Simulation methods in physics

Homework 1
Return before 13.00 on Tue 11.2., to Ari Hietanen's mailbox (C-wing 3rd floor), or by e-mail ari.hietanen@helsinki.fi

1. Show that in the expression for σ_N in Monte Carlo integrations,

$$\sigma_N = \sqrt{\frac{\langle f^2 \rangle - \langle f \rangle^2}{N-1}},$$

   where $\langle f \rangle = \sum_i f_i/N$, $\langle f^2 \rangle = \sum_i f_i^2/N$, the factor $N-1$ is indeed necessary to obtain an unbiased ("correct") estimate for σ.

   Hint: you can consider an ensemble of K separate Monte Carlo integrations, and let $K \to \infty$.

2. Let us compare the performance of the Monte Carlo integration to the regular midpoint method. Consider the integral

$$I = \int_V d^d\vec{x} f(\vec{x})$$

   where V is a d-dimensional hypercube with $0 \le x_i \le 1$, and

$$f(\vec{x}) = \prod_{i=1}^{d} \frac{3}{2}(1 - x_i^2)$$

   (The integral is of course trivial, $I = 1$ independent of d, but let us not be bothered about that and treat f as if it were a honest function. Don't cheat and separate the integral!)

   Calculate the integral in dimensions $d = 1 \ldots 10$, using

   a) the midpoint method, i.e. divide the volume into small hypercubes by dividing each axis into n intervals ($n \lesssim 10$) and sum the integral by evaluating f on the centers of the small cubes.

   b) standard Monte Carlo integration in d dimensions, using N random vectors.

   For concreteness, choose $n = 6$ and $N = 20000$. For both of the methods, report the *result* and the *cpu-time used* for the integration for each $d = 1 \ldots 10$. (if you can go to higher d, do it.) How does the error – i.e. the deviation from 1 – behave?

   The time is easy to obtain using UNIX standard `time` command: `time` *your-program-name*.

   For added interest, you may also calculate the volume of the positive quadrant of a d-dimensional sphere by substituting $f(\vec{x}) = \Theta(1 - |\vec{x}|)$.

3. RANDU is a notorious random number generator. (It was the standard generator in IBM mainframes in 60's.) It is defined by

$$I_i = (65539 I_{i-1}) \bmod 2^{31} \qquad\qquad x_i = I_i/2^{31}$$

Measure the cycle time of the generator (by brute force). Repeat it by using a couple of different seeds (i.e. values of I_0).

Generate 2-tuples $t_i = (x_{2i}, x_{2i+1})$ from successive pseudorandom numbers and plot these on a plane (dots, not connected by lines). When the range of x, y-coordinates is from 0 to 1, the distribution should look pretty homogeneous. Zoom in to a smaller square on a plane. How small the square should be before something interesting appears? Print the plot. (When you zoom in, you have to generate a much longer random sequence, since most of the tuples fall outside of the square and are rejected. Print and plot only the tuples within the square). It should be enough to have ~ 1000 points in the plot to see structure.

About implementing the generator: in most computers the size of the integer variable is 32 bits. Thus, the multiplication in the generator would cause overflow. Easiest way to avoid this is to use 64 bit double precision floating point variables for the constants and I_i. The mantissa is large enough to store the result of the multiplication exactly. (In C, the double precision mod is function fmod(a,b)). Another option is to use the Shrage method described in the lectures, or bit masks.

Even more dramatic effect can be obtained by plotting 3-tuples (x_i, x_{i+1}, x_{i+2}) inside a (small enough) 3d cube.