Mini-project in TSRT04: Five-of-kind in Yatzy

21 March 2017

1 Problem Formulation

The dice game of Yatzy is played with five regular six-sided dice. The players throw the dice one at the time and try to get different combinations of die values. The highest number of points is given by five-of-a-kind, which is also called Yatzy. A player may throw the dice three times at each turn: one original throw and two times when the player may keep some of the dice and re-roll the other ones.

In this mini-project, we will not limit ourselves to three throws, but wonder: How large is the probability to obtain five-of-a-kind using a certain number of throws? What is the average number of throws (see the description of the expected value in Section 3) needed to obtain five-of-a-kind? One way of measuring the variations in the number of throws (how common it is to be close to the expected value) is the variance, which is also described in Section 3.

1.1 Presentation

This is one of the mini-projects in the course TSRT04. To pass the course you need to solve one of the mini-projects according to the guidelines and present it to a teacher at one of the lab exercises that offers examination. The examination will be in English so please write your code and comments in English (a good exercise since English coding is common at companies). The problem should be solved in groups of two students, or individually. Since it is an examination assignment it is not allowed to share or show MATLAB code or notes to other students. It is, however, permitted to discuss the project orally with other groups; for example, to share some good advice!

- You should write a MATLAB function `yatzy` that simulates a large number of attempts to obtain five-of-a-kind, and computes the expected value and variance of how many throws that are required. The function should also draw a histogram over the number of throws that are needed and compare the results with the analytic solution given in Section 3. The function `yatzy` should call other functions, that you've written yourself, to solve certain subproblems.
- The solution should be demonstrated and the code should be showed to the teachers at Lab 3 or Lab 5. We will check if you have followed the guidelines, if you have a reasonable amount of comments in the code, if

the plots are easy to understand, and if you can describe what the different parts of the code are doing. There is a style guide available on the course homepage, which elaborates on how a good solution should look like. Please make sure to read it and follow the recommendations!

- When the teachers have told you that you have passed the project you should also submit the code to Urkund for control of plagiarism. You send an email to `hakan.johansson.liu@analys.urkund.se` with your full names in the text field. The code should be attached in a text document called coursecode_year_studentid1_studentid2.txt (e.g., TSRT04_2017_helan11_halvan22.tx This document should contain all functions and script that were examined, including a short example of how to call the function to solve the project.

2 Suggested Solution Outline

Begin by reading through the whole document so that you understand the problem specification and our suggested way to divide the problem into smaller subproblems.

The steps below provide one way to split the main programming problem into subproblems that can be solved one at a time. We have chosen the subproblems to make it possible to verify each part before you continue with the next part. Make sure to take these opportunities! Although the solution outline suggests that you write small functions that solve subproblem, this doesn't mean that it is good or efficient to use (call) all these small functions to solve the original problem. Some functions take care of such small pieces that it is better to copy the code into a new larger function, than to call the small function.

Note that this solution outline is intended for students with no or little previous programming experience. An experienced programmer would probably, based on previous experience, divide the problem into different and/or fewer parts. If you consider yourself an experienced programmer and think that you have a better approach to solve the problem, you are allowed to solve it in your own way. However, if you need help, the teaching assistant has more experience of the outlined solution. If you don't follow the solution outline you must at least make sure that the solution gives the same functionality.

2.1 Several Throws

Write a function that simulates a throw with a given number of dice and returns the result. Let the number of dice be an argument to the function and return the result as a vector with the results of the throw. Hint: Create a vector with random numbers in the interval 0 to 1, one number for each die, multiply by 6, and round upwards.

Make sure that the probability for a die to show 1, 2, 3, 4, 5, or 6 is identical. One way to verify this is to throw many dice, e.g., 1000, and plot a histogram of the outcome (see `help hist`). How is the histogram supposed to look?

2.2 Count the Number of Each Outcome

Write a function that takes the outcome of a throw with five dice as argument and returns a vector with the number of ones, twos, and so on. Ask the teaching assistant for help if you get stuck here for too long!

Example:

- The outcome [1 4 2 2 4] should yield [1 2 0 2 0 0] (one one, two twos, no threes, two fours, and no fives or sixes).

2.3 Find Out Which Outcome is Most Common

Modify the function from the last step to return the most common outcome (ones, twos, threes, ...). The vector previously returned should come handy. If there are two pairs in the result, just return one of them. It doesn't matter which of them that is returned, thus you choose if you like small or large values. Test the function and make sure that it works properly!

Example:

- The outcome [4 5 4 4 1] should yield 4 (four is the most common outcome).
- The outcome [1 4 2 2 4] should yield either 2 or 4 (you decide).

2.4 Find the Dice to Throw Again

With the knowledge of the most common outcome it is time to decide which dice to save and which to throw again. There are two ways to do this:

1. One alternative is to modify the function from the last step so that it returns a vector containing the indices of the dice to throw again; that is, the dice not showing the number computed in Step 2.3. Test the function!
   - The dice [4 5 4 4 1] should yield [2 5] (throw dice number 2 and 5 again).
   - The dice [1 4 2 2 4] should yield [1 2 5] or [1 3 4] (you decide).
2. The second alternative is to put the dice to save at the beginning of the dice vector. Since the value that the dice you save (from Section 2.3) and how many such dice you have (from Section 2.2) is known, it is possible to create a vector with the correct number of dice and values. After modifying the function make sure to test it before you proceed!

   Example:

   - The outcome [4 5 4 4 1] should yield [4 4 4 $\star$ $\star$] (where $\star$ will be thrown again and so the value is unimportant).
   - The outcome [1 4 2 2 4] should yield either [2 2 $\star$ $\star\star$] or [4 4 $\star$ $\star\star$] (you decide).

2.5 Five-of-a-Kind

Now extend the function to throw the selected dice again, and to repeat this procedure until five-of-a-kind is obtained (i.e., there are no dice to throw again). Let the function return how many throws were needed.

Before continuing with the next step make sure that the function works. One way to do this is to, just for now, display and study the result after each throw.

Be careful with the situation when you first get two-of-a-kind in one throw and on the next throw you get three of another kind, then you should save the one with three-of-a-kind instead of the one you saved at first.

2.6 Monte-Carlo Simulation

A Monte-Carlo simulation means that that you perform an experiment many times in order to get an idea of how underlying probability function looks like. Write a (new) function that throws dice until you have obtained five-of-a-kind many times. Your code should be able to make 10000 experiments in one or a few minutes, otherwise you need to optimize your code. It may be good to write out something with `disp` so that one can follow the progress. Store the number of throws needed in each experiment in a vector. Plot a histogram to illustrate the result (i.e., the number of throws needed). The histogram bins should have width 1. The function should take the number of experiments as an input argument.

2.7 Compute Estimates of the Expected Value and Variance

Modify the function to also return estimations of the expected value and variance for the number of throws. The average value is an estimate of the expected value, and a formula to estimate the variance is given in Section 3. In this case x_i is the number of throws needed in experiment i, and n is the number of experiments. Ask the teaching assistant if you cannot work out how to use the formulas!

Section 3 also presents the theoretical expected value and variance. Make sure that the estimates from your function are close to the exact values!

2.8 Compare with the Analytic Solution

Extend your function to also plot the analytic probability function in the same figure as the histogram from the Monte-Carlo simulations (Hint: Plot the histogram first and then the probability function by using `hold on`). The analytic probability function can be found in Section 3 and you should compute it for k from 1 to some appropriate number. To compare the results, keep the following in mind: The sum of the heights of the bins in the histogram equals the number of experiments in the Monte-Carlo simulation. On the other hand the sum of the probabilities in the probability function adds up to 1 (five-of-a-kind

will turn up sooner or later). Hence, to get comparable results either the values in the histogram or the probability function must be scaled.

Note: The histogram and the analytic probability function should have approximately the same shape, particularly when you consider 10000 experiments in the Monte-Carlos simulatoin. If this is not case, something might be wrong in your solution. Discuss this with your teaching assistant!

If you have called your function `yatzy` the following command in MATLAB should produce a plot with the estimated probability function (the histogram) and the analytic solution, as well as the estimated expected value `mhat` and the variance `s2hat`..

```
>> [xhat, s2hat]= yatzy(experiments)
```

where `experiments` is the number of experiments in the Monte-Carlos simulation (the number of five-of-a-kind that are simulated). Compare the behavior for a small and large number of experiments, respectively.

2.9 Preparations for the Presentation

To prepare for the project presentation, you need to read through the section "Presentation" in the beginning of this document and also read the Examination and Coding Guide that you find on the course homepage. At these places you will find that your need to have at least two functions in your code, functions and variables need to have descriptive names, there should be a reasonable amount of comments in the code, all figures should be self-explaining, comments should be in English, etc. If there is any requirements in the Examination and Coding Guide that you do *not* fulfill, then you need to take care of this *before* you present the project, otherwise you might have to wait a long to get a second chance to present.

Finally: Don't forget to submit the code to Urkund when you have passed to project assignment. You will find the instructions in the section "Presentation" in the beginning of this document.

3 Useful Facts

Probability Theory: Definitions

Let X denote the outcome (result) of a die throw (X is a so called random or stochastic variable). P denotes the probability for a certain outcome; for example, $P(X = 3)$ is the probability to get a three.

The probability function $p_X(k)$ is defined to be $p_X(k) = P(X = k)$.

For a fair die the probabilities of the outcomes are $p_X(k) = \frac{1}{6}$, for $k = 1, \ldots, 6$.

To throw several dice, or the same die several times, are independent events. It follows that $P(X_1 = k_1, X_2 = k_2) = P(X_1 = k_1) \cdot P(X_2 = k_2)$, which means that the individual probabilities are multiplied.

As another example of a stochastic variable, let Y be the number of throws needed to get five-of-a-kind.

The expected value (mean) of a stochastic variable Y can be interpreted as the average outcome of an experiment:"How many throws are needed on average to get five-of-a-kind?"

The variance is a measure that tells how large the (squared) deviations from the expected value are. If the variance is small that implies that the number of throws needed to get five-of-a-kind stays approximately the same from time to time, whereas a large variance indicates that the number of throws may vary substantially.

If $x_1, \ldots, x_n$ are n stochastic outcomes, the expected value can be estimated as

$$\hat{m} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

and the variance as

$$\hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \hat{m})^2.$$

3.1 The Probability Density Function in the Assignment

As described above, the probability function in the problem formulation can be described using a matrix: Let $p(k)$ be the probability that k throws are needed to get five-of-a-kind. It then holds that

$$p(k) = e_1^T A^k e_5 \tag{1}$$

where $k = 1, 2, 3, \ldots$, where ^{T} denotes the matrix transpose, and

$$A = \begin{bmatrix} 0 & \frac{1}{6} & \frac{1}{36} & \frac{1}{216} & \frac{1}{1296} \\ 0 & \frac{5}{6} & \frac{10}{36} & \frac{15}{216} & \frac{25}{1296} \\ 0 & 0 & \frac{25}{36} & \frac{80}{216} & \frac{250}{1296} \\ 0 & 0 & 0 & \frac{120}{216} & \frac{900}{1296} \\ 0 & 0 & 0 & 0 & \frac{120}{1296} \end{bmatrix}, \qquad e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \qquad e_5 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

The expected value of the number of throws needed is $\frac{191283}{17248} \approx 11.0902$, and the variance is $\frac{12125651655}{297493504} \approx 40.7594$.

For the interested: The above formulas are derived using the following approach: The probability to get five-of-a-kind in k throws is equal to the probability to have four-of-a-kind in throw $k-1$ and in throw k get another of the

same kind, together with the probability of three-of-a-kind in throw $k-1$ and get another two in throw k, *etc*. The vector $x(k)$ can now be constructed,

$$x(k) = \begin{pmatrix} P(\text{first time with five-of-a-kind in throw } k) \\ P(\text{four-of-a-kind in throw } k) \\ P(\text{three-of-a-kind in throw } k) \\ P(\text{two-of-a-kind in throw } k) \\ P(\text{all different in throw } k) \end{pmatrix}$$

and the probabilities can be calculate recursively as $x(k) = Ax(k-1) = A^2x(k-2) = \ldots$, where A contains the probabilities for the result in one throw.

4 Voluntary Extra Assignments

This section describes a number of voluntary extra assignments. If you are experienced programmers and feel that the project was too easy, we recommend that you also do some of these extra assignments, to learn MATLAB properly.

MaxiYatzy: In the game MaxiYatzy one uses six dice instead of five. If all six dice yield the same value, then one has got MaxiYatzy. Modify your code to take the number of dice as an argument. You can make this a non-mandatory argument so that one can choose to not provide the number of dice as an argument and thereby get five dice. You can use the variable `nargin` for this purpose.

Other types of dice: Conventional six-sided dice are called D6 (where D stands for dice). There are other types of dice with more or fewer number of sides, but where all the sides still have the same shape and therefore the same probability. Some examples are D4, D8, D12 and D20. To see how these dice look like you can make a Google search on "Platonic solids". Edit your code so that the number of sides on the dice is an argument. Check how this changes the results and the number of experiments needed in the Monte-Carlo simulations to get a smooth graph.

False dice: At casinos it is very important that all dice have exactly the right shape so that the probability is exactly $\frac{1}{6}$ for each of the six outcomes. Suppose that the Yatzy dice are note exact, but has a higher probability of getting a six. Modify your code to handle this case. The probability of getting a six can be an argument. Check what happens when the probability of getting a six approaches 1.