Exercises for the lecture „Moderne Methoden der Datenanalyse”

Prof. Dr. S. Hansmann-Menzemer, M. Schiller
Physikalisches Institut der Universität Heidelberg

May 4th 2010

Exercise 2: Random Numbers

Monte Carlo simulations heavily rely on random numbers. In principle, random physics processes like e.g. the radioactive decay of nuclei could be used to obtain random numbers. However, for practical purposes, algorithms are used which produce pseudo random numbers. It is very important that the numbers from these algorithms do not contain any correlations and behave like real random numbers. Many tests have been developped to check for a (hidden) structure in a sequence of random numbers.

- **Exercise 2.1**
  Write a random number generator for uniformly distributed numbers u_j between 0 and 1 by using the formula for a linear congruent generator:

  $$i_j = (a \cdot i_{j-1} + c) \mod m, \qquad u_j = i_j/m$$

  where a, c and m are positive integer numbers. Choose a value for each of the three parameters which seems appropriate for a random number generator with a high periodicity, but be careful to avoid an overflow of the integer value range.

- **Exercise 2.2**
  Check the randomness of your random number generator from exercise 2.1, of a linear congruent generator with $a = 205$, $c = 29573$ and $m = 139968$ and of the default random generator implemented in ROOT (`gRandom->Rndm()`):

  - Fill a one-dimensional histogram of k bins from 0 to 1 with $N \gg k$ random numbers. Then calculate the value

    $$\chi^2 = \sum_{i=1}^{k} \frac{(N_i - N/k)^2}{N/k}$$

    where N_i is the number of entries in bin i. The obtained value should follow a χ^2-distribution with $k-1$ degrees of freedom. So on average $\chi^2/(k-1)$ should be 1.

- Make a two-dimensional plot using n pairs of subsequent random numbers. Use the class `TGraph` and its method `SetPoint` for this. Draw ist with the Option `"AP"`. The displayed points should be equally distributed in the square $[0,1] \times [0,1]$ without showing as structure.
- Apply the sequence or up-down test: Compare subsequent random numbers and assign to the comparison the bit 0 if the successor u_{j+1} is smaller than the previous random number u_j, assign the bit 1 if $u_{j+1} > u_j$. Now look at sequences of bits with the same value. Determine the length k of each sequence and count how many sequences of length k were produced. This number $N(k)$ should on average have the following value:

$$N(k) = \frac{2[(k^2+3k+1)N - (k^3+3k^2-k-4)]}{(k+3)!}$$

  Here $N+1$ is the total amount of random numbers. Check also that the relation $\sum_{k=1}^{N} k \cdot N(K) = N$ holds.
  Make a histogram of the obtained and of the expected $N(k)$ and plot both of them together. Use the method `SetBinContent()` for the histogram of expected values. The method `TMath::Factorial` can be used to calculate the factorial.

- **Exercise 2.3**
  Calculate the integral

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

  - analytically
  - numerically by the approximation

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

    where the integrand f is evaluated at N values with constant step size $\Delta x = (b-a)/N$, $x_i = a + \Delta x \cdot (i + 1/2)$.
  - by Monte Carlo integration as a function of $N = 1$ to $N = 150$. Add the expected error of the Monte Carlo integration to the plot. The variance V of the Monte Carlo integration is given by $V = (b-a)^2/N \cdot V[f(x_i)]$.

  (Hint: Use `TMath::TwoPi()` for 2π)

- **Exercise 2.4**
  Write a macro which generates two random number distributions according to $f(x) = 1 + x^2$ on the interval $[-1,1]$. Use the random numbers r_i in the interval $[0,1]$ from a uniform random number generator.
  We use the decomposition method where $f(x)$ is split into two parts: $f_a(x) = 1$ and $f_b(x) = x^2$. So a certain fraction of the events have to be generated according to f_a and the other events according to f_b. This

fraction is determined by calculating the integral of both function on the interval $[-1, 1]$. Since

$$\int_{-1}^{1} f_a(x)\mathrm{d}x = 2 \qquad \int_{-1}^{1} f_b(x)\mathrm{d}x = \frac{2}{3}$$

we have to generate on $3/4$ of the cases a number according to $f_a(x)$, otherwise according to $f_b(x)$.
First, a test value is generated. If this r_1 is less than 0.75, we generate a number according to f_a. Then a new r_2 is generated which gives $x = 2 \cdot r_2 - 1$. If $r_1 > 0.75$ there are two ways to generate random numbers distributed according f_b:

– **”Hit and Miss” method**
  Generate values r_j and r_k. Transform both random numbers to the considered intervals (here $[-1, 1]$ and $[0, f_{b,max}]$ respectively, here $f_{b,max} = 1$. If $r_{k,trans} \leq f_b(r_{j,trans})$, here $r_{k,trans} = r_k \leq (r_{j,trans})^2 = (2 \cdot r_j - 1)^2$, fill the histogram with $r_{j,trans}$.
– **Transformation method**
  - ∗ We have random numbers r_j distributed according to a uniform distribution $g(r)$ and want to generate random numbers x_i according to a probability density function (p.d.f.) $f(x)$ in the interval $[p, q]$.
  - ∗ From the equation $f(x)\mathrm{d}x = g(r)\mathrm{d}r$
    we get $r = F(X) = \int_{-\infty}^{x} f(x')\mathrm{d}x'$, which we solve: $x = F^{-1}(r)$.
  - ∗ If r_j are uniformly distributed random numbers between $F(p)$ and $F(q)$, the x_i are following the p.d.f. $f(x)$.
  - ∗ The method works well if $F(x)$ is analytical and can be easily inverted.

  Fill x_i into a histogram. Here, $x_i = (3r_j - 1)^{1/3}$, $r_j \epsilon [0, \frac{2}{3}]$.
  (Hint: x^y in C++: `std::pow(x,y)`, from `#include <cmath>`)

Using a uniform random number generator, it is possible to generate a random number according to any sample and thus simulate each distribution. There are different methods which are summarised on the sheet “Summary of Main Concepts”.

(Exercises with * can be skipped if no time is left.)

- **Exercise 2.5***
  Generate random numbers according to an exponential distribution $\exp(-x)$ for $x > 0$. Take uniformly distributed random numbers and apply the transformation method. Write 100,000 expontenially distributed random numbers to an ntuple (root class **TNtuple**).

- **Exercise 2.6***
  Use the uniform random number generator for simulating the coin flipping of eight coins. Let us do this a thousand times. When the random number is greater or equal 0.5, we will call it head (1), otherwise tail (0).

  – Plot the average sum as a function of the flip number. To calculate the average sum, use the *running average.* The running average is defined as
  $$\mu_k = \frac{k-1}{k}\mu_{k-1} + \frac{1}{k}x_k$$
  where μ is the average, k the number of flips, x_k is the averaged coin flip result and $\mu_0 = x_0$. The running average is a very handy tool since it is easily updated after every new measurement and it avoids problems with large sums.

  – Occasionally you will throw 5 heads and 3 tails.(What do you expect how often this will happen?) Plot the distribution of the difference of the flip number, e.g. if you throw this combination in experiment 5 and then again in 9, the difference is 4.
  Notice that you have generated a Poisson distribution using a uniform random number generator. Make an exponential fit and explain the fit result.