Tutorial

5: Simple and importance sampling. Random walks.

Kai Grass, Nadezhda Gribova, J. J. Cerdà, Marcello Sega

January 11, 2012
ICP, Uni Stuttgart

Contents

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 Homework 1 (1 point)

1. Download and install (if not already present on your computer) the latest version of GSL. Hint: after unpacking the source code archive, run the `./configure` command (check the `--prefix` option!) and then run `make` and `make install`.

2. Modify the Makefile so that the compiler can find the location of the GSL libraries (`-L` switch) and headers (`-I` switch). See `man gcc` for reference.

3. 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.

4. 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. You can check this by starting the demo with the following line:

   ```
   GSL_RNG_TYPE=taus GSL_RNG_SEED=123 ./gsl-demo.
   ```

   Report the output of this command.

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.

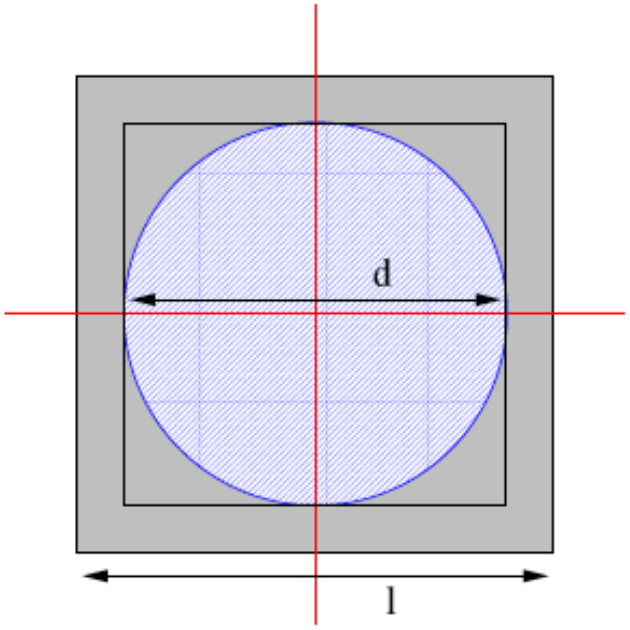

Figure 1: A circle of diameter d surrounded by a square of length l ($l > d$).

2.1 Homework 2 (2 points)

1. Complete the small Monte Carlo program `mc-pi.c` to calculate π using the method outlined above.
2. How does the accuracy of the result depend on the ratio l/d and the number of generated trial points? You can use the variance of the values of π obtained in the different cycles to estimate the accuracy.
3. Is it a good idea to calculate many decimals of π using this method? Why? List at least another method for computing π and discuss its efficiency.

The above simple MC method stems from the well-known theorem:

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

where the average value $E(f(x))$ is called here the expectation value of $f(x)$. The expectation value in the simple sampling approach is evaluated by sampling points x from a uniform distribution in the interval $[a,b]$. For a finite number of samples n the Monte Carlo estimate $\tilde{f}(x)$ of the expectation value is

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

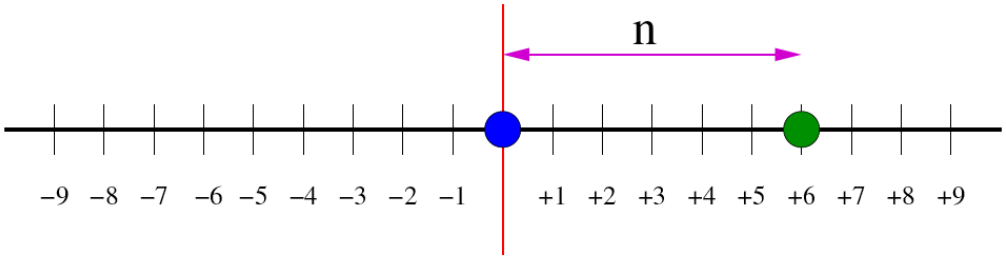

Figure 2: One dimensional random walk.

In the previous task of computing π, the function to be integrated is just defined as $f(x, y) = 1 \iff x^2 + y^2 < d^2$.

2.2 Homework 3 (1 point)

The advantage of Monte Carlo methods over regular numerical integration schemes is that the error on the estimate scales like $n^{-1/2}$, independently on the dimension of the problem. Modify the mc-pi.c code to compute the volume of the 6-dimensional sphere, calculate it, and show that the error still decreases like $n^{-1/2}$. Check your result against the analytical expression of the volume of the 6-sphere.

2.3 Homework 4 (1 point)

An improved version of the simple sampling Monte Carlo is the *importance sampling* method. Find information about it and 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 do we expect importance sampling to be better than simple sampling?

3 Random walk in one dimension

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. In theoretical physics, random walks are often employed to model systems with no memory. A good example is the classical Brownian motion.

We start with a random walk in one dimension. The random walk starts 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 ($p = 0.5$).

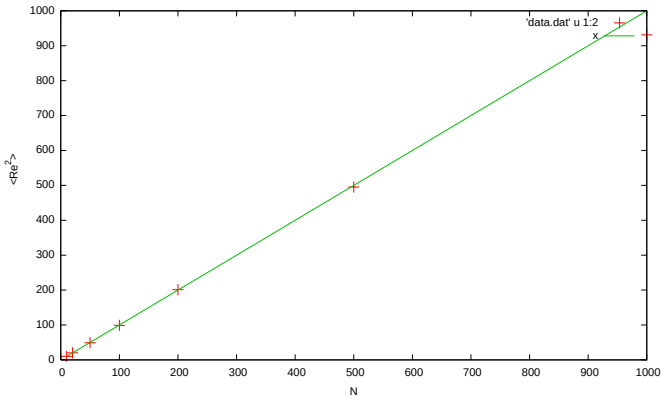

Figure 3: Averaged squared end-to-end distance of a one-dimensional random walk.

3.1 Homework 5 (2 points)

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 walk 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 walk is found after N steps, if the starting point is at the origin, then
$$R_{ee}^2 = pos(N)^2,$$
where $pos(N)$ refers to the position of the random walk 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 $\langle R_{ee}^2 \rangle \propto N$.

Your results should look similar to Figure 3.

3.2 Homework 6 (1 point)

Increase the probability for the random walk to go right to 0.8 (consequently, the probability to go left now is only 0.2). What happens now? Discuss.

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 Random walk in two dimensions

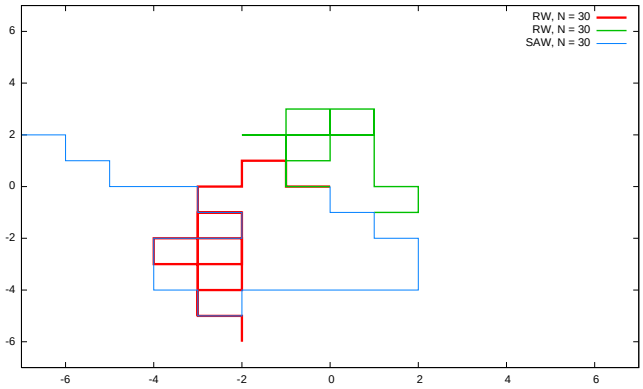

Figure 4: Sample two-dimensional random walks.

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 property that they never return to a place they have been to earlier, *i.e.* a SAW does 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. The 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.

4.1 Homework 7 (1 point)

1. Read the code `randomwalk-2d.c` and 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.
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.

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.

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?

4.2 Homework 8 (1 point)

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. Can this code become a bit more efficient by changing the way of storing the RW? How and why?

Note: Taking only self-avoiding walks and discarding the rest *significantly* increases the time to generate random walks. Practically, 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.

5 Suggested Resources

GNU Scientific Library (GSL)

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

Monte Carlo techniques and Importance 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