Kirchhoff-Insitut für Physik
Physikalisches Insitut

Winter semester 2013-14
KIP CIP Pool (1.401)

# Exercises for Statistical Methods in Particle Physics

http://www.physi.uni-heidelberg.de/~nberger/teaching/ws13/statistics/statistics.php

**Dr. Niklaus Berger** (nberger@physi.uni-heidelberg.de)
**Dr. Oleg Brandt** (obrandt@kip.uni-heidelberg.de)

# Exercise 2: Pseudo random number generators

**21. October 2013**
**Hand-in solutions by 14:00, 27. October 2013**

*"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

Like last time, please send your solutions to `obrandt@kip.uni-heidelberg.de` by 29.10.2013, 14:00. Make sure that you use *SMIPP:Exercise02* as subject line. Please put each macro into one separate .C or .py file, which can easily be tested (i.e. executable via e.g. `root -l mycode.C`). Test macros and programs before sending them off...

Pseudo random numbers (i.e. sequences of numbers that appear random, but are fully deterministic and 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 Write your own pseudo random number generator

Write a programme that generates pseudo random numbers in [0, 1] in double precision. If you have heard of linear congruent generators before, pretend you did not.
*(Attach the .C or .py file)*

## 2 Evaluating properties of pseudo random number generators

Use your programme above to generate 100,000 random numbers. Using `root` histograms with an appropriate binning, perform the following tests on your random number sequence (do not worry if your code fails some of them - writing good generators is hard...):

- *Equidistribution*: Test if your numbers are equally distributed in the inteval [0, 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) ;
```

  Note that the `Fill()` function now takes two arguments;

- *Serial test (expanded)*: Do the same for triplets of numbers using `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 the `scale` variable.
- *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 Evaluate standard ROOT pseudo random number generators

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 $\sigma_i^2 = (b-a)^2\sigma_N^2\,/\,N$, where $\sigma_N^2$ is the sample variance

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

  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 $2\pi$ (in double precision) via `TMath::TwoPi()`.