First Week Assigment

[redacted]

April 24, 2020

1 The secretary problem

The secretary problem is a problem that demonstrates a scenario involving optimal stopping theory. The basic form of the problem is the following: imagine an administrator who wants to hire the best secretary out of N rankable applicants for a position. We have to choose the best possible candidate from N applicants so that we can only once (or not at all) interview each applicant and have to decide on the spot. The strategy is that we refuse the first k applicants irrespective of their performance and then we accept the first applicant that outperforms the best of the first k. If none do we have to employ the very last applicant.

1.1 Task

- What is the $R_1 = k/N$ ratio that maximizes the chance that we employ the best applicant?
- What is the $R_2 = k/N$ ratio that maximizes the average quality (rank) of the accepted applicant.

1.2 Results

We expect to get for R_1 some value close to the analytic $\frac{1}{e} \simeq 0.36$ value. For R_2 we expect something similar that is because we have less strict limitation. From the simulation we get the following results:

- $R_1 \simeq 0.32$ (ratio that maximizes the chance that we employ the best applicant)
- $R_2 \simeq 0.09$ (ratio that maximizes the average quality (rank) of the accepted applicant)

See Figure 1

1.3 Execution

Compilation and execution of the contents of the 01_*Secretary_problem* folder:

- python3 secretary.py
- gcc secretary.c -o secretary

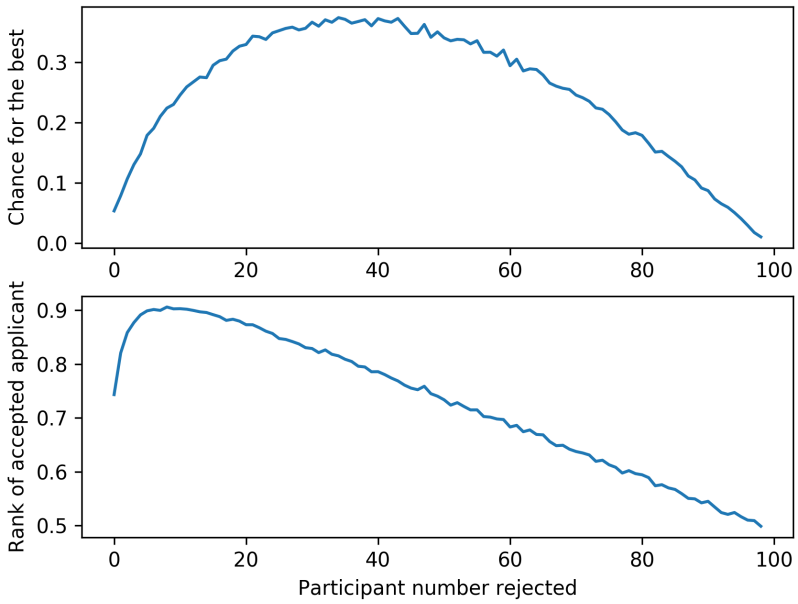

Fig imulat study he se ary It as inves wo sep te problem he str w max izes e t chance t t we em-ploy th t ap can and t ra gy t imiz the aver quality (ra k) of e a pted ppl nt. Th gu can be r oduced compiling and running the 01_*Secretary_problem/secretary.c* program and plotting with 01_*Secretary_problem/secretaryPlot.py*. The parameters used in simulation are the following: number of participants $N = 100$ and sampeling size $smplsize = 10000$

- ./secretary

Plotting with python script into secretary.png:

- python3 secretaryPlot.py

2 Non transitive dice

Two players play dice. The first player picks randomly any dice from a set of several dice and throws. The second player picks according to the right strategy his/her dice to throw and he/she will always win the game (on the long run).

We have two type of dices:

- Case 1:

$A : 2, 2, 6, 6,$
$B : 1, 1, 5, 5, 9, 9$
$C : 3, 3, 4, 4,$

- Case 2:

$A : 4,$ $4, 0$
$B :$ $3, 3$
C $, 6,$ $2, 2$
$D : 5, 5,$ $1, 1,$

2.1 Task

- Determine the strategy of the second player for the two sets of specially numbered dice above (Case 1 and 2) and find analytically the ratio of games won.
- Demonstrate this finding by simulation.

2.2 Results

2.2.1 Case 1 Analytical

$A : 2, 2, 6, 6, 7, 7$
B $5, 5, 9,$
$3, 3, 4, 4, 8$

We have to check all possible outcomes:
From studying table 1, we get:

$$P(C > A) = P(A > B) = P(B > C) = \frac{5}{9}$$

2.2.2 Case 1 Simulation

The simulations gives similar results:

	Analytical Re	Simulation Result
$P($ $> A)$	$\frac{5}{9} = 0.(5)$	0.56
$P($ $)$	$\frac{5}{9} = 0.(5)$	0.55
$P(B >$ $)$	$\frac{5}{9} = 0.(5)$	0.55

For execution instructions see subsection 2.3. (Can be reproduced by running $01\_NonTransitive\_Dice/NTdice.py$ with $smplsize = 10000$. Running time is about 1 sec.)

Table 1: Possible outcomes (We write in the table in each case the player which will win the game)

(a) C beats A

C \ A	2	6	7
3	C	A	A
4	C	A	A
8	C	C	C

(b) A beats B

A \ B	1	5	9
2	A	B	B
6	A	A	A
7	A	A	B

(c) B beats C

B \ C	3	4	8
1	C	C	C
5	B	B	C
9	B	B	B

2.2.3 Case 2 Analytical

$$A : 4, 4, 4, 4, 0, 0$$
$$B : 3, 3, 3, 3, 3, 3$$
$$C : 6, 6, 2, 2, 2, 2$$
$$D : 5, 5, 5, 1, 1, 1$$

Here we can similarly check possible outcomes:
From table 2:

$$P(A > B) = P(B > C) = P(C > D) = P(D > A) = \frac{2}{3}$$

2.2.4 Case 2 Simulation

The simulations gives similar results:

	Analy[illegible]ult	[illegible]lation[illegible]ult
$P(A > B)$	$\frac{2}{3} = 0.(6)$	0.669[illegible]
$P(B > C)$	$\frac{2}{3} = 0.$[illegible]	.666[illegible]
$P(C > D)$	$\frac{2}{3} = 0.($[illegible]	661[illegible]
$P(D > A)$	$\frac{2}{3} = 0.(6)$	0.6693

For execution instructions see subsection 2.3. (Can be reproduced by running $01\_NonTransitive\_Dice/NTdice.py$ with $smplsize = 10000$. Running time is about 1 sec.)

2.3 Execution

Compilation and execution of the contents of the $01\_NonTransitive\_Dice$ folder:

- python3 NTdice.py

Table 2: Possible outcomes (We write in the table in each case the player which will win the game)

(a) A beats B

A \ B	3	3	3
4	A	A	A
4	A	A	A
0	B	B	B

(b) B beats C

B \ C	6	2	2
3	C	B	B
3	C	B	B
3	C	B	B

(c) C beats D

C \ D	5	5	5	1	1	1
6		C	C		C	
6		C			C	
2					C	
2			D		C	
		D	D		C	
		D	D		C	

(d) D beats A

D \ A	4	4	4	4	0	0
5	D	D	D	D	D	D
5	D	D	D	D	D	D
5	D	D	D	D	D	D
1	A	A	A	A	D	D
1	A	A	A	A	D	D
1	A	A	A	A	D	D

- gcc NTdice.c -o NTdice
- ./NTdice

3 Random walk

A random walk is a random process that describes a path that consists of a sequence of random steps in some mathematical space. An elementary example of a random walk is the random walk on the integer number line (1 dimensional), which starts at 0 and at each step moves $+1$ or -1 with p_+ and p_- probability.

3.1 Task

Finding the relationship between the average squared distance made by the walker and the number of steps.

$$\langle d^2 \rangle_N = N^\alpha$$

$$\sigma^2 = \langle d^2 \rangle_N - \langle d \rangle_N$$

- Find α for $p_+ = p_-$ (Study the average square distance for simple one dimensional random walk)
- Find α for $p_+ \neq p_-$ (Study the average square distance for biased random walk in one direction)
- Study σ^2 (Study the behavior of the variance)

3.2 Results

As we decrease the probability of going in the left direction, the α parameter increases. The variance vs. step number is a straight line in every case (see Table 3 and Figure 2).

	α
[illegible]	1.00
30%	[illegible]59
[illegible]	[illegible]89
0%	2.00

Table 3: Fitted parameter to the biased random walk. This can be reproduced by compiling and running 01_*Random_walk*/*randomwalk.c* and/or 01_*Random_walk*/*randomwalk.py* and 01_*Random_walk*/*randomwalkPlot.py*, with the parameters $N = 100$, $smplsize = 1000$ and $p = 0.5, 0.3, 0.1, 0.0$. Running time is less than 1 minute.

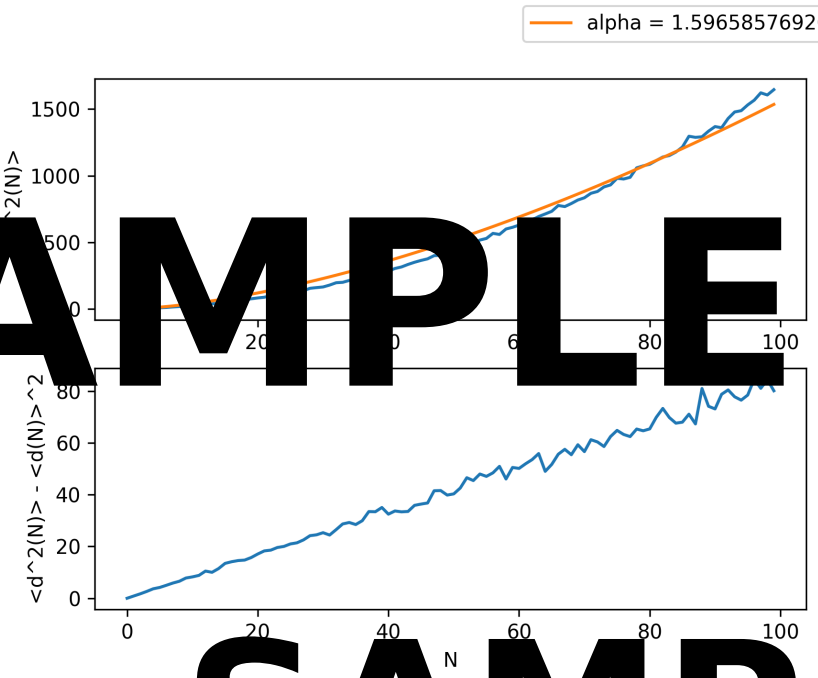

Figure 2: Simulation results for a specific case random walk. On the figures we can see the average squared distance made by the walker vs. the number of steps and the variance in the same configuration. (The parameters used in the simulation are the following: $N = 100$, $smplsize = 1000$, $p_- = 0.3$) One can reproduce the figures by compiling and running 01_*Random_walk*/*randomwalk.c* and/or 01_*Random_walk*/*randomwalk.py* and 01_*Random_walk*/*randomwalkPlot.py*

3.3 Execution

Compilation and execution of the contents of the 01_*Random_walk* folder:

- python3 randomwalk.py
- gcc randomwalk.c -o randomwalk
- ./randomwalk

Plotting with python script into randomwalk.png:

- python3 randomwalkPlot.py

In the pythonish implementation we made a little bit cheating because the N step random walk is not independent of the $N-1$ step random walk and because of this the curve for the same sampeling size is smooter in the pythonish way.

4 Dice throwing

4.1 Task

- Show that the distribution is uniform (1/6)
- The variance goes to zero
- How fast?

4.2 Results

In Figure 3, the smoothing of the histogram is visible after increasing the number of throws. Figure 3 also shows, that the variance vs. number of throws appears as an approximate line on a log-log scale, therefore we can conclude that the data follows a power law decay.

4.3 Execution

Compilation and execution of the experiments in the (*Dice_throw*

- python3 dice.py
- gcc dice.c -o dice
- ./dice

Plotting with python script into dice.png:

- python3 dicePLOT.py

5 Dependecies

- gcc 4:7.4.0-1ubuntu2.3
- python 3.6.7-1 18.04 (scipy 1.3.1, numpy 1.17.2)

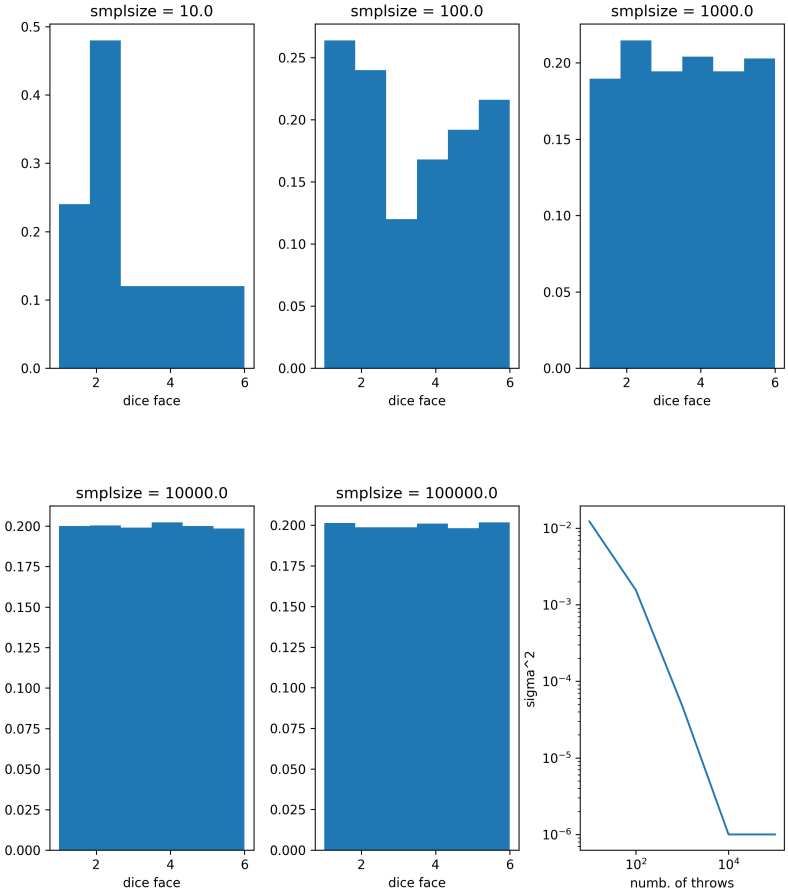

Figure 3: [redacted] the distribution of the dice face[redacted] On the first five figure we can see histograms for $10, 100, 1000, 10000, 100000$ throws. The last fugure shows that the variance vs. number of throws on log-log scale[redacted] which indicates a power law decay[redacted] running the 01_*Dice_throwing/dice.c* and/or 01_*Dice_throwing/dice.py* and 01_*Dice_throwing/dicePLOT.py* with the parameters indicated above and on the figures. Running time is less than a minute.