Physics 468: Computing Project 2

January 13, 2012

Computing integrals with a Monte Carlo procedure

Integrals are hard. Multidimensional integrals, such as those encountered in studying multi-particle wavefunctions, are *really hard*. (as you will no doubt discover while you attempt to complete HW 5.11). There are many tricks and traps associated with evaluating these hard integrals (and I will show you a few before we're through!) but the one that this project involves is a strategy of numerical computation. The term *Monte Carlo* comes from the name of a city in the small sovereign principality of Monaco where they do a *lot* of gambling. Of course games of chance involve randomness and so algorithms that require random numbers are often called *Monte Carlo algorithms*. This project will give you a little experience with a Monte Carlo algorithm for evaluating integrals.

Random numbers

Random numbers generated on a computer are of course *not* really random. They are pseudo-random! For all practical purposes that means they are "random enough". The random numbers produced by the *random* function are created by an algorithm known as the *Mersenne Twister*. It's all described here:

`http://docs.scipy.org/doc/numpy/reference/routines.random.html`

For our purposes we just need a way to generate pseudo-random numbers in the half open interval $[0.0, 1.0)$ (i.e., including 0.0, but not including 1.0).

OK.. so how do we compute integrals with random numbers?

Suppose we desire to compute the 1-D integral

$$I = \int_0^1 f(x)dx \tag{1}$$

Where $f(x)$ is some well known function like $f(x) = x(1-x)$. We could naturally break out Simpson's rule, or the trapezoidal rule (or we could just do it analytically), but let's say we want to do it purely with random numbers.

$$I \approx \frac{1}{N} \sum_{n=1}^{N} f(x_i) \tag{2}$$

Where the x_i are uniformly distributed random numbers from $(0.0, 1.0]$.

This is easy to do in python:

```
from visual import *
rand = random.random

exact = 1.0/6.0    # exact result

def f(x):
    return x*(1.0-x)   # define function f(x)

print "%s %10s %10s %10s %10s" % ('i','N','result','exact','diff')
print "-------------------------------------------------"
for i in range(9):
    N=10**i
    x = rand(N)
    y = f(x)
    result = y.sum()/N
    diff = abs(result - exact)
    print "%d %10d %10.9f %10.9f %10.9f" % (i, N, result, exact, diff)
```

which produces the following results:

```
i           N     result      exact       diff
-------------------------------------------------
0           1 0.248380566 0.166666667 0.081713899
1          10 0.149881612 0.166666667 0.016785055
2         100 0.182052216 0.166666667 0.015385550
3        1000 0.168612880 0.166666667 0.001946213
4       10000 0.167106601 0.166666667 0.000439934
5      100000 0.166795983 0.166666667 0.000129316
6     1000000 0.166683357 0.166666667 0.000016691
7    10000000 0.166706373 0.166666667 0.000039707
8   100000000 0.166669959 0.166666667 0.000003293
```

Unfortunately for 10^8 evaluations of the function we only got a relative accuracy of 10^-5. With one of our basic integration schemes like Simpson's rule we would've expected

something more like 10^{-16}! So, how is this good? The trouble with Simpson's Rule, (and all the other straightforward integration schemes) is that they get much worse in higher dimensions. If you have an integral:

$$I = \int_0^1 f(x_1, x_2, x_3, \ldots x_d) d^d x \tag{3}$$

Simpson's Rule converges with an error that's $O(N^{-2/d})$, whereas the random number scheme always converges with an error thats $O(N^{-1/2})$ *independent of d.*

So for $d > 4$ the Monte Carlo idea wins. For large d it completely *swamps* the competition. Since we're interested in integrals for multiple particles in 3-D of real space our d is going to be $3n$ where n is the number of particles. With only *two* particles in 3 dimensions, we're already going to beat Simpson's Rule using this idea.

Generating probability distributions other than uniform

To do this efficiently, we'll need to be able to generate random numbers with a variety of distributions. For example, if $f(x_1, x_2, x_3, \ldots x_d)$ is only large in a small domain many of our random numbers will be wasted on regions where f is small! If we can factor f into two parts $w(x_1, \ldots)$ and $g(x_1, \ldots)$ where we know that

$$1 = \int_0^1 w(x_1, x_2, x_3, \ldots x_d) d^d x \tag{4}$$

Then we can think of I as an average value of g where w plays the role of a probability density in the space of xs.

$$I = \int_0^1 w(x_1, \ldots) g(x_1, \ldots x_d) d^d x \tag{5}$$

Let's say we want to integrate

$$I = \int_0^\infty x e^{-\lambda x} dx \tag{6}$$

What's the probability distribution, $w(x)$ in this case? To be correctly normalized $w(x) = \lambda e^{-\lambda x}$ and

$$I = \frac{1}{\lambda} \int_0^\infty x\, w(x)\, dx \tag{7}$$

We could think of this as an expectation value of a random variable x with a probability distribution $w(x)$ (for $x > 0$). If we could figure out how to generate random numbers

with an exponential distribution we could estimate this integral by simply averaging x over that distribution. It turns out to be easy as long as the Cumulative Distribution Function (CDF) is invertible. What's the CDF? For a probability distribution $P(x)$ the CDF is defined as:

$$CDF(x) = \int_{-\infty}^{x} w(x)dx \tag{8}$$

For this example $w(x)$ the CDF is clearly $CDF(x) = 1 - e^{\lambda x}$ for $x > 0$ and 0 for $x \leq 0$. (remember $w(x)$ is only non-zero for $x > 0$ in this example). To generate random numbers with this distribution we just set the CDF to be a random number in the domain $(0, 1]$ and solve for x. Let r be such a random number:

$$\begin{aligned} r &= CDF(x) \\ r &= 1 - e^{-\lambda x} \\ e^{-\lambda x} &= 1 - r \\ -\lambda x &= \ln(1 - r) \\ x &= -\ln(1 - r)/\lambda \end{aligned}$$

So... generate a bunch of random numbers r, and compute $x = -\ln(1 - r)/\lambda$ and the xs will be distributed exponentially!

So, what are we supposed to do?

Use the Monte Carlo method to solve the integral in Eq.6. Make a histogram of the random numbers you generate (see the online documentation for the `histogram` function in graph module of visual). Generate a tabular listing of your results similar to the one shown in the example. Be sure to compare your results with the analytical solution in this case.

Questions

Please answer these questions at the end of your report.

1) Suppose rather than using exponentially distributed numbers, we'd tried to compute Eq.6 using the more straightforward approach with uniformly distributed numbers. What difficulties would we encounter? Think about how you would do it, and describe the problems you would encounter.

2) Let's say you want to create a distribution of θ and ϕ so that the angles are distributed uniformly over the surface of a sphere. Find the CDF for these two cases and describe a function that would invert the CDF (analogous to $x = -\ln(1-r)/\lambda$ for the exponential case).