Exercise 2: Pseudo random number generators

N. Berger (nberger@physi.uni-heidelberg.de)

17.10.2011

"Is 2 a random number?"
Donald E. Knuth

"Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin."
John von Neumann

Please send your solutions to nberger@physi.uni-heidelberg.de until 24. 10. 2011, 12:00. Put your answers in an email (subject line *SMIPP:Exercise02*) with macro files, root files and plots as mentioned in the attachments. Test macros and programs before sending them off...

Pseudo random numbers (i.e. numbers that appear random, but can easily be reproduced), play a very important role in particle physics and are the central ingredient for all Monte Carlo methods. Probably the definitive text on the subject of pseudo random number generators is chapter 3 in Donald E. Knuths *"The Art of Computer Programming"* (Volume II, pages 1-193). Read it, if you ever consider to write a generator of your own (outside of this exercise).

1. **Make your own** Try to write code that generates apparently random numbers between 0 and 1. If you have heard of linear congruent generators before, pretend you did not.
   (Attach the .C or .py file)

2. **Test it** Use your code form above to generate 100'000 random numbers. Using `root` histograms with an appropriate binning, perform the follwoing tests on your random numbers (do not worry too much if your code fails some of them - writing good generators is hard...):

   - *Equidistribution*: Test if your numbers are equally distributed between 0 and 1;
   - *Serial test*: Test that if you look at pairs of subsequent numbers, all pairs are equally likely (you can produce 2D histograms in root with

     ```
     TH2F("name","title",100,-0.5,99.5,100,-0.5,99.5);
     ```

     the `Fill()` function now takes two arguments;
   - *Serial test (expanded)*: Do the same for triplets of numbers - 3D histograms are called `TH3F`.

- *Lower bit check*: Repeat the serial test for just the lower bits of your numbers, which you can access via `fmod(number*scale,1)`, where you should take a power of 2 for scale, powers of two are easily written as `(1 << power)`; at which point would you expect this test to fail for any generator?
- *Gap test*: Check how long the gaps between two occurences of a specific number in the sequence are (do this for 3 different numbers) - what distribution would you expect from a good generator?;
- *Up-Down test*: Check how often the difference between two numbers in the sequence is positive or negative.

(Attach the .C or .py file)

3. **Root generator** Show that the built-in default generator of `root`, `TRandom`, is a bad generator (hint: see above). Collect some evidence that this is not the case for `TRandom3`. (Attach the .C or .py file)

4. **Monte Carlo Integration** Calculate the following integral

$$\int_0^{100} \cos(2\pi x)\mathrm{d}x \tag{1}$$

- analytically,
- numerically by the approximation,

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{N} f(x_i)\Delta x \tag{2}$$

where the function f is evaluated N times with a constant step size $\Delta x = (b-a)/N$, $x_i = a + \Delta x \cdot (i + 1/2)$ (graph how the error changes as you increase N, you can either (ab)use a hisogram for this (with `SetBinContent()` you can set bin contents to arbitrary values) or use a `TGraph`),

- by Monte Carlo integration

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{N}\sum_{j=1}^{N} f(x_j) = (b-a)\cdot\langle f\rangle \tag{3}$$

where the x_j are random values in the interval from a to b, $\langle f\rangle$ is the mean of the function value. The variance of the estimate of the integral is

$$\sigma_i^2 = (b-a)^2\frac{\sigma_N^2}{N}, \tag{4}$$

where σ_N^2 is the sample variance

$$\sigma_N^2 = \frac{1}{N}\sum_{j=1}^{N}(f(x_j) - \langle f\rangle)^2, \tag{5}$$

Again show the estimate (this time with its expected error) as a function of N (use either `SetBinError()` in a histogram of a `TGraphErrors`). By the way, you can get 2π (in double precision) via `TMath::TwoPi()`.