TP 0 - Stochastic processes

Metaheuristics for optimization

September 20, 2021

S

Throughout the semester, you will have to implement algorithms that require efficient event generation with given probabilities. This series is an introduction to several possible methods. The problem is always the following: *let us consider N possible events occurring with probability P_i, $i \in \{0, ..., N-1\}$.* Generate a sequence of events such that each event occurs with the appropriate probability. The problem is not so trivial, as we will see through examples.

Simulation of a balanced dice

First, we simulate rolling a N-face balanced die. Each side of the die has a $1/N$ probability of being obtained.
For each roll of the die, generate a random number $r \in [0, 1)$, and convince yourself that the event is $i = \lfloor rN \rfloor$. Events can be visualized as "drawers" in the space of possible random numbers. This case is very simple, because it is assumed that the operations involved can be performed in $O(1)$.
Generate a long sequence of rolls of a six-sided die, and check that the frequency of appearance of each side is $\frac{1}{6}$.

Simulation of a biased dice using a balanced dice

Suppose the die is not balanced, but each face comes out with probability P_i. The above method no longer works, but if the probabilities are commensurable, it can be recycled by dividing each drawer into sub-drawers of constant size.
If, for example, we simulate 6 events with $\vec{P} = (1/4, 1/4, 1/4, 1/8, 1/16, 1/16)$, we divide the segment $[0, 1)$ in drawers of length $1/16$, then we use the previous procedure to generate a drawer in $O(1)$. Note that this involves a pre-analysis step: we must store an array that indicates which real event the generated drawer corresponds to. Here the array is of length 16, but it can become catastrophically cumbersome if the least common multiplier of the denominators D_i of the P_i is very large (in the worst case the complexity is $O(\prod_{i=0}^{N-1} D_i)$). This procedure is therefore not recommended in general, although the generation itself remains in $O(1)$.
Implement this method to simulate the generation of events whose probabilities of occurrence are non-uniform.

Simulation of a biased coin toss

A coin has a probability p of falling on 'tail' and a probability $1-p$ of falling on 'head'. When $p \neq 0.5$, the coin is biased. To simulate the throw of a coin, generate a random number $r \in [0,1)$, and use the variable $\lambda = \lfloor r+p \rfloor$. What are the possible values of λ, and with what probabilities are they obtained? Finally, use the following variable $x = \lambda a + (1-\lambda)b$ (a for 'tail' and b for 'head') to verify that the part you generated is biased with the p probability you defined.

Simulation of a biased dice using a biased coin

Consider the following example: we want to simulate six events with $\vec{P} = (1/4, 1/4, 1/4, 1/8, 1/16, 1/16)$. We start by flipping a coin with $p = \frac{1}{4}$. If it falls on "stack", we are done, and we return event "1". Otherwise, we normalize the probabilities of the remaining events by the remaining mass (of probability) $1 - \frac{1}{4} = \frac{3}{4}$. So, $\vec{P}_{temp} = (1/3, 1/3, 1/6, 1/12, 1/12)$. We now toss a coin with $p = \frac{1}{3}$. If it falls on "tails", we have finished, and we return event 2. Otherwise, we normalize the probabilities of the remaining events by the mass of remaining probability, *etc.* Implement this method to simulate the generation of events whose probabilities of occurrence are non-uniform.

Roulette method

The last method considered in this TP is the roulette method. Given N events and their probabilities P_i, $i \in \{0, ..., N-1\}$, the initialization step consists in calculating the cumulative probabilities $P_i^{cumul} = \sum_{j=0,...,i} P_j$, $i \in \{0, .., N-1\}$. Generate a random number $r \in [0,1)$, and convince yourself that the event performed is the smallest i such as $P_i^{cumul} > r$.

Implement the latter method and check that for a large number of events generated, the frequency of occurrence of each event corresponds to the probability associated with it.

Report

For this series, collect your code and a short report **(2-3 pages)** will upload on moodle, by Sunday, September 27, 2020. This short report must include: (1) the description of each exercise, and (2) the methodology/pseudo code adopted to solve it, as well as (3) results obtained with your own code.

NB: For this series and all subsequent ones, only codes in C/C++, python, rust, java and matlab will be accepted. Codes rendered in other languages will not be taken into account. In addition, the use of LaTeX (https://openclassrooms.com/fr/courses/1617396-redigez-des-documents-de-qualite-avec-latex) is strongly encouraged. Finally, graphics that do not include a title or legends for the axes will not be taken into account.