Tutorial

2: Random walks

Kai Grass*

April 24, 2007
SimBio group, FIAS, Frankfurt

Contents

1 Important sampling

This section of this tutorial is a wrap-up of last weeks introduction to simple Monte Carlo sampling.

A major draw back of a simple MC sampling can be seen by looking at Figure 1 which shows the normal distribution N(x)

$$N(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{(2\pi)}}\exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

and the probability density function of a uniform distribution $U(x)$ and that of a Cauchy distribution $C(x)$

$$C(x;\mu,\gamma) = \frac{1}{\pi}\left[\frac{\gamma}{(x-\mu)^2+\gamma^2}\right].$$

*grass@fias.uni-frankfurt.de

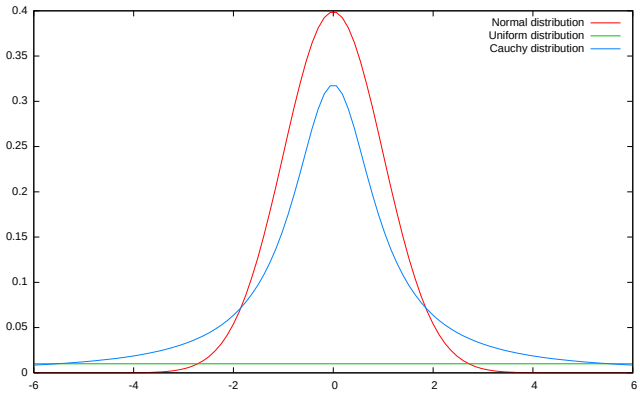


Figure 1: The MC sampling problem.

It is difficult to correctly sample the fast decaying tails of the Normal distribution with random numbers drawn from a uniform distribution. Important sampling offers a smart way around this sampling problem. At some cost however.

For uniform sampling, we can calculate the integral value by the following relation.

$$\tilde{F}(a,b) = (b-a)E(f(x)) = \frac{1}{n}\sum_{i=1}^{n} W(x_i)f(x_i),$$

where $W(x_i)$ is the individual weight of a random sample x_i. $W(x_i)$ normalizes the probability distribution. For uniform sampling $W(x_i) = (b-a)$ is independent of x_i, i.e. every number is equally likely.

If we use non-uniform sampling, the weighting function $W(x_i)$ is not a constant anymore, but depending on x_i. In fact, it is the inverse of the correctly normalized probability distribution.

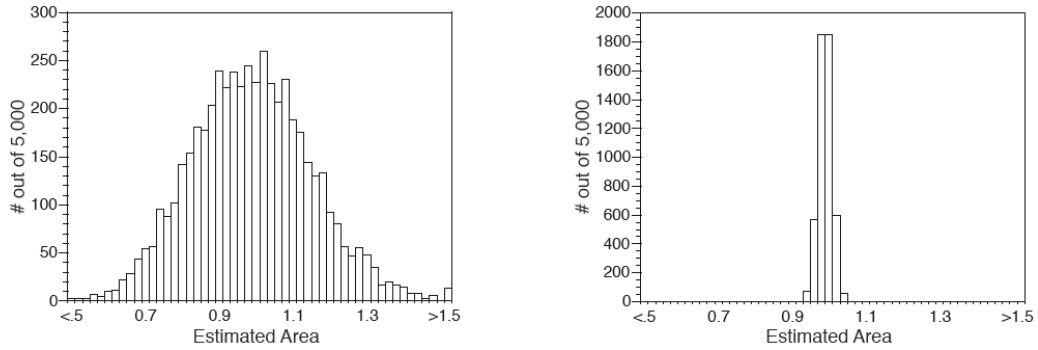
$$W(x_i) = \frac{1}{p(x_i)} \quad \text{, where} \quad \int_a^b p(x)dx = 1.$$

1.1 Tasks

1. Read the code `mc-integration.c` and try to understand it. It uses GSL routines where possible. Check the GSL documentation to learn more about them.

2. Compile and run the sample program. Look at the difference in the accuracy between an uniform sampling and an important sampling using the Cauchy function.

3. Compare your observations to Figure 2. Do you see the same behaviour?



(a) Uniform (b) t_1 Dist.

Figure 2: Distribution of the estimated area with uniform (a) and with important sampling (b).

1.2 Supplementary tasks

1. **Optional:** Figure 2 shows the Area distribution obtained from 5000 independ estimates of 1000 samples each. Try to obtain a similar picture from `mc-integration.c` by adding an appropriate binning function. (This task is non-trivial and will take some thinking and time if you attempt it.)

2 Random walk in one dimension

The second part of today's tutorial covers random walks. We start with a random walker in one dimension.

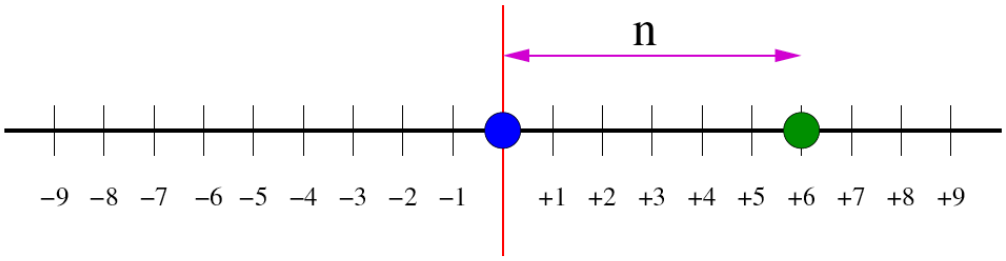


Figure 3: One dimensional random walk.

The random walker starts out at the origin and can make a fixed number of steps N. Each step is randomly chosen, either to the left or to the right with equal probability ($p = 0.5$).

N	10	20	50	100	200	500	1000
$\langle Re^2 \rangle$							

Table 1: Mean end-to-end distance of a 1D random walk for varying length N.

2.1 Tasks

1. Read the code `randomwalk-1d.c` and try to understand it.
2. Compile and run the sample program for different lengths of the random walk. Confirm, that in average the final position is close to the origin, i.e. the random walker is not biased.
3. Estimate the number of samples to average that is needed to obtain good statistics at acceptable computing time.
4. Complete Table 1 with the values for the averaged squared end-to-end distance for varying lengths N of the random walk. Use the sample size estimated before.
5. Plot the data of Table 1 and compare to theoretical prediction $\langle Re^2 \rangle \propto N$.

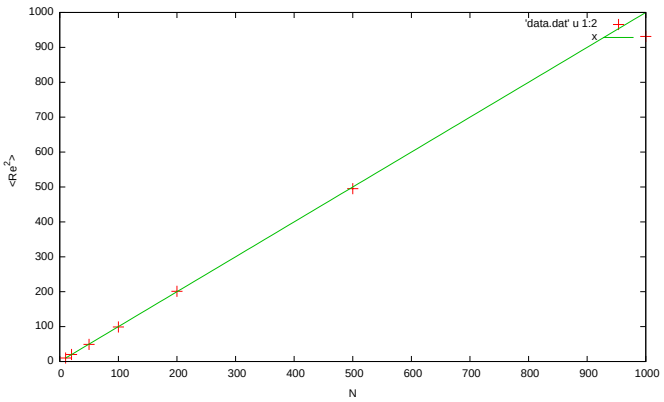


Figure 4: Averaged squared end-to-end distance of a one-dimensional random walk.

Hint: To plot the data, it is recommended to use `gnuplot`, but if you are more familiar with another graphics program, feel free to use that.

If the data to plot is stored in a datafile called `data.dat` in columnar fashion, i.e.

```
#N   <Re^2>
10   10.2
20   20.1
50    53.3
..
```

you can use `plot 'data.dat' using 1:2` to plot this data in `gnuplot`. You can also include the theoretical predition into this plot by extending the command.

```
plot 'data.dat' using 1:2, x
```

Your results should look similar to Figure 4.

2.2 Supplementary tasks

1. Increase the probability for the random walker to go right to 0.8 (consequently, the probability to go left now is only 0.2). Check the influence on the mean final position.

3 Random walk in two dimensions

In this section, we study the two dimensional random walk.

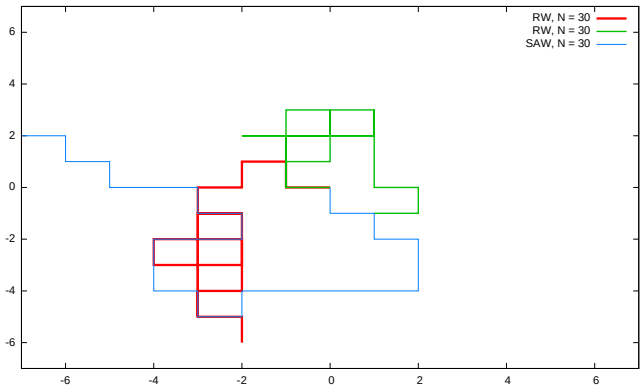


Figure 5: Sample two-dimensional random walks.

In two (or more dimensions) it is possible to distinguish two different types of random walks (see Figure 5): *normal random walks (RW)* and *self-avoiding random walks (SAW)*. SAW have the decisive property, that they never return to a place they have been to earlier, i.e. they do not touch or cross itself. This restriction is not made for general random walks and leads to significantly different properties.

We generate two dimensional random walks, by randomly choosing one of the four directions (up, down, left, right) at each step. A random walk is stored as sequence of numbers from 0 to 3, indicating the chosen directions. To analyze the properties, the sequence of directions has to be converted into a sequence of coordinates.

If there are no self-contacts, this random walk is a self-avoiding random walk.

3.1 Tasks

1. Read the code `randomwalk-2d.c` and try to understand it.

N	10	20	50	100	200	500	1000
Mean number of contacts							
Fraction of SAWs							
$\langle Re^2 \rangle$							

Table 2: Properties of 2D random walks of varying length N.

2. Make sure, that the constant `SAW` is set to 0. Compile it and run it for different parameters.
3. Look at some generated random walks by plotting the file `rw.dat` which contains the last random walk generated.
4. Complete Table 2 for two dimensional random walks.
5. How does the mean number of self-contacts per random walk and the fraction of self-avoiding walks depend on the length N of the random walks?
6. Plot the mean squared end-to-end distance versus the length N and compare the result to the one-dimensional random walk. What do you observe?

3.2 Supplementary tasks

1. Set `SAW` to 1 and recompile the code. Now, only self-avoiding walks are accepted, while non-self-avoiding walks are discarded. How does the average end-to-end distance change with N now? (Only use $N < 30$!!!)
2. Extend the code of `randomwalk-2d.c` to three dimensions. Be careful, where you have to make changes. Then, calculate $\langle Re^2 \rangle(N)$ as before.
3. Is the fraction of random walks without self-contact (SAW) changing in three dimensions?

Note: Taking only self-avoiding walks and discarding the rest *significantly* increases the time to generate random walks. Essentially this method can not be used for $N > 30$. There exist other, smarter method to generate self-avoiding random walks which are not discussed here.

4 Links

4.1 Important sampling

- http://ib.berkeley.edu/labs/slatkin/eriq/classes/guest_lect/mc_lecture_notes.pdf

4.2 Random walks

- http://www.ms.unimelb.edu.au/ tonyg/lectures/MAV2003.pdf