Université de Genève

Metaheuristics for optimization

TP 0: Stohastic processes

Author: Joao Filipe Costa da Quinta

E-mail: Joao.Costa@etu.unige.ch

September 25, 2021

Simulation of a balanced dice

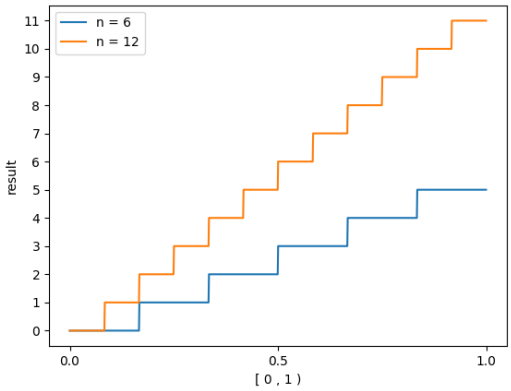
Intro and methodology

During this exercise we will be simulating rolls of a N-face balanced dice. This means that the dice has N possible outcomes at each throw, all faces are equally likely with a probability of 1/N.
For each roll we need to do the following tasks:

(1) compute a new random value $r \in [0, 1)$, done with random() function from random package

(2) compute $i = \lfloor N * r \rfloor$, done with floor() function from math package

After doing the three tasks, we get a value, that is the simulated roll.
The following graphic represents this phenomenon for N = 6, 15



Values for N = $\{6, 12\}$

In the image above we see this kind of step function, this is due to the fact that we use the function .floor(). The function that does this computation can be found in the functions.py file, and its signature is the following:

balancedDice(N), where N corresponds to the number of faces in the balanced dice.

To check if the function works well we simply use it a large amount of times, or n times, if all faces are represented the same amount of times we know it is a good simulation. Obviously we need n to be large enough.

Results

For our results we will run the script with N = 6, and n = $\{100, 1000, 10000\}$.
To 'read' the results, we will be computing the frequency of each face. Let's say the face represented by the value 1 was seen 20 times out of 100 rolls, then its frequency is 20/100 = 0.2.

Face	1	2	3	4	5	6
100 rolls	0.18	0.21	0.15	0.18	0.15	0.13
1000 rolls	0.184	0.17	0.148	0.165	0.167	0.166
10000 rolls	0.1742	0.167	0.1671	0.1663	0.1641	0.1613

Frequency of each face for different number of rolls

The perfect frequency is $(n/N)/n = 0.166$ for $N = 6$ and $n = \{100, 1000, 10000\}$. We can easily see that we get closer to the perfect frequency as n becomes larger.

Simulation of a biased dice as a balanced dice

Intro and methodology

A biased dice is a N-face dice, where all faces are equally likely. Which means we can't use the same function as we did in the previous exercise.
The solution is to transform the N-face biased dice into a N'-face balanced dice. However, this can only be done if the probabilities are commensurable.
To do this we have to:

(1) find the smallest probability of the biased dice $\longrightarrow p_{min}$

(2) divide the segment $[0, 1)$ into $1/p_{min} = N'$ divisions, which means that we now have a N'-face balanced dice

(3) assign the different faces of the N'-face dice to the previous N-face biased dice

N-face biased dice	1				2				3				4		5	6
N'-face balanced dice	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Assignment of N'-face balanced dice to N-face biased dice

With this table it becomes easy to the exercise. After computing the N'-face balanced dice, we simply use the function from the previous exercise.

Results

For our results we will run the script with 6-face biased dice of probability P = $[1/4, 1/4, 1/4, 1/8, 1/16, 1/16]$. 1/16 is the smallest probability, which means our balanced dice has 16 faces of 1/16 probability each. We will do n = $\{100, 1000, 10000\}$ number of rolls.
To 'read' the results, we will be computing the frequency of each face. Let's say the face represented by the value 1 was seen 20 times out of 100 rolls, then its frequency is $20/100 = 0.2$.

Face	1	2	3	4	5	6
100 rolls	0.25	0.25	0.2	0.17	0.08	0.05
1000 rolls	0.253	0.234	0.258	0.137	0.052	0.066
10000 rolls	0.2541	0.2511	0.2417	0.1209	0.0657	0.0665

Frequency of each face for different number of rolls

Face	1	2	3	4	5	6
Perfect Frequency	0.25	0.25	0.25	0.125	0.0625	0.0625

Perfect frequency for each face

The perfect frequency is different for each face, since they have different probabilities. We can easily see that we get closer to the perfect frequency as n becomes larger.

Simulation of a biased coin toss

Intro and methodology

A coin toss has 2 possible outcomes, tail with probability p, and head with probability 1-p. For the coin to be balanced, p needs to be 0.5, any other value and the coin is biased.
For each coin toss we need to do the following tasks:

(1) compute a new random value $r \in [0, 1)$, done with random() function from random package

(2) compute $\lambda = \lfloor p + r \rfloor$, done with floor() function from math package

λ has two possible outcomes $\{0, 1\}$ 0 occurs when $p + r < 1$ and 1 occurs when $1 \leq p + r$. Other values are never possible, as $r < 1$ and $0 \leq p \leq 1$, which means $p + r < 2$ for every p and r.
If $\lambda = 0$ then the coin toss resulted in head, if $\lambda = 1$ then the coin toss resulted in tail.

Results

For our results we will run the script with p = 0.3 biased probability. We will do n = $\{100, 1000, 10000\}$ number of tosses.
To 'read' the results, we will be computing the frequency of each face. Let's say the face represented by the value 1 was seen 20 times out of 100 rolls, then its frequency is 20/100 = 0.2. The perfect frequency for tail corresponds to p = 0.3, and the perfect frequency to head corresponds to 1 - p = 0.7.

Face	Tail	Head
100 tosses	0.39	0.61
1000 tosses	0.306	0.694
10000 tosses	0.3028	0.6972

Frequency of tail / head for p = 0.3

Just like in the previous exercises, a larger number of tosses/rolls, results in better results.

Simulation of a biased dice using a biased coin

Intro and methodology

In this exercise we will simulate a N-face biased dice. However, this time we will do it by using the coin toss function from exercise 3. Let P = $[1/4, 1/4, 1/4, 1/8, 1/16, 1/16]$. 1/16 be the probability of the 6-face biased dice we want to simulate, to do this we need to:

(1) toss a coin with probability $p = P_0 = 1/4$

(2) if the result is tail, then the toss was a success, and the the result of the dice throw is 1

(3) if the result is head, then we need to remove 1/4 of our probability mass, and normalize the probabilities we have left

(4) toss a coin again, now with the new normalized probability for $p = P_1$

(5) if the result is tail, then we return 2, if it is head, we do point (3) again.

Face	1	2	3	4	5	6
Probability	1/4	1/4	1/4	1/8	1/16	1/16
toss n.1	**FAIL**	(1/4)/(3/4)=1/3	(1/4)/(3/4)=1/3	(1/8)/(3/4)=1/6	(1/16)/(3/4)=1/12	(1/16)/(3/4)=1/12
toss n.2		**FAIL**	(1/3)/(2/3)=1/2	(1/6)/(2/3)=1/4	(1/12)/(2/3)=1/8	(1/12)/(2/3)=1/8
toss n.3			**FAIL**	(1/4)/(1/2)=1/2	(1/8)/(1/2)=1/4	(1/8)/(1/2)=1/4
toss n.4				**FAIL**	(1/4)/(1/2)=1/2	(1/4)/(1/2)=1/2
toss n.5					**FAIL**	(1/2)/(1/2)=1

Normalization example of a 6-face biased dice of probability P = $[1/4, 1/4, 1/4, 1/8, 1/16, 1/16]$

The normalization process is done in the following steps:

(1) remove the failed toss from the probability mass, for example, if the first throw fails , we have $1-1/4 = 3/4$

(2) compute a new P_new such that : $P\_new_i = P_i/(3/4)$

Results

For our results we will run the script with 6-face biased dice of probability P = $[1/4, 1/4, 1/4, 1/8, 1/16, 1/16]$. We will do n = $\{100, 1000, 10000\}$ number of rolls.
To 'read' the results, we will be computing the frequency of each face. Let's say the face represented by the value 1 was seen 20 times out of 100 rolls, then its frequency is $20/100 = 0.2$.

Face	1	2	3	4	5	6
100 rolls	0.33	0.22	0.19	0.1	0.08	0.08
1000 rolls	0.245	0.259	0.261	0.12	0.057	0.058
10000 rolls	0.25	0.2547	0.245	0.1249	0.0633	0.0621

Frequency of each face for different number of rolls

Face	1	2	3	4	5	6
Perfect Frequency	0.25	0.25	0.25	0.125	0.0625	0.0625

Perfect frequency for each face

The perfect frequency is different for each face, since they have different probabilities. We can easily see that we get closer to the perfect frequency as n becomes larger.

Simulation of the Roulette method

Intro and methodology

This is yet another method in which we can simulate a N-face biased dice.
This method is very easy to understand, let's see what the steps are:

(1) from probability P, compute P_comul where P_comul_i is equal to the sum of every probability P_j such that $0 \leq j \leq i$

(2) compute a new random value $r \in [0, 1)$, done with random() function from random package

(3) check which $P\_comul_i < r$, starting from i = 0, the first P_comul_i that respects this condition, is the i we return

P	1/4	1/4	1/4	1/8	1/16	1/16
P_comul	1/4	2/4	3/4	7/8	15/16	16/16 = 1

Computation of P_comul for P = [1/4, 1/4, 1/4, 1/8, 1/16, 1/16]