Tutorial

6: Simple and important sampling. Random walks.

Kai Grass, Nadezhda Gribova, J. J. Cerdà *

January 20, 2010
ICP, Uni Stuttgart

Contents

*jcerda put and at before icp.uni-stuttgart.de

1 Introduction: the GNU Scientific Library (GSL)

The GNU Scientific Library (GSL) (see `http://www.gnu.org/software/gsl`) is a collection of routines for numerical computing. The routines have been written from scratch in C, and present a modern Applications Programming Interface (API) for C programmers, allowing wrappers to be written for very high level languages. The examples given in this tutorial rely on some GSL functions, most notably random number generators.

The GSL is open source, i.e. the source code is distributed under the GNU public license. As such, the use of this library is free of charge.

The library covers a wide range of topics in numerical computing. Routines are available for numerous areas of numerical methods, such as Complex Numbers, Sorting, Random Numbers, Differential Equations, Minimization, ...

The use of these routines is described in the manual. Each chapter provides detailed definitions of the functions, followed by example programs and references to the articles on which the algorithms are based.

In addition to the broad range of functions, the most important advantage of the using GSL is that the functions are already implemented (saves you work) and checked by many people (you can rely on the correctness of the implementation).

1.1 Tasks

1. Look at the simple program `gsl-demo.c` and try to understand how to generate uniformly distributed random numbers with the aid of the GSL.

Remark: All source code of this tutorial is accompanied by a Makefile that helps you compiling it. Simply type `make` in the source folder.

1.2 Supplementary tasks

1. A nice feature of the GSL random number generator implementation is, that the method to generate random numbers can be chosen at run time. See the documentation for details on this.

Remark: You can check this by starting the demo with the following line:
`GSL_RNG_TYPE=taus GSL_RNG_SEED=123 ./gsl-demo`.

2 Calculating π

Consider a circle of diameter d surrounded by a square of length l ($l > d$). Random coordinates within the square are generated. The value of π can be calculated from the fraction of points that fall within the circle.

Estimating this fraction is quite easy: we just look at a set of points in the square and determine, what fraction of these lies in the circle. However, this method only works if the trial points are more or less uniformly distributed over the square. Uniformly distributed simply means that

the probability for a point to picked should be the same for all points of the square. Conveniently, typical random number generators, such as included in the GSL, generate uniformly distributed random numbers.

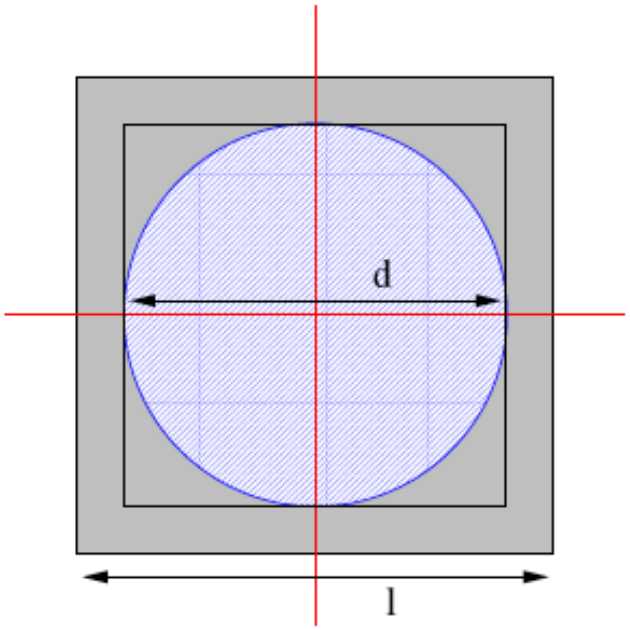

Figure 1: A circle of diameter d surrounded by a square of length l ($l > d$).

2.1 Tasks

1. How can π be calculated from the fraction of points that fall in the circle?
2. Complete the small Monte Carlo program `mc-pi.c` to calculate π using this method.

2.2 Homework 1 (2 points)

1. How does the accuracy of the result depend on the ratio l/d and the number of generated trial points? Is it a good idea to calculate many decimals of Π using this method?.

3 Numerical integration

Definition: Monte Carlo is the art of approximating an expectation value by the sampled mean of a function of simulated random variables.

3.1 Simple sampling

The MC method can be used to numerically integrate a function $f(x)$:

$$F(a,b) = \int_a^b f(x)dx = (b-a)E(f(x)),$$

where $E(f(x))$ is the expectation value of $f(x)$ on the interval $[a,b]$. This expectation value can be calculated via the Monte Carlo method:

$$E(f(x)) = \int_{x\in\chi} f_\chi(x)dx$$

Here x is drawn from a uniform distribution χ. For a finite number of samples we get the *Monte Carlo estimate* $\tilde{f}(x)$

$$E(f(x)) \approx \tilde{f}_n(x) = \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

3.2 Homework 2 (4 points)

1. How would be the generalization to two-dimensional (double) and three-dimensional (triple) integrals? **Hints: think about how you would write the expected value of the function in such cases.**

2. Use the knowledge you have now about generating random numbers in a given region of the space to make a C program that computes the Coulomb interaction $U(r) \sim 1/r$ as a function of the distance r between the centers of two spheres of radius $R = 0.5$. The spheres bear a uniform charge/mass distribution $\rho = 1/V$ where V is the volume of the sphere. Check if the interaction between such two finite bodies is equivalent to the one one would expect for two point charges/masses with q or m equal to one. Check it for the range $r \in [0, 4R]$ via plotting the value of $U(r)$ as a function of the separation between centers r. If we replace the Coulomb interaction by an interaction of the type $U(r) \sim 1/r^n$ with $n = 2,\ 3$, is the interaction of the two finite bodies equal to that of two point bodies? in which range?

   Hints: the interaction between two bodies S_1 and S_2 can be computed via the following double integral

$$U(r) = \int_{S_1} \mathbf{dr}_1\ \rho(\mathbf{r}_1) \int_{S_2} \mathbf{dr}_2\ \rho(\mathbf{r}_2)\ U(r = |\mathbf{dr}_1 - \mathbf{dr}_2|). \tag{1}$$

   Remember that a point (x,y,z) belongs to a sphere of radius R if $x^2 + y^2 + z^2 <= R^2$

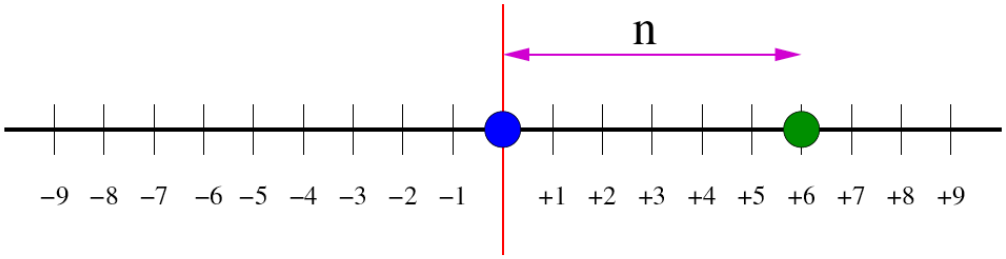

Figure 2: One dimensional random walk.

3. Have you ever heard about *the importance sampling* method. Find information about it and try to describe briefly how it works. How could such method be applied to the calculation of the integral of a function like the normal distribution $N(x) = \frac{1}{\sqrt{(2\pi)}} \exp(-\frac{x^2}{2})$ integrated over $x \in [a, b]$? Why we expect importance sampling to be tetter than simple sampling?

4 Random walk in one dimension

The second part of today's tutorial covers random walks. The random walk problem is also known as the "drunk man's problem", because a random walk consists just of a series of random steps, like the path of a completely drunk man trying to get home. In theoretical physics, random walkers are often used to model systems with little memory. A good example is the classical Brownian motion: it is in fact caused by the interactions of particles with the solvent (e.g. water), but from the observer's point of view, it seems as if the particles perform random walks.

We start with a random walker in one dimension. 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 ($p = 0.5$).

4.1 Tasks

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

2. Compile and run the sample program for different lengths of the random walk. Confirm, that on average the final position is close to the origin, i.e. the random walker is not biased.

3. Complete Table 1 with the values for the averaged squared end-to-end distance for varying lengths N of the random walk. The end-to-end distance $R_{ee}(N)$ is the distance between the starting point and the final point where the random walker is found after N steps, if the starting point is always the origen, then

$$R_{ee}^2 = pos(N)^2,$$

where $pos(N)$ refers to the position of the random walker in the N-th step.
Hint: The number of samples required for good statistics increases with N. Try to estimate how many samples you need as a function of N.

4. Plot the data of Table 1 and compare to theoretical prediction $\langle R_{ee}^2 \rangle \propto N$.

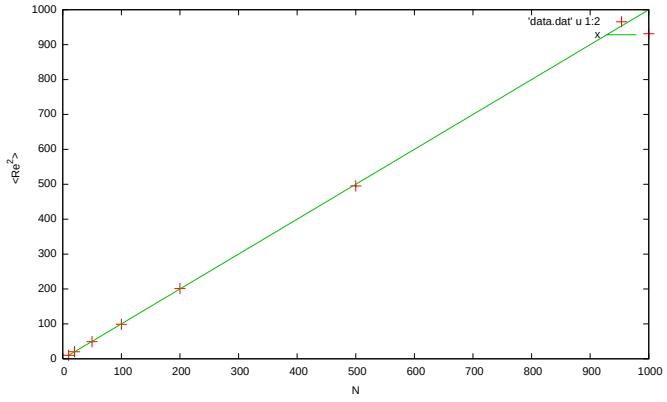

Figure 3: Averaged squared end-to-end distance of a one-dimensional random walk.

Hint: To plot the data, it is recommended to use `gnuplot` or `xmgrace`, 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`. Alternatively `xmgrace data.dat` . If you use gnuplot, you can also include the theoretical prediction into this plot by extending the command.

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

Your results should look similar to Figure 3.

N	10	20	50	100	200	500	1000
$\langle R_{ee}^2 \rangle$							

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

4.2 Homework 3 (2 points)

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.

5 Random walk in two dimensions

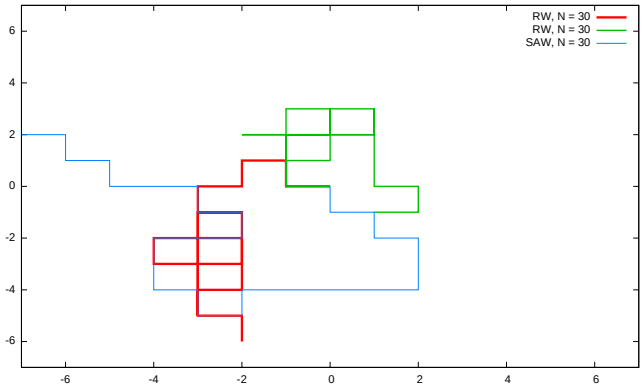

Figure 4: Sample two-dimensional random walks.

In this section, we study the two dimensional random walk. In two (or more dimensions) it is possible to distinguish two different types of random walks (see Figure 4): *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.

Just as general random walks, which represent for example Brownian motion, SAWs are used in important physical models, most notably to model flexible polymers. Such a polymer can be viewed as a chain of monomers connected by (flexible) bonds. The relative positions of two adjacent beads are essentially uncorrelated, with the exception, that monomers of course cannot overlap. This leads naturally to a SAW.

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.

5.1 Tasks

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

2. Make sure that the constant `SAW` is set to 0. Compile it and run it for different parameters, i.e. length of the walk and number of samples.

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

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

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?

5.2 Homework 4 (2 points)

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 $\langle R_{ee}^2 \rangle(N)$ 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.

6 Suggested Resources

GNU Scientific Library (GSL)

http://www.gnu.org/software/gsl/manual/html_node/

Monte Carlo techniques and Important sampling

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

Random walks

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