PHYS358: Session 08, Group B

Monte Carlo Integration (2): Importance Sampling

Here, you'll meet you first true MC integration technique. We already discussed rejection sampling, which solves the first problem of MC, namely how to sample a (nearly) arbitrary probability density function $p(x)$, i.e. how to draw random numbers distributed according to $p(x)$.

Now, we move on to importance sampling. The problem is this: You would like to integrate a function $f(x)$,

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

$$\approx \frac{1}{N}\sum_{i=0}^{N-1} f(x_i) \quad \text{with } x_i \sim U(a,b). \tag{2}$$

Now imagine that $f(x)$ is a function with very small support - strongly spiked around 0, for example. Clearly, most of the support points won't contribute to the integral. Can we place the support points such that they "count"?

Yes, we can, of course. But how does that affect the integral? To see this, just check the above equations again. Remember that the probability density function for the uniform distribution $U(a,b)$ is given by

$$p(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

In other words, when making the transition from the integral Φ to the sum $\hat{\Phi}$, we divide by $p(x)$ and sample x from $U(a,b)$.

This approach can be generalized to any probability density function:

$$\int f(x)dx \;\; x \text{ drawn from some distribution} \tag{4}$$

$$= \int \frac{f(x)}{p(x)}p(x)dx \tag{5}$$

$$= \int \frac{f(y)}{p(y)}dy \;\; \text{where } y \text{ is drawn from } p. \tag{6}$$

In other words, we can draw from a different distribution to move our support points to where it matters, but we have to "correct" by dividing by the corresponding density function evaluated at those support points.

Let's see how this works:

1. We will test importance sampling with a Gaussian :

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}. \tag{7}$$

   What's the value of the integral of $f(x)$ on the interval $-\infty < x < \infty$?

2. Run `mci_importance.py -h` to understand the call sequence in the next step.

3. Run `mci_importance.py gaussian 0 uniform 100 1`. This will integrate the Gaussian using a uniform "prior" (or sampling density), with 100 support points. Identify the lines on the plot, and write down the error σ. Does the expectation value μ (i.e. the estimator for the integral Φ) agree with your analytical result?

4. Try the above again, now with 1000 support points. Does σ change? Why (not)?

The following is often referred to as **variance reduction techniques**. Beyond the brute-force approach to MC integration outlined above, this is pretty much the main thrust of MC methods development.

1. We already know one way to decrease the variance: repeat the experiment over and over again (Central Limit Theorem!). I suggest to reduce the support points to 100 again, and attempt 100 repetitions, i.e. `mci_importance.py gaussian 0 uniform 100 100`. You should see three plots: the mean and variance against the iteration number, and the resulting distribution of the mean. Clearly, increasing the sample size is a way to reduce the variance.

2. How about "clever sampling", i.e. **importance sampling**? We need a weight function $p(x)$ placing support points where $f(x)$ is large, i.e. around 0. One example would be the Lorentz/Cauchy distribution:

$$p(x) = \frac{1}{\pi(1 + x^2)} \tag{8}$$

   Run the program replacing the uniform distribution with they Cauchy distribution: `mci_importance.py gaussian 0 cauchy 100 1`. Compare your results (variance!) with the results for the uniform distribution. By what factor does the variance decrease?

3. Of course, we can also increase the sample size for our improved integration: `mci_importance.py gaussian 0 cauchy 100 100`. Note how the variance now decreases smoothly with increasing sample size.

4. One key element in this game of importance sampling is to find a "good" sampling distribution $p(x)$. For example, we could have chosen a Gaussian instead of the Cauchy distribution for sampling. You can try this by `mci_importance.py gaussian 0 normal 100 1`. You can repeat this several times. Keep your eyes on the variance, and explain the results.