

# Lab/Project II — Exploring Random Variables\*

Peter Brinkmann

This document gives you the opportunity to explore some important random variables in a hands-on fashion. I am assuming that you have worked through Lab I, available at

<http://www.math.uiuc.edu/iprom/materials.html>.

In particular, you should be able to create, modify, and run simulations.

## 1 Setting up the experiment

Given a random variable  $X$ , we want to understand its probability mass function  $p(x) = P\{X = x\}$ , for all values  $x$  in the image of  $X$ . How can we accomplish this in a computer experiment? One way to do this is to have the computer simulate the random variable  $X$  and display the results in a *histogram*, i.e., we repeatedly perform an experiment whose outcome determines the value of  $X$ , keeping track of how often each possible value of  $X$  occurs. Then, for each value of  $X$ , we draw a vertical bar whose height indicates how often this value has occurred. Because of the relative frequency interpretation of probability, the resulting graph gives us a good idea of the general shape of the probability mass function  $p(x)$ .<sup>1</sup>

---

\*Copyright © 2003, Peter Brinkmann ([brinkman@math.uiuc.edu](mailto:brinkman@math.uiuc.edu)). Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is available at <http://www.fsf.org/copyleft/fdl.html>.

<sup>1</sup>Of course, to turn this histogram into a probability mass function we would need to divide through by the total number of experiments performed. But this normalization only affects the scale of the plot, not its shape.

**Action 1.** Open the file `blank.mdl` and save it under a new name, say `my_random_variable.mdl`. Increase the size of the window, because you'll be filling it with a fair number of blocks from the IPROM library.

In the 'Simulation' menu, change the stop time to a large value, say 1000. Open the IPROM library by typing `iprom` at the Matlab prompt, copy the blocks labeled 'Dice' and 'Histogram' into your new model, and connect them with a line. You have now created a model that simulates 1000 rolls of a die and displays the results as a histogram.

Run the simulation a few times. After each run, you should see six vertical bars, one for each face of the die, and their heights should be about the same. Double-click on the Histogram block and change the update frequency to a small positive integer, say 10. If you run the simulation again, you'll see intermediate results.

In addition to the probability mass function, we know some other interesting properties of a random variable, the expected value and the variance. IPROM can compute estimates of both.

**Action 2.** Copy an Average block and a Display block into your model. Connect the output of the Average block to the input of the Display block. In order to feed the output of the Dice block into your Average block without losing the histogram, you need to branch the output of the Dice block off to the input of the Average block.<sup>2</sup>

If you run the simulation again, you should see the average value (about 3.5) as well as the histogram. Try this a few times.

**Action 3.** Similarly, copy a Standard Deviation/Variance block and another Display block into your model and connect them as before. Double-click on the Variance block and configure it to compute the Variance rather than the standard deviation.

If you run the simulation again, you should now see a histogram, the average value, and the variance (about  $35/12 \approx 2.92$ ).

**Action 4.** Now modify the model to simulate the sum of three dice: double-click on the dice module and choose three dice instead of one. Then drag a Matlab Expression block into your model and drop it on the line emanating

---

<sup>2</sup>Remember how to do this? Move the mouse to a point on the line between the Dice block and the Histogram block. Now hold down the CTRL key while dragging the mouse to the input of the Average block.

from the Dice module. If you do it right, it should automatically get patched into this line. Make sure you put the Expression block between the Dice block and the first branch point, or else your averages or variances will be wrong. You may have to move your Dice block to make room for the Expression block.

Now double-click on the Matlab Expression block and enter the expression `sum(u)`, which will compute the sum of the dice (see Appendix A). You also need to set the number of outputs to 1.

If you run the simulation now, you should see a histogram that shows a distinct hump in the middle. If you plot the result from [Ros02][Page 171, Problem 3], the average should be about 10.5, and the variance about  $35/4 = 8.75$ .

**Action 5.** When everything is working as it is supposed to, save your model from Action 4, then print it out and hand it in as part of this project. Label the page clearly with ‘**Three dice**’.

## 2 Investigating random variables

You have now created the basic setup for investigating random variables. In this section, you will use this setup to become acquainted with discrete random variables of fundamental importance.

### 2.1 Binomial Random Variables

Take a coin with  $P(\text{heads}) = p$ . Flip the coin  $N$  times and let  $X$  be the random variable that indicates how many times the coin comes up heads. This random variable is called a *binomial* random variable with parameters  $(N, p)$  (see [Ros02][Section 4.6]).

Modify your model so that it simulates the random variable  $X$ . You can do this by replacing the Dice block by the Coins block (say with  $N = 10$  coins).<sup>3</sup> This works because the Coins block returns a vector of 0s and 1s, with 0 for tails and 1 for heads, so that the sum of the entries of this output vector gives the number of coins that came up heads.

---

<sup>3</sup>You’ve probably noticed that the IPROM library includes a block labeled ‘Binomial Random Variable’ that simulates the kind of random variable we’re interested in. However, for educational purposes, I want you to construct a binomial random variable from scratch. Later we’ll use the prefabricated Binomial block.

Save the new simulation under a new name, say `binomial.mdl`, and run it a few times, with various numbers of coins and various values of  $p$  to get a feeling for the behavior of this random variable. When configuring the Coins module, instead of entering a probability distribution like [11] (which means heads and tails are equally likely) you can input just the probability of heads, e.g., 0.5 or 0.2.

**Action 6.** Fill in the table on the last page. You may want to increase the stop time in order to get better approximations.

Beside the table, write the known theoretical formulas for  $E(X)$  and  $Var(X)$  (in terms of  $N$  and  $p$ ). You should check that your experimentally determined values are reasonably close to the theoretical values. (If not, then your model might be wrong.)

## 2.2 Geometric Random Variables

Flip a coin until it comes up heads. Let  $X$  be the number of flips it takes for the coin to come up heads. Then  $X$  is called a *geometric* random variable ([Ros02, Section 4.8.1]).

Modify your simulation so that it simulates a geometric random variable. This is a bit tricky to do correctly, but you can cheat ever so slightly and still come up with a decent simulation.

That is, adjust the Coins block so that it flips a large number of coins, say 500, and just hope that at least one of those coins will come up heads. That's a fairly safe assumption unless  $p$  is close to 0. The output of the Coins block will be a vector of length 500 at every step. Now, modify the Expression block so that it finds the index of the first nonzero entry of the vector (i.e., the first coin showing heads). You'll find the necessary Matlab functions in Appendix A.

**Action 7.** Save your simulation under a new name, say `geometric.mdl`, and run it a few times. Fill in the table on the last page.

Beside the table, write the known theoretical formulas for  $E(X)$  and  $Var(X)$  (in terms of  $p$ ). You should check that your experimentally determined values are reasonably close to the theoretical values. (If not, then your model might be wrong.)

**Action 8.** (Challenge problem in case you're feeling frisky.) Can you modify the model so that it simulates the number of steps it takes for the coin to

come up heads  $k$  times (not necessarily in a row), for  $k \geq 1$ ? This is known as a *negative binomial* random variable ([Ros02, Section 4.8.2]). You can find all the necessary Matlab functions in Appendix A.

### 2.3 Hypergeometric Random Variables

Take an urn containing  $N$  balls, of which  $m$  are white and  $N - m$  are black. Draw  $n$  balls from this urn (without replacement), and let  $X$  denote the number of white balls selected. Then  $X$  is a *hypergeometric* random variable ([Ros02, Section 4.8.3]).

Save your model under a new name, say `hypergeometric.mdl`. Replace the Coins block by the Urn block from the IPROM library. Double-click on the Urn block. In the configuration dialog, replace the distribution vector by the vector  $[m, (N - m)]$ , for some  $m$  and  $N$ , and adjust the number  $n$  of balls to be drawn.

Now the output of the Urn block will be a vector consisting of two numbers, the number of white balls and the number of blacks balls that were drawn. We can now adjust the Expression block to extract the number of white balls, which is what we're really interested in. To do this, replace the Expression by `u(1)`, which denotes the first entry of the vector.

**Action 9.** Fill in the table on the last page.

Beside the table, write the known theoretical formulas for  $E(X)$  and  $Var(X)$ . You should check that your experimentally determined values are reasonably close to the theoretical values.

## A Some useful Matlab functions

This appendix lists some Matlab functions that may help you when you're working on this lab, as well as functions that may be helpful later.

<code>sum(u)</code>	sum of the entries of the vector $u$ , e.g., <code>sum([1 2 3 4])=10</code>
<code>prod(u)</code>	product of the entries of $u$ e.g., <code>prod([1 2 3 4])=24</code>
<code>diff(u)</code>	vector of differences between subsequent entries of $u$ e.g., <code>diff([1 2 4 7 5])=[1 2 3 -2]</code>
<code>cumsum(u)</code>	vector of partial sums of the vector $u$

`find(u)` e.g., `cumsum([1 2 3 4])=[1 3 6 10]`  
list of indices of nonzero entries of  $u$   
e.g., `find([0 1 0 0 1 0])=[2 5]`  
`min(u)` the minimum of the entries of  $u$   
`max(u)` the maximum of the entries of  $u$   
`length(u)` the number of entries of  $u$ ,  
e.g., `length([0 1 2 3 4])=5`  
`sort(u)` sorts the entries of  $u$   
e.g., `sort([1 4 -2 3 0])=[-2 0 1 3 4]`

## References

- [Ros02] Sheldon Ross. *A first course in probability*. Prentice Hall, Upper Saddle River, NJ 07458, sixth edition, 2002.

Name:

Binomial Random Variable			
Parameters		Estimates	
$N$	$p$	$E(X)$	$Var(X)$
8	0.5		
12	0.5		
10	0.2		
25	0.2		
9	$\frac{1}{3}$		

Geometric Random Variable		
Parameters	Estimates	
$p$	$E(X)$	$Var(X)$
0.5		
0.8		
0.2		
0.1		

Hypergeometric Random Variable				
Parameters			Estimates	
$N$	$m$	$n$	$E(X)$	$Var(X)$
20	8	10		
18	6	9		
15	10	3		

Matlab Expression for Problem 8 (optional):

Room for notes