EE2 2006
University of Iceland

Hannes Jónsson
Kristján F. Alexandersson

Monte Carlo Simulations and Analysis of Experimental Errors

In this exercise, you will use the random number generator to simulate experimental measurements which are subject to errors and you will analyse the error in various ways, in particular find out how the error of an estimate is reduced as more measurements are carried out. First, you will explore the functionality of the random number generator in Matlab. Then, you will use random numbers to estimate π and carry out various statistical analysis of the results to see how well the results converge to the true value as the number of trials is increased. The calculation of π illustrates how random numbers can be used to estimate a ratio of integrals, an example of so called Monte Carlo simulations.

A. Uniform random numbers

Most computers have a function predefined for generating random numbers uniformly distributed on the interval [0,1]. There are many different random number generators. None of them generates truly random numbers but rather a sequence of numbers which appear to be random for most practical purposes. The string of numbers is, in fact, deterministic and there is inevitably some correlation between the numbers. Also, the same numbers will eventually be repeated, but the repeat periodicity is very long. Often these are called 'pseudo-random' numbers. Some of the random number generators are better than others. For many applications pseudo-random numbers are random enough.

In Matlab (and Octave) the random number generator is called with the rand function. No argument is needed. An input number is in fact needed for any random number generator, the so called 'random number seed', but if no argument is given, the seed is set to 0 each time the program is started. If you want your calculations using random numbers to be reproducable step by step, then the random number seed can be set using the rand('state'). For example, try calling rand a few times

```
>> rand
```

A list or matrix of random numbers can be generated by defining the dimensions in the rand function

```
>> rand(1,7)
```

Run this command a few times. Each time the output is different. Now, set the random number seed to a certain value

```
>> rand('state',1000);
```

and again get a list of random numbers

```
>> rand(1,7)
```

If the seed is again set to the same value

```
>> rand('state',1000);
```

the same list of numbers is generated

```
>> rand(1,7)
```

There are several tests that a random number generator needs to pass in order to qualify as a useful random number generator. A minimal requirement is that the probability distribution function is constant over the interval [0,1]. To test this, you can generate a plot showing how often a random number is obtained in various subintervals. Such a plot is called a histogram. First a list of random numbers is generated. Let the variable 'npoints' specify the length of the list, here chosen initially to be 100.

```
>> clear all
>> npoints=100;
>> rannumb=rand(1,npoints);
```

Then, the interval [0,1] is divided into bins. Let the number of bins be specified by the variable 'nbins'. The bin number that a given random number belongs to can be obtained by multiplying the random number with nbins (then a number ranging from 0 to nbins will be obtained), adding 0.5, and then rounding to the nearest integer using the round function. This ensures that numbers in the range [0.0,0.1[will be counted in bin 1, numbers in the range [0.1,0.2[will be counted in bin 2 and so on for nbins = 10.

```
>> nbins=10;
>> bins=round(rannumb*nbins+0.5)
```

Q1: Why is it necessary to add 0.5 in the argument of the round function in this calculation?

The Matlab function hist can be used to count how often a certain pattern (here a given number) appears in a list and then create a histogram of the result. As always when using a function for the first time, it is useful to read the hist help file to better understand the parameters that the function uses.

```
>> hist(bins,1:nbins)
```

The histogram shows a course grained distribution function for the random numbers over the interval [0,1]. With only 100 random numbers, the statistics are not very good, and the histogram shows large variations from one bin to another.

Q2: Repeat the above procedure (copy all the relevant lines into a m file) for 1000, 10000 and 100000 random numbers. Remember to suppress the output from the lists by adding a semicolon (;) at the end of the function call.