TRINITY COLLEGE DUBLIN, SCHOOL OF MATHEMATICS

Assignment #1, Module: MA5634

Gustavo Ramirez

November 14, 2016

1 COMPARISON OF RANDOM GENERATORS

The OS used in general for this assignment is Ubuntu 16.04 LTS, with Python 2.7 as scripting language.
According to the original dieharder's website [1], the two random generators required to test, The Mersenne Twister and RANDU, are labeled by 041 and 403, respectively.
From the man pages of the dieharder command, the -g flag allows to specify which random numbers generator is to be tested, and the -a enables all tests to be performed. Then, for testing the two generators required:

```
$ dieharder -g 041 -a
$ dieharder -g 403 -a
```

and the two outputs were redirected to the files ./dieharder_tests/output_randu.txt and ./dieharder_tests/output_mersenne.txt.
It is noticeable how The Mersenne Twister got much better results in the tests than RANDU, as the former didn't fail a test, while the latter failed more than half of the tests.

[1] http://www.phy.duke.edu/~rgb/General/dieharder.php

2 On RANLUX installation and usage

For the usage of the program RANLUX from Python, an intermediate program was used to access the random numbers generator from Python.
In the file ./mediator_ranlux.c, a program in C was written to call the ranlxs.h library, and from it generate the required random numbers. In lines from 7 to 9 in that C file (the commented lines), is an specificacion of the compilation process for generating the program ./mediator_ranlux, which is then called from Python to retrieve lists of random numbers.

3 Brief explanation on integration by using random numbers

For a random variable U uniformly distributed in the interval $(0,1)$, the following relation applies:

$$E[g(U)] = \int_0^1 g(x).$$

And it is known, by the strong law of large numbers, that:

$$\sum_1^k \frac{g(U_i)}{k} \rightarrow E[g(U)] = \int_0^1 g(x).$$

meaning that we can approximate the desired integral by generating a large number of random numbers u_i, and then performing an average of $g(u_i)$. This is the Monte Carlo approach to integration.

4 Analytical values

Before going into describing the numerical simulations, the whole set of analytical values is established here, in order to be able to make direct comparisons and error determinations:

- linear function

$$\mu = E[g(x)] = \int_0^1 g(x)\,dx = \int_0^1 x\,dx = 0.5$$

$$VAR[g(x)] = E[(g(x)-\mu)^2] = \int_0^1 (x-0.5)^2\,dx = \tfrac{1}{12} = 0.08333$$

- quadratic function

$$\mu = E[g(x)] = \int_0^1 g(x)dx = \int_0^1 x^2\ dx = \frac{1}{3} = 0,333....$$

$$VAR[g(x)] = E[(g(x)-\mu)^2] = \int_0^1 (x^2 - \frac{1}{3})^2 dx = \frac{4}{45} = 0.08889$$

- square function

$$\mu = E[g(x)] = \int_0^1 g(x)dx = \int_0^1 \sqrt{x}\ dx = \frac{2}{3} = 0,666....$$

$$VAR[g(x)] = E[(g(x)-\mu)^2] = \int_0^1 (\sqrt{x} - \frac{2}{3})^2 dx = \frac{1}{18} = 0.05556$$

5 STRUCTURE OF SCRIPTS AND RESULT FILES

The solution to this assignment is composed of the following directories and files:

- data_for_plots (dir): these is the data generated from the execution of the whole set of numerical simulations. The format of all these files is a two-columns format, in which the first column (left column) represents the *x* axis.
- mediator_ranlux* (files): the set of files and directories obtained from the official RANLUX program and documentation. the directory ranlux-3.3/ was kept, in order to be able to access to the COPYING, README and documentation of the original source code.
- gnuplot_data.gp (file): this is a gnuplot script for plotting all the files in the directory data_for_plots.
- plots (dir): once the script gnuplot_data.gp is executed, all the plots are saved to this directory.
- general.sh (file): this bash script was written in order to centralize all the executions. This script executes first the Python script for generating all the data after the numerical simulations, and then executes the gnuplot script to obtain the plots from data.
- integration_by_rand.py (file): this Python script is in charge of all the numerical simulations (from summing and averaging over the random samples, to call RANLUX for accessing the random samples).
- notes_on_theory_and_results.tex (file): the TeX file for this PDF.

6 GOING DEEPER IN THE DESCRIPTION: INTEGRATION_BY_RAND.PY

The script for performing the whole set of numerical simulations is integration_by_rand.py. Within that script, the most important functions are:

- ranlux_call: returns a string, with the format of a Python list, and that list contains a set of N random numbers
- monte_carlo_integration: takes the list retrieved by the Python function ranlux_call and takes care of the averaging process (both for averages and variances)
- average_running: writes the numerical simulations outputs in the corresponding files

7 ACCURACIES AND NUMBER OF SAMPLES

The required number of samples N to achieve a certain accuracy, for the three different approximated functions, is:

- linear:
  - 2 digits: ~3800
  - 3 digits: ~48300
  - 4 digits: ~70000 (although, is important to note that around 100000 random samples, the 4 digits accuracy dissapears, in the case of this simulation)
- quadratic:
  - 2 digits: ~4850
  - 3 digits: ~50000
  - 4 digits: ~70000 (again, the 4 digits accuracy dissapeared at around 100000 random samples)
- square:
  - 2 digits: ~7700
  - 3 digits: ~70000
  - 4 digits: ~?

Taking the three ordered pairs for the linear case, and fitting a linear equation for the first two pairs (in all the following fits, the output of the function f is samples number N):
$f(digits = d) = 44500d - 85200 \Rightarrow N = f(8) = 270800$
Similar (not so realistic) results are obtained for the two other functions. More realistic results are obtained with exponential fitting (the errors and coefficient of determination are omitted for this fitting process):

- $f_{linear} \approx f_{quadr} = 4516e^{0,7067d} \Rightarrow f_{quadr}(8) \approx 1\,288\,490$
- $f_{sqrt} = 93,17e^{2,2d} \Rightarrow f_{sqrt}(8) \approx 4100\,709\,241$ (for this third case, not enough information is available to make the prediction more sensible)

8 VARIANCES WITH EXACT AND APPROX AVERAGE

- deeper in the description of the files in the directory ./plots/: there are four types of files in this directory: approxvariances* (variances are calculated using approximated/simulated values for the average), exactvariances* (an exact value for the average is taken), errors*, and the approximating plots.
- comparison of variances with exact and approx averages: at the scale of simulated values (around 350 000 random samples for the maximum set), there appears to be no difference between the two simulations.
- comparison of variances and errors (plots and analytical values): as can be seen from the files linear.dat, quadratic.dat and sqrt2.dat at ./data_for_plots/, the 'final' values using 350 000 samples are (respectively): 0.49927915..., 0.3325852... and 0.6661264..., which results in accuracies of: 0.011..., 0.001... and 0.0005... It is noticeable that the lowest analytical variance is in the square root case (~0.05556), which results also in the best accuracy (~0.0005 as mentioned before).

9 ON THE SUGGESTED CALCULATION FOR π

Although the result of the integral of $g(x) = \frac{2}{\sqrt{1-x^2}}$ from 0 to 1 helps in obtaining the value of π analytically, there is an indefinition of the integrand at 1, which makes the integral improper (as the integran goes to infinity for the upper limit of the integral).

So far, in the implemented code for Monte Carlo integrations, uniformly distributed stochastic variables have been used. For improper integrals, the uniform distribution for such variables is inadequate.

If the number of samples N is taken to infinity, there shouldn't be a problem on the numerical integration. The problem is that N is always finite. When integrating numerically, many large jumps (around the divergence of the integrand) will appear, caused by occasional points falling very close to the upper limit 1 (in this case); thought this jumps are rare, they contributed largely in the sums and then the averages become arbitrarily large.