Fundamentals of Simulation Methods

Exercise Sheet 9

Daniel Rosenblüh, Janosh Riebesell

January 15th, 2016

Random Numbers and Monte Carlo Integration

1 Pitfalls of pseudo-random number generation

Consider the linear congruential random number generator `RANDU`, introduced by IBM in System/360 mainframes in the early 1960s. (Donald Knuth called this random number generator "really horrible", and it is indeed notorious for being one of the worst generators of all time.) The recursion relation of `RANDU` is defined by

$$I_{i+1} = (65539 \cdot I_i) \mod 2^{31}, \tag{1}$$

and needs to be started from an odd integer. The obtained integer values can be mapped to pseudo-random floating point numbers $u_i \in [0, 1)$ through

$$u_i = \frac{I_i}{2^{31}}. \tag{2}$$

(a) Implement this number generator. Make sure that you do not use 32-bit integer arithmetic, otherwise overflows will occur. (Use 64-bit integer arithmetic instead, or double precision for simplicity – its precision is sufficient to represent the relevant integer range exactly.)

(b) Now generate 2-tuples of successive random numbers from the sequence generated by the generator, i.e. $(x_i, y_i) = (u_{2i}, u_{2i+1})$. Generate 1000 points and make a scatter plot of the points in the unit square. Does this look unusual?

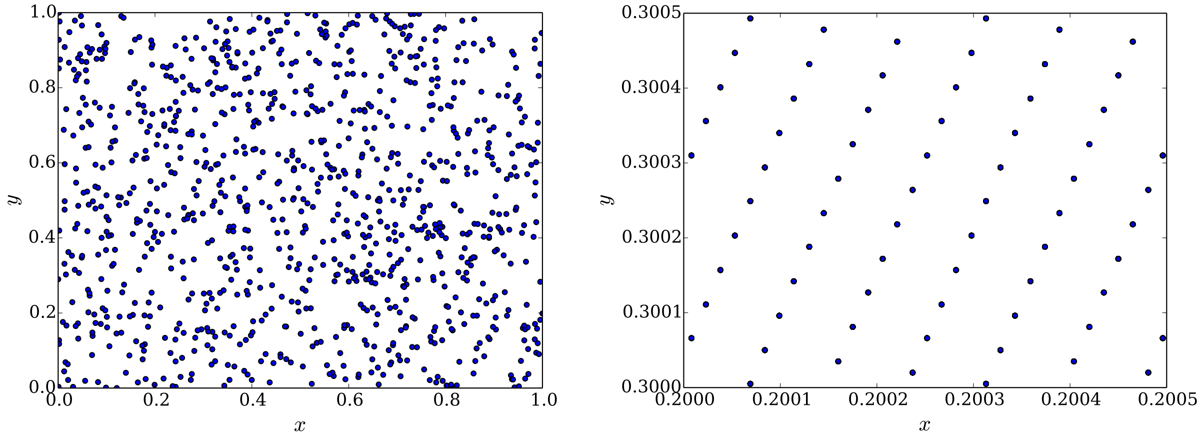
(c) Now zoom in by a large factor onto a small region of the square, for example $[0.2, 0.2005] \times [0.3, 0.3005]$, and generate enough points that there is again the same number of points within the small region as before. Interpret the result.

(d) Repeat the above for your favorite standard random number generator.

(a) See `numgen.c`.

(b) Figure 1a displays a scatter plot of 1000 points whose coordinates where generated with consecutive `RANDU` numbers scaled to the unit square. Nothing seems out of the ordinary.

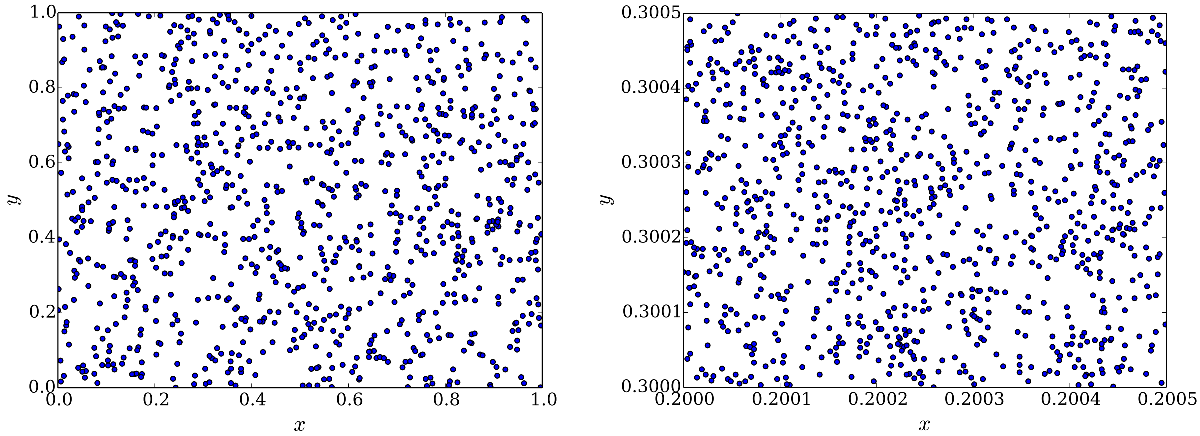
(c) Figure 1b again shows 1000 points of consecutive `RANDU` numbers, but this time restricted to the interval $[0.2, 0.2005] \times [0.3, 0.3005]$. The points display a distinct ordering without a trace of randomness.



(a) Unit square (b) Zoomed in

Figure 1: Scatter plots of 1000 points generated using the `RANDU` algorithm

(d) Figures 2a and 2b display the results of the same procedure as performed in parts (b) and (c), but this time carried out using the `drand48` algorithm.



(a) Unit square (b) Zoomed in

Figure 2: Scatter plots of 1000 points generated using the `drand48` algorithm

2 Performance of Monte Carlo integration in different dimensions

We would like to compare the performance of the Monte Carlo integration technique with the regular midpoint method. To this end, consider the integral

$$I = \int_V f(\boldsymbol{x}) \, \mathrm{d}^d \boldsymbol{x}, \tag{3}$$

where the integration domain V is a d-dimensional hypercube with $0 \leq x_i \leq 1$ for each component of the vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$. The function we want to integrate is given by

$$f(\boldsymbol{x}) = \prod_{i=1}^{d} \frac{3}{2}(1 - x_i^2). \tag{4}$$

This has an analytic solution of course, which is $I = 1$ independent of d, but we want to ignore this for the moment and use the problem as a test of the relative performance of Monte Carlo integration and ordinary integration techniques. To this end, calculate the integral for $d \in \{1, 2, 3, \ldots, 10\}$, using

(a) the midpoint method, where you divide the volume into a set of much smaller hypercubes obtained by subdividing each axis into n intervals, and where you approximate the integral by evaluating the function at the centers of the small cubes.

(b) standard Monte Carlo integration in d dimensions, using N random vectors.

For definiteness, adopt $n = 6$ and $N = 20\,000$. For both of the methods, report the numerical result for I and the CPU-time needed for each of the dimensions. (If you manage, you can also go to slightly higher dimension.)

(a) Results obtained using the midpoint method are compiled in table 1.

Table 1: Results for the integral I and the corresponding computation time t in seconds according to the midpoint method in various dimensions d

d	1	2	3	4	5	6
I	1.003 472	1.006 957	1.010 453	1.013 961	1.017 482	1.021 015
t	8×10^{-6}	4×10^{-6}	3×10^{-5}	0.000 119	0.000 666	0.004 12
d	7	8	9	10	11	12
I	1.024 56	1.028 118	1.031 688	1.035 27	1.038 864	1.042 472
t	0.027 276	0.178 799	1.1121	7.172 17	41.9002	272.923

(b) Results obtained using the Monte Carlo method are compiled in table 2.

Table 2: Results for the integral I and the corresponding computation time t in seconds according to the Monte Carlo method in various dimensions d

d	1	2	3	4	5	6
I	1.002 038	0.998 260 2	1.005 617	0.989 098 9	1.012 177	0.986 053 4
t	0.000 458	0.000 822	0.001 163	0.001 337	0.001 778	0.002 112
d	7	8	9	10	11	12
I	1.004 149	0.995 099 4	0.990 344 3	1.006 027	0.988 703 3	1.022 291
t	0.002 361	0.002 706	0.002 943	0.003 286	0.003 511	0.003 816