EA1 Homework Program 2: Monte Carlo Simulation

Due Friday, Oct. 6, 2023, at 6:00am

In this homework you will write a MATLAB program to compute the probability of a random event. Since probability measures the chance of a particular outcome of an experiment, sometimes it is easy to perform some experiments directly to calculate the probability of an event. For example, the probability of a fair (or evenly-balanced) coin landing on Heads is 1/2, and the probability of it landing on Tails is 1/2. Likewise, when rolling an evenly-weighted die, the probability of getting any particular value is 1/6, since there are six sides.

If you were not sure of the probability of either of these events, you could perform some experiments. For example, you could toss a coin 100 times, and record how many times it lands on a particular side. Dividing that value by the number of tosses would give a value somewhere near 1/2. The more tosses you perform, the more accurate your estimate should be. Likewise with a die: you can roll a die 100 times and record how many times it lands on a particular value. Dividing by the number of rolls would give a probability near 1/6 for each value. The more rolls you perform, the closer you expect to get to 1/6. This type of simulation to estimate a probability is called "Monte Carlo Simulation".

Some problems are not so straightforward. For example, suppose you want to find, after rolling 2 dice (whose number of sides may not be 6), the probability of the sum being a particular value. There is a mathematical way to compute this, but instead we will do an experiment again. We roll 2 dice many times, and again tally up how many times we get a particular value. We can do this for each possible value the sum can take. After the appropriate division, we get a set of estimated probabilities for the different sum values. This kind of reasoning can be extended to finding the probability that the sum takes a particular value after rolling an arbitrary number of dice, assuming we perform enough trials.

In this assignment, we will estimate the probability of getting each possible sum value after rolling d dice, each with s sides. Our estimate will involve performing n trials. Follow the steps below to complete this task. *See Appendix A for an example of each of the important steps.*

Using the MATLAB editor, create a script M-file which performs the following steps:

1. Ask the user to input a value of d, and store the result in a variable called `d`.

2. Ask the user to input a value of s, and store the result in a variable called `s`.

3. Ask the user to input a value of n, and store the result in a variable called `n`.

4. We want to create a matrix `rolls` whose entries denote outcomes of rolling the dice. Specifically, each column represents a particular die (d columns), and each row represents one roll of all the dice (n rows). All of the entries must be between 1 and s, since these are the possible values of a die roll. Use the function `randi()` to create the matrix `rolls`. Note: `randi(x,y,z)` creates a `y`×`z` matrix with random integers between 1 and `x`.

5. Find the sum of all the experiments by summing across the rows of `rolls` using the `sum()` command with an optional argument, and store your result in a variable called `dice_sums`. Read the documentation for `sum`!

6. You now have a vector, `dice_sums`, that contains the outcomes of all your experiments. Now it's time to tally them up. Use the command
   ```
   [tally,sums] = hist(dice_sums,d:s*d)
   ```
   to return the number of times each value in the second argument (`d:s*d`) appears in the first argument (`dice_sums`), along with the possible values of the sums (stored in `tally` and `sums`,

respectively). For an example, please see Appendix C. (Appendix C contains an example with non-integer values, which does not apply to this assignment, but is there for full context.) Note that MATLAB will advise that you use `histogram` instead of `hist`. For this assignment, you can ignore this suggestion.

7. To get a probability from the tally count, we must divide by the total number of trials. Create a variable `probs`, which stores `tally` divided by the number of trials, n.

8. Finally, we have to plot our results. Use the `bar()` function to create a bar plot of your results. Read the documentation! Make sure to label your axes. To create a nice title, use the command:
   ```
   title(sprintf('%i Dice, %i Sides, %i Trials', d, s, n))
   ```
   An example of the output is shown in Figure 1.

9. As a last step, create a two-column matrix called `results`, where the first column contains the values in `sums`, and the second column contains the values in `probs`.

10. Run your script with input values d=3, s=10, and n=1000. Copy and paste your `results` matrix into a comment at the end of your M-file. An example of the `results` matrix is shown in Figure 2.

11. Repeat for d=4, s=6 and n=1000000. Once again, copy and paste your `results` matrix into a comment at the end of your M-file.

12. Try the experiment with increasing numbers of trials, e.g. 10, 100, 1000, 10000, 100000. In a comment at the end of your code, please describe qualitatively what you notice about the shape of the histogram as the number of trials increases. Please also explain why this happens.

Example matrices are shown in Appendix A, a script template is shown in Appendix B, and an explanation of the "hist" function is shown in Appendix C.

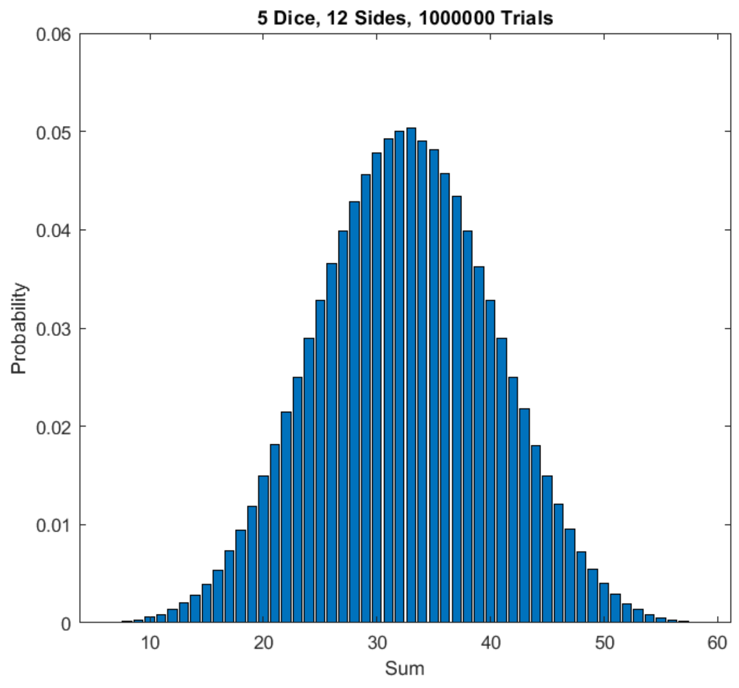

Figure 1

```
    3.0000    0.0020
    4.0000    0.0010
    5.0000    0.0060
    6.0000    0.0080
    7.0000    0.0170
    8.0000    0.0310
    9.0000    0.0180
   10.0000    0.0350
   11.0000    0.0330
   12.0000    0.0580
   13.0000    0.0600
   14.0000    0.0570
   15.0000    0.0640
   16.0000    0.0880
   17.0000    0.0790
   18.0000    0.0760
   19.0000    0.0700
   20.0000    0.0600
   21.0000    0.0510
   22.0000    0.0500
   23.0000    0.0460
   24.0000    0.0320
   25.0000    0.0280
   26.0000    0.0150
   27.0000    0.0090
   28.0000    0.0050
   29.0000    0.0010
   30.0000         0
```

Figure 2

Appendix A: Examples

Below are examples of the important steps, using $d = 3$, $s = 6$, and $n = 10$.

$$\texttt{rolls} = \begin{bmatrix} 2 & 5 & 1 \\ 2 & 3 & 4 \\ 6 & 4 & 2 \\ 5 & 6 & 5 \\ 4 & 1 & 6 \\ 2 & 2 & 4 \\ 3 & 1 & 2 \\ 2 & 5 & 2 \\ 6 & 2 & 2 \\ 2 & 2 & 6 \end{bmatrix} \qquad \texttt{dice\_sums} = \begin{bmatrix} 8 \\ 9 \\ 12 \\ 16 \\ 11 \\ 8 \\ 6 \\ 9 \\ 10 \\ 10 \end{bmatrix}$$

$$\texttt{tally} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 2 \\ 2 \\ 2 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad \texttt{sums} = \begin{bmatrix} 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ 16 \\ 17 \\ 18 \end{bmatrix} \qquad \texttt{probs} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.1 \\ 0 \\ 0.2 \\ 0.2 \\ 0.2 \\ 0.1 \\ 0.1 \\ 0 \\ 0 \\ 0 \\ 0.1 \\ 0 \\ 0 \end{bmatrix} \qquad \texttt{results} = \begin{bmatrix} 3 & 0 \\ 4 & 0 \\ 5 & 0 \\ 6 & 0.1 \\ 7 & 0 \\ 8 & 0.2 \\ 9 & 0.2 \\ 10 & 0.2 \\ 11 & 0.1 \\ 12 & 0.1 \\ 13 & 0 \\ 14 & 0 \\ 15 & 0 \\ 16 & 0.1 \\ 17 & 0 \\ 18 & 0 \end{bmatrix}$$

Appendix B: Template of script file

hw2_ilya.m

```
% Homework Program 2
%
% Name:   Mikhelson, Ilya
% Date:   10/6/2023

% Get inputs:
  (insert code here)

% Create matrix rolls:
  (insert code here)

% Calculate dice_sums:
  (insert code here)

% Calculate tally and sums:
  (insert code here)

% Find desired probabilities:
  (insert code here)

% Plot bar graph of results:
  (insert code here)

% Create a results matrix:
  (insert code here)

%
% (insert answer to step 12, as well as the copy/pasted output
% from steps 10 and 11, commented out)
%
```

Appendix C: Example of "hist" function

Suppose you have a vector `x`, defined below.

$$\mathtt{x} = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 6 \\ 4 \\ 1 \\ 2 \\ 5 \\ 3 \\ 5 \end{bmatrix}$$

If you wanted to count the number of occurrences of all of the numbers, you would use the `hist` function in MATLAB. An example is shown below.

```
tally = hist(x, 1:6)
```

Here, we are tallying up how many times each of the values in the second input argument (i.e. `1:6`) appears in `x`. The result would be:

```
[3 2 1 1 2 1]
```

which says that the value 1 appears 3 times, the value 2 appears 2 times, the value 3 appears 1 time, and so on. If the numbers in `x` are not integers, this would still work. However in this case, the values in the second input argument would represent the centers of the bins. In other words, it would count up all the values in `x` that are from halfway to the preceding value up to halfway to the following value. In the example above, this would mean that all values from 2.5 to 3.5 would be in the "3" bin, values from 3.5 to 4.5 would be in the "4" bin, and so on. As an example, consider the vector below:

$$\mathtt{y} = \begin{bmatrix} 1.2 \\ 1.4 \\ 2.1 \\ 6.7 \\ 4.9 \\ 1.1 \\ 2.6 \\ 5.7 \\ 3.8 \\ 5.2 \end{bmatrix}$$

Now, the code

```
tally = hist(y, 1:6)
```

results in

```
[3 1 1 1 2 2]
```

which says that there are 3 values between $-\infty$ and 1.5, 1 value between 1.5 and 2.5, and so on.

Oftentimes, it is also useful to get the bin centers as an output of the function directly, and this can be done using a second output argument, so the function looks like this:

```
[tally, bin_centers] = hist(y, 1:6)
```

Even though this may look strange since we manually specified the bin centers, it is a good habit because there are other ways to call the `hist` function where the centers are not so clear, and it is always useful to know what centers were used.