Exercise 3: Probability density functions

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

29.10.2012

Please send your solutions to `nberger@physi.uni-heidelberg.de` until 5. 11. 2012, 12:00. Put your answers in an email (subject line *SMIPP:Exercise03*) with macro files, root files and plots as mentioned in the attachments. Test macros and programs before sending them off...

1. **Random walk** Simulate a random walk in one dimension. Take a test particle at $x = 0$, $t = 0$. In every time step, the particle has a 50% probability of moving one step to the right or one step to the left (use a random number generator to decide the direction). Track the particle for 100 time steps and then store the end position in a histogram. Repeat for many particles. What distribution do you see in the histogram?
(Attach the .C or .py file)

2. **Analytic muon beam** The Paul Scherrer Institute (PSI) in Switzerland provides the most intense continuous muon beams in the world. These beams are created by shooting over 2 mA of 590 MeV/c protons at a 4 cm thick carbon target. In the $p - C$ interactions, many pions are generated and then stopped in the target. Charged pions decay mostly to a muon and and a muon (anti)neutrino, $\pi^+ \to \mu^+ \nu_\mu$ and $\pi^- \to \mu^- \bar{\nu}_\mu$. These are the muons (usually the positive ones) used in the experiments. In the pion rest system, the muons will have a fixed momentum (two-body decay) of $p_{max} = 29.79$ MeV/c. Muons traversing the target material will loose some energy, thus the highest momentum muons come from the target surface. In fact, the muon intensity behaves as

$$I(p) = \begin{cases} I_0 \cdot p^{3.5} & \text{if } 0 < p < p_{max} \\ 0 & \text{otherwise} \end{cases} , \tag{1}$$

where I_0 is an (arbitrary) normalization. For the simulation of the future $\mu \to eee$ experiment at PSI, we would like to generate muons with this momentum distribution, but we only have a generator (e.g. `TRandom3`) for equidistributed numbers from 0 to 1.

If a general distribution $f(x)$ is analytically integrable ($F(x) = \int_{-\infty}^{x} f(t)dt$ exists) and the integral $F(x) = u$ is analytically invertible ($x = F^{-1}(u)$ exists), then if we generate u_i equidistributed in $[0, 1]$, $x_i = F^{-1}(u_i)$ will follow the distribution $f(x)$. Use this to generate muon momenta. Due to the nature of the distribution, you can replace $-\infty$ in the lower bound of the integral with 0. You also have to make sure that the integral up to the upper bound p_{max} is normalized to 1.

Generate 100'000 muon momenta and fill them into a histogram with appropriate binning.
(Attach the .C or .py file)

3. **Non-analytic muon beam** In reality, the sharp edge at $p_{max} = 29.79$ MeV/c is washed out because many pions are not perfectly at rest in the target. The resulting distribution is the intensity from the above problem (I_{ideal}) convoluted with a Gaussian distribution

$$I_{real}(p) = \int_{-\infty}^{+\infty} I_{ideal}(\tau) \cdot g(p-\tau) d\tau = \int_{-\infty}^{+\infty} g(\tau) \cdot I_{ideal}(p-\tau) d\tau \quad (2)$$

where $g(x)$ is the Gaussian or normal distribution,

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (3)$$

In our example, the width σ of the distribution happens to be a convenient 1 MeV/c and the center μ is conveniently at 0. The above integral has no analytical solution, so we are going to approximate it numerically by a sum (using the fact that the normal distribution falls off rather rapidly):

$$\int_{-\infty}^{+\infty} g(\tau) \cdot f(x-\tau) d\tau \approx \frac{\sum_{\tau=-a}^{a} g(\tau) f(x-\tau)}{\sum_{\tau=-a}^{a} g(\tau)}, \quad (4)$$

where a is chosen as a few (e.g. 3) σ of the normal distribution and the steps of τ are chosen to be small enough. Write a function that implements this convolution starting from a function implementing a normal distribution with a σ of 1 and a mean of 0 (does not need to be normalised) and a function implementing I_{ideal}. In native (C++) root, these functions should have prototypes of the form

```
double myFunction(double * x, double * par){
    p = x[0];
    parameter0 = par[0];
    parameter1 = par[1];
    ...
    return result;
}
```

where `x` is an array of the running variable (of length 1 in a 1D function) and `par` is an array of the function parameters. In Python, everything is somewhat simpler due to implicit typing:

```
def myFunction(x, par):
    p = x[0]
    parameter0 = par[0]
    parameter1 = par[1]
  ...
  return result
```

These functions can then be used in TF1 objects to draw the function.

```
TF1 * rootfunction = new TF1("myFunction",myFunction,0,30,2);
rootfunction->SetParamter(0,1.3);
rootfunction->SetParamter(1,2.7);
rootfunction->Draw();
```

where the arguments to the constructor are name, the actual function, the lower and upper edges of the range and the number of parameters, which can then be set via `SetParameter(index, value)`. In python this works analogously. Determine the number of steps needed for τ by drawing the function and increasing the number until you obtain a smooth behaviour. (Attach the .C or .py file)

4. **More non-analytic muon beam**

   Use the hit and miss method to generate muons with the distribution obtained in the last problem in the range from 0 to 35 MeV/c. First find the maximum value of the function I_{real}^{max}. Then generate pairs of equidistributed random numbers, x_i and y_i. Scale x_i to the range from 0 to 35 MeV/c and use it as the muon momentum. Then determine I_{real} for that value of p and keep the event if $y_i < I_{real}/I_{real}^{max}$, otherwise reject it. Plot the resulting distribution. Count how many random numbers you need per generated muon. Could this be made more efficient? How?