of total attempts, N_R/N_{tot} is a measure of the integral $\int_a^b P(x)\,dx$, i.e. this is a Monte Carlo integration method.

(3c) Implementation and test [20 pts]: Implement the algorithm described under (3b) in the routine `reject` as defined in `mci_reject.py` and test it with the functions

$$P_1(x) = \frac{\exp(x)}{\exp(2)-\exp(-4)} \quad -4 \leq x \leq 2 \tag{7}$$

$$P_2(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) \quad -4 \leq x \leq 4 \tag{8}$$

(i.e. run `mci_reject.py exp uniform 1000` and `mci_reject.py normal uniform 1000`). Your test should print out the analytical value and compare it to the numerical result, and it should show a histogram of x_r, overplotted with $P(x)$ and $Q(x)$. If everything works, the histogram should trace $P(x)$. Typical values for the sample length $R = 100, 1000, 10000$. Your program should print out the following information:
– the ratio of accepted over total tries, and
– the mean and the variance of the distribution $\{x_r\}$.
Do mean and variance change with the number of tries?

(3d) Non-uniform priors: transformation of probabilities [5 pts]: The uniform distribution is in most cases a pretty inefficient prior, which results in a large number of rejected attempts. Here, we'll step up the game and generate normally distributed x_r using the Cauchy (for mathematicians) or Lorentzian (for physicists) PDF,

$$Q(x) = \frac{1}{\pi(1+x^2)}. \tag{9}$$

You will need to draw random variables from $Q(x)$. Show analytically that the prescription

$$x_Q = \tan\left(\pi\left(u - \frac{1}{2}\right)\right), \tag{10}$$

with the random variable u drawn from the uniform distribution $U(0,1)$, achieves this goal.

(3e) Non-uniform priors: Implementation [15 pts]: Implement sampling from the Lorentzian distribution (eq. 9), and use it to draw $\{x_r\}$ from the normal distribution $N(0,1)$ via the rejection method. Remember that you need to find a constant c such that $\forall x: cQ(x) \geq N(0,1)$. As before, plot the resulting histogram of $\{x_r\}$, and $N(0,1)$ and $cQ(x)$. Compare the mean and the variance with your results from (3c) for $N(0,1)$.

Please note that the only task you have to do in terms of implementation is to code up `reject`. *All function definitions etc have been included already in* `mci_reject.py`.