EE2 in 2006
University of Iceland

Hannes Jónsson
Andri Arnaldsson
Susan Rempe
Kristján Alexandersson

B. Simulated experiments: Monte Carlo estimates of π

In this exercise you will use random numbers to estimate π. This is an example of a 'Monte Carlo' simulation. An estimate of π is obtained from a set of trials. This is analogous to an experimental measurement where the best estimate of some quantity is obtained as an average over a few measurements. The question is, how does one estimate the error in such an estimate? Typically, the results of a numerical calculation or experimental measurement should be accompanied by some estimate of the error. Each measurement is subject to various sources of error due to the instruments used, sample preparation, etc. By carrying out several independent measurements and then taking the average of the results to obtain a best estimate of the quantity being measured, the non-systematic sources of error can be reduced. The more independent measurements are carried out, the better the estimate becomes when an average is taken. The question is, how does one estimate the error in the measured value, and how does the error get reduced as more measurements are included in the average? (A very good discussion of error analysis is given in the book 'Experiments in Physical Chemistry' by Shoemaker, Garland and Nibler, chapter 2B).
In order to study error analysis, you will simulate experimental measurements using random numbers. In each 'trial', two random numbers will be used to estimate the ratio of a circle and a square that inscribes the circle. The ratio of the area of the circle and the area of the square is π/4, so this can be thought of as an 'experimental' measurement of π. The ratio of the area of the circle and the quare can be estimated by counting how often a pair of random numbers, {x,y}, distributed on the interval [-0.5,0.5] lands inside the circle. Assuming the circle has a radius of 0.5, the condition for a hit is

$$x^2 + y^2 < 0.25$$

A simulation using random numbers is usually referred to as a Monte Carlo simulation. A Matlab code for estimating π by this method can be written in the following way (piestimate.m)

```
----------------------------------------------
clear all
close all
ndarts=100;
for i=1:ndarts
        xcoord(i)=rand-0.5;
        ycoord(i)=rand-0.5;
end
distancesq=xcoord.^2+ycoord.^2;
nhits=0;
for i=1:ndarts
    if distancesq(i)<=0.25
        nhits=nhits+1;
    end
end
estimate=4*nhits/ndarts
----------------------------------------------

>> piestimate
```

The latter for loop increases the counted number of hits by 1 each time the the distance squared between the {x,y} point and the origin is less than or equal to 0.25. The Monte Carlo simulation can be illustrated by graphing the circle and the location of the random number pairs. Adding the following commands to the end of piestimate.m will achieve just that (although here Octave users might not get as pretty a picture due to the graphics incompatibilities).

```
--------------------------------------------------------
t = 0:pi/20:2*pi;
plot(sin(t)/2,cos(t)/2,xcoord,ycoord,'*r','Markersize',10)
axis image
--------------------------------------------------------
```

Q1: Run the calculation a few times (remember to generate a new set of random numbers) and notice how the pattern changes. Each time a different set of random number pairs (x,y) is generated. What is, roughly, the magnitude of the error in these estimates?

The results of the calculations above show clearly that more than 100 random number pairs need to be used in order to get a good estimate of π. To study systematically how the error in an estimate obtained from a Monte Carlo simulation varies with the number of trials, it is instructive to carry out several simulations, each with some specific value of trials (here random number pairs), and analyse how the estimated value varies. In this exercise, you will estimate π from a simulation using a certain number of random number pairs, say 'ndarts'. You will then repeat the simulation several times, thereby obtaining a set of estimates of π. You will then use the statistical methods that are often used to analyse experimental results to analyse your data. This involves calculating the variance and standard deviation. The key question is: How does the standard deviation in the estimate of π depend on the number of trials used in the Monte Carlo simulation? Clearly, the results are better when more trials are used, but how many more trials are needed in order to improve the estimate by one significant figure?

Before carrying out multiple Monte Carlo simulations, it is convenient to turn the Matlab code into a function.
Defining a function called throwdarts with an argument that specifies how many 'darts' (random number pairs) to use

```
--------------------------------------------------
```

```
function estimate=throwdarts(ndarts)

for i=1:ndarts
        xcoord(i)=rand-0.5;
        ycoord(i)=rand-0.5;
end

distancesq=xcoord.^2+ycoord.^2;

nhits=0;
for i=1:ndarts
    if distancesq(i)<=0.25
        nhits=nhits+1;
    end
end

estimate=4*nhits/ndarts;
--------------------------------------------------
```

The Monte Carlo calculation can then be carried out using, for example, 10000 darts by

```
>> throwdarts(10000)
```

Q2:   Do a few Monte Carlo calculations using 10,000 darts per estimate. What is, roughly, the magnitude of the error in these estimates?

A large data set with results from several experiments can now be generated. Let the variable 'nexp' be the number of Monte Carlo calculations (number of 'experiments'), each one carried out with 'ntrials' pairs of random numbers ('darts'), and then let ntrials increase in increments of 'nincr' so that several sets of experiments gets generated, each set corresponding to different number of trials used. For example, if the first set of 25 calculations involve only one dart each, then another set of 25 calculations involving 6 darts, etc, up to 250 darts per experiment, then the array of data can be generated by the following commands

```
------------------------------------------------
ntrials=250;
nincr=5;
nexp=25;

for i=1:nexp
    for j=1:nincr:ntrials
        data(i,1+(j-1)/5)=throwdarts(j);
    end
end

data
------------------------------------------------
```

Note that calculations using only one dart either give an estimate of 4 or 0 depending on whether the dart landed inside the circle or not. The results of the n-th calculation carried out with 1+5*(m-1) trials can be retrieved from the array with data(n,m).

This data set, called 'data', contains all the data needed to carry out an analysis of the error in the Monte Carlo estimate. The goal is to plot the standard deviation as a function of the number of random number paris used in the calculation. First of all, a mean value needs to be found for all the experiments which were carried out with the same number of darts. Then the variance and finally the standard deviation need to be calculated.

The mean value from all the 25 'experiments' that make use of the same number of trials can be calculated in the following way

```
----------------------------------------------------
m=ntrials/nincr;

for k=1:m
    datamean(k)=0;
    for i=1:nexp
        datamean(k)=data(i,k)/nexp+datamean(k);
    end
end

datamean
----------------------------------------------------
```

The for function is used to repeat the same calculation many times as specified by an index, here 'k' that runs from 1 to 'm'. Here two such loops are being used, the inner one (over the index i) loops over all the 'experiments' with a given number of trials, and the outer one loops over the various nunmber of trials used.

For those of you interested in efficient computing, the following code is about 2 times faster, because it bypasses the use of for loops.

```
----------------------------------------------------
lgth=size(data);
```

```
datamean=sum(data,1)/lgth(1)
----------------------------------------------------
```

The variance is defined as

$$\mathrm{var} = \frac{1}{N-1}\sum_{i=1}^{N}(x-\bar{x})^2$$

and the standard deviation (a commonly used estimate of the 'error') is

$$\sigma=(\mathrm{var})^{1/2}$$

Q3: Calculate the variance and standard deviation (std) from your data set. Make sure you treat separately experiments based on different number of trials (you should not average over results obtained with different number of trials).

To plot the standard deviation as a function of the number of trials, it is necessary to create a vector containing the standard deviation (Q3) and a second one corresponding to the number of trials. The trials vector and plot is is created as follows

```
---------------------------
for i=1:m
   trials(i)=1+5*(i-1)
end

plot(trials,std,'r*')
---------------------------
```

Q4: How much does the standard deviation drop as the number of trials is increased? For example, if one would like to add another significant figure to an estimate of some quantity, it is necessary to decrease the standard deviation by a factor of ten. How many more measurements would that require? The theory of statistics predicts that the standard deviation drops as the inverse square root of the number of measurements

$$\sigma = \frac{\alpha}{\sqrt{N}}$$

where N is the number of trials, and α is a proportionality factor (see helpful hints below).

As Matlab does not have a built in fit function for curves other than polynomial and exponential ones, it becomes necessary to use linear algebra to fit the statistical prediction to the results.

For example, to model the data sets y and t with a decaying exponential function y(t)=c1+c2*exp(-t) we procede as follows

```
--------------------------------------
t = [0 .3 .8 1.1 1.6 2.3]';
y = [.82 .72 .63 .60 .55 .50]';

A = [ones(size(t)) exp(-t)];
c=(A'*A)\(A'*y)
--------------------------------------
```

Matlab returns

```
c =
     0.4760
     0.3413
```

which means in other words that the least squares fit to the data is y(t)=0. 4760+0.3413*exp(-t)

Q5: Fit the data to the statistical prediction and plot the best fit curve on top of the data to show how well this prediction works.

Optional material:

The Monte Carlo calculation can easily be generalized to higher dimensions. For example, in three-dimensions, a triplet of uniform random numbers needs to be generated for each trial and the trial is a success if the point lands inside a sphere that is inscribed into a cube. In an analogous way, the calculation can be generalized to an arbitrary number of dimensions, in which case one refers to a hypersphere inscribed in a hypercube. Multi-dimensional integrals are common in science, for example in statistical mechanics. For each dimension added, the probability that a dart lands within a hypersphere becomes smaller and smaller. The ratio of the volume of the hypersphere and the hypercube becomes smaller and smaller the larger the dimensionality, and therefore the probability of landing inside the hypersphere becomes smaller. A Matlab function for throwing darts in arbitrary number of dimensions is given below.

```
--------------------------------------------------------------------------------------------------------------------
function dartsinDdim(ndarts,ndim)

for j=1:ndarts
   distancesq(j)=0;
   for i=1:ndim
      distancesq(j)=(rand-0.5)^2+distancesq(j);
   end
```

```
end

nhits=0;
for i=1:ndarts
    if distancesq(i)<=0.25
        nhits=nhits+1;
    end
end

estvol=2.0^ndim*(nhits/ndarts);
exactvol=pi^(ndim/2.)/gamma(ndim/2.+1);
cubevol=2.0^ndim;
fracvolofsphere=exactvol/cubevol;

% display command not used by octave so remove these commands
display(['number of hits = ' num2str(nhits)]);
display(['estimated volume = ' num2str(estvol)]);
display(['exact volume = ' num2str(exactvol)]);
display(['volume of hypersphere / volume of hypercube = ' num2str(fracvolofsphere)]);

% and instead use these
% nohits=nhits
% estimvolume=estvol
% exactvol=exactvol
% fracvol=fracvolofsphere
```

Throwing 1000 darts in 5 dimensions can now be done easily by running the function

```
>> dartsinDdim(1000,5)
```

Q6: Carry out calculations in 3, 4, 5, and 10 dimensions using at least 1000 trials. Notice how the success rate drops as the number of dimensions is increased. This illustrates that Monte Carlo calculations of systems with many degrees of freedom need to make use of special sampling algorithms in order to be a viable way of doing calculations. Many such algorithms exist, such as the Metropolis algorithm and umbrella sampling.