SF2565/FSF3565 Assignment 4

Björn Wehlin

December 2024

Stochastic simulation

In this assignment, we will investigate (numerically) the behavior of some stochastic processes, and write parallel code to do this expediently. In particular, we will consider Itô diffusion processes on $\mathbb{R}$. A variant of such a process can be described by a stochastic differential equation (SDE)

$$dX(t) = \mu(X(t))\,dt + dW(t), \qquad X(0) \sim \pi,$$

where $\mu\colon \mathbb{R} \to \mathbb{R}$ is a given function, $W(t)$ is Brownian motion, and π is some initial distribution on $\mathbb{R}$.

To simulate this process, one can employ an Euler-Maruyama stepping scheme. Fix $\Delta t > 0$ and let $X_n \approx X(n\Delta t)$. We then iterate

$$X_{n+1} = X_0 + \mu(X_n)\Delta t + \sqrt{\Delta t} Z_n, \tag{1}$$

where $(Z_n)_{n \geq 0}$ is sequence of iid standard normal random variables.

We will impose one further condition. If $X_N \leq 0$ for some N, then the process stops evolving. That is, $X_n = X_N$ for all $n > N$, where $N = \min\{n : X_n \leq 0\}$. Note that N is a random variable.

Processes like this are of interest in, for example, population modeling, where $X(t)$ is the the population at time t, and the population goes extinct if $X(t) = 0$ for some t.

We now ask the following question: what is the probability that extinction has not happened before time s? In the discrete setting, this corresponds to computing

$$\mathbb{P}(N\Delta t > s). \tag{2}$$

In practice, one can pick a number $M \gg 1$ and run the iteration (1) M times, recording the extinction time for each independent run and then simply taking the proportion of extinction times $> s$ as an estimate of the probability (2). In other words, first draw $X_0^1, X_0^2, \ldots, X_0^M$ from the initial distibution π, and then iterate

$$X_{n+1}^m = X_0^m + \mu(X_n^m)\Delta t + \sqrt{\Delta t} Z_n^m, \tag{3}$$

for each $m = 1, \ldots, M$, letting $N^m = \min\{n : X_n^m \leq 0\}$. The estimated survival probability at time s is then

$$\hat{\mathbb{P}}(\text{extinction after time } s) := \frac{1}{M}\sum_{m=1}^{M} \delta_{N^m > s},$$

where $\delta_{N_m > s} = 1$ if $N_m > s$ and zero otherwise.

Note that if we are only interested in survival up to some time s', we can stop the iteration after time s' has passed, even though the process has not reached 0.

Task I: Serial implementation

As a warmup, you will implement the algorithm in a single-threaded fashion.

Let $\mu(x) = -b$ for $b > 0$ and take π to be the Gamma distribution with shape $\alpha = 2$ and rate $\beta = 1/b$.

In order to get access to random numbers, you will need `#include <random>`. There are three ingredients to random number generation: seed, generator, and distribution. The seed can be a number of your choice, or you can use `std::random_device` to get (hopefully) different results for each run. For the generator, you can use, for example, `std::mt19937_64`.

For the Gamma distribution, use `std::gamma_distribution`. A complete example with seeding the generator and drawing random numbers from the Gamma distribution is available at `https://en.cppreference.com/w/cpp/numeric/random/gamma_distribution`.

- Start by picking $b = 1$. Plot a survival curve (s on the x-axis, and the survival probability at time s on the y-axis).
- Try the same for some different values of b. What can you say about the survival probability as b increases, in relation to the underlying stochastic process?
- The survival probability should follow an exponential distribution[1], which has density

$$f(s) = \lambda e^{-\lambda s},$$

  for some $\lambda > 0$. Does this appear to be the case? (Can you identify λ?)

Throughout, use a reasonably large M, say 10,000, and small Δt, say 0.001.

Task II: Parallel implementation

For this part, we will run with a much larger M, say 1,000,000, or however much is necessary so that the serial implementation spends at least a few seconds for the computation.

Your task now is to parallelize the implementation using threads. The parallelization scheme is up to you. You may, for example, compute M/p Euler-Maruyama steppings on each thread if you have p threads, but you are free to take a different approach if you think it will be more efficient.

Keep in mind that the random number generator is not thread-safe (it holds a state that gets updated with each random draw). One approach is to lock a mutex during the draw, but this will be very inefficient, so it is recommended instead that you keep one random generator per thread, or per m.

[1]If you are interested in the theory behind this, see for example: Sylvie Méléard. Denis Villemonais. "Quasi-stationary distributions and population processes." Probab. Surveys 9 340 - 410, 2012. `https://doi.org/10.1214/11-PS191`

- Complete the implementation of the parallel code.
- Verify that the parallel code gives similar results as the serial implementation.
- Measure and plot the speedup running on different numbers of threads against running on one thread. Fit an Amdahl curve to your data. What is your proportion of the program that can benefit from adding more processors? *Hint: to get a good Amdahl fit you may have to use only a portion of your data! Why is this?*

Task III: Consistency

(This task is optional for Master's students.)

When we write parallel programs, we would like them to give the same output regardless of how many processors are used. Can you come up with and implement a scheme to ensure that this is the case? Run with a small M, say 50, and try different numbers of threads to verify that your implementation works as intended.

Submission

The programming exercises should be done individually, or in groups of two. Hand in a report containing:

- Comments and explanations that you think are necessary for understanding your program. (But do not comment excessively.)
- The output of your program according to the tasks. Don't forget to draw conclusions!
- Printout of the source code in PDF format. (We need this to be able to comment on your code, points will be deducted for missing to do this.) You can either put the source code as an appendix to your written report, or you can hand in separate PDF(s).

In addition, all source code for your program(s) in .zip/.tar format and instructions for how to compile/run.

Important! Your submission must contain at least the following two files:

- PDF report (not zipped!) with your findings and source code
- .zip/.tar archive containing your source code

Submissions not following this format will not be accepted and will have to be resubmitted (these will be considered late submissions).

Good luck!