DTU 02635 Fall 2016 — Assignment I

This is the first of two assignments in this course in which you will write a short program and a short report. The purpose of this assignment is to train and assess your ability to:

- design, implement, and document a program that solves a mathematical problem
- debug and test mathematical software.

Please note that this assignment will count towards your final grade (approximately 10% of final grade).

Practical information

You may work on this assignment alone or in small groups of 2-3 students, but you must hand in an **individual report** regardless of whether you choose to work alone or as part of a small group. You may hand in your group's code if the code was written collaboratively by you and your group.

Deadline

Your **report** and **code** must be submitted via CampusNet no later than on **October 26, 2016**.

Report

Your report should be short and concise and **no longer than one A4 page**. If necessary, you may include a one-page appendix with code snippets. Your report should also include either a front page or a header/footer with the following information:

- course number and date
- name and student number
- group members (if applicable).

You must submit your report as a **PDF file**.

Code

You must submit your code as a single **zip file** that contains all **source files and a makefile**. You are encouraged to test your code and your makefile on the DTU Unix system (e.g. via ThinLinc) to make sure that there are no compilation errors.

Your code will be evaluated based on:

- program structure
- readability (use comments in the code, indentation, etc.)
- correctness and error checking.

Questions

Please post any questions that you may have about the assignment on Piazza.

Background

Random number generators

Random numbers play an important role in computer simulations of stochastic processes and are used extensively in many branches of mathematics, science, and engineering. In this assignment, you will implement a couple of simple methods for generating pseudo-random numbers from uniform, exponential, and Poisson distributions using the pseudo-random number generator `rand()` in the standard library (`stdlib.h`). The `rand()` function generates and returns an integer from 0 to $I_{\max}$ where (ideally) the $I_{\max} + 1$ possible outcomes have approximately equal probability. The maximum value $I_{\max}$ for the `rand()` function is defined as `RAND_MAX` in `stdlib.h`.

The term *pseudo-random* alludes to the fact that the generated numbers appear to be random, but they are in fact deterministic. The output of the pseudo-random number generator is determined by the generator's *state*. The initial state of the generator is set using a so-called *seed*, and this can be changed using the function `srand()` which is also defined in the standard library. The prototype is `void srand(unsigned int seed)`.

Uniform distribution

Given a random integer I between 0 and $I_{\max}$, you can approximate a continuous uniform distribution on $[0, 1]$ using the transformation

$$U = \frac{I}{I_{\max}}$$

where I is a pseudo-random integer. Similarly, to obtain an approximately uniform distribution on $(0, 1)$ (i.e., excluding the values 0 and 1), we can use the transformation

$$U = \frac{I + 1}{I_{\max} + 2}.$$

Exponential distribution

The exponential distribution is a continuous distribution with the probability density function (pdf)

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0. \end{cases}$$

The exponential distribution can be used to describe the time between events, and hence the parameter λ is sometimes called the arrival rate or the event rate. The mean of the exponential distribution is equal to λ^{-1}.

The so-called cummulative distribution function (cdf), which for the exponential distribution is given by

$$F(x) = \int_{-\infty}^{t} f(t)\, dt = \begin{cases} 1 - e^{-\lambda x} & x \geq 0 \\ 0 & x < 0, \end{cases}$$

can be used to generate samples from an exponential distritution using the so-called *inverse transform sampling*. Specifically, given a uniformly distributed random variable $U \in (0, 1)$, the random variable

$$Z = F^{-1}(U) = -\frac{\log(1 - U)}{\lambda}$$

will be exponential.

Remark: The "quality" of random numbers generated using this method depends on the "quality" of the pseudo-random number generator used to generate samples from a uniform distribution. This is outside the scope of this course, but keep in mind that the `rand()` function in the C standard library is system-dependent and is not guaranteed to be a high-quality random number generator. In fact, according to the C standard, `RAND_MAX` may be as low as $2^{15} - 1 = 32767$.

Poisson distribution

The Poisson distribution is a discrete probability distribution that models the number of events Y that occur within some time interval (say, 1 second), and the time between events follows an exponential distribution. Given a positive parameter λ, the probability that a Poisson random variable Y takes the value k is given by

$$\mathbf{Pr}(Y = k) = \frac{\lambda^k e^{-\lambda}}{k!}, \quad k = 0, 1, 2, \ldots,$$

where $\lambda > 0$ is equal to the expected value of Y.

We will investigate two methods that can be used to generate pseudo-random numbers from a Poisson distribution with parameter λ.

Method 1

Since the Poisson distribution can be viewed as the number of events in a time interval, we can generate a random number from a Poisson distribution by simulating the events where the time between events follows an exponential distribution with parameter λ. In other words, we simply generate samples $z_1, z_2, z_3, \ldots$ from an exponential distribution with parameter λ until

$$z_1 + z_2 + \cdots + z_k \leq 1, \quad z_1 + z_2 + \cdots + z_k + z_{k+1} > 1.$$

This corresponds to k events occurring within a 1 second interval and the $(k+1)$th event occurring after the interval. The following pseudo code summarizes the method:

Require: $\lambda > 0$
  $t = 0$, and $k = 0$.
  repeat
    Generate a sample z from an exponential distribution with parameter λ
    $t := t + z$
    $k := k + 1$
  until $t \geq 1$
  return $k - 1$

Method 2

The second method that we will investigate originates from a paper by A. C. Atkinson from 1979. It is based on a technique known as *rejection sampling*. A variant of the method can be summarized in pseudo-code as follows:

Require: $\lambda > 0$
  $\beta = \pi/\sqrt{3\lambda}$, $\alpha = \beta\lambda$, and $c = 0.767 - 3.36/\lambda$
  repeat
    repeat
      Generate a sample u_1 from a uniform distribution
      Set $x = \beta^{-1}(\alpha - \log((1 - u_1)/u_1))$
    until $x \geq -0.5$
    Set $k = \lfloor x + 0.5 \rfloor$ (i.e., "floor" of $x + 0.5$)
    Generate a sample u_2 from a uniform distribution
    $v_1 = \alpha - \beta x + \log(u_2) - 2\log(1 + \exp(\alpha - \beta x))$
    $v_2 = \log c - \lambda - \log\beta + k\log(\lambda) - \log(k!)$
  until $v_1 \leq v_2$
  return k

Note that this method is based on an approximation that is not very accurate for small values of the parameter λ. A general recommendation is that this method should be avoided when $\lambda < 30$. Note also that when implementing this method, the term $\log(k!)$ must be computed with care to avoid overflow. Specifically, we can avoid integer overflow by noting that $k! = \Gamma(k+1)$, and hence $\log(k!) = \log(\Gamma(k+1))$ where the right-hand side is the so-called log-Gamma function, evaluated at $k + 1$. The log-Gamma function is available as `lgamma()` in `math.h`.

Assignment

Code

Write a program that generates a two-dimensional array of size $m \times n$ with independent random numbers from either an exponential distribution with parameter λ or from a Poisson distribution with parameter λ. Your program should:

- prompt the user to enter the size (m and n) and choose the desired pseudo-random distribution (exponential or Poisson distribution), including the distribution parameter λ
- print the two-dimensional array with random numbers
- use dynamic memory allocation for arrays.

Report

Write a short report (see requirements under "Practical information") in which you briefly address the following questions:

1. Program structure: how did you organize your code? Where is memory allocated/deallocated? Which data types did you choose and why?
2. How did you test your code to verify that it is correct? Does your code work as expected for small/large values of λ (say, $\lambda \approx 0.1$ or $\lambda \approx 10^6$)?
3. Compare method 1 and method 2 for generating samples from a Poisson distribution. What are the advantages/disadvantages of the two methods? Did you use both methods in your program or only one of the methods?