

Lab I — IPROM Familiarization*

Peter Brinkmann

The purpose of this lab is to introduce you to IPROM, the Illinois Probability Modules. I am assuming that you have already settled into your computer account, and that you know how to create directories, copy files, etc. If you have not yet installed IPROM in your account, follow the instructions at

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

before you proceed.

1 Working with simulations

Change into your IPROM directory (e.g., if your IPROM directory is called `my_iprom`, you want to type `cd my_iprom` at the prompt in your terminal window). Start Matlab by typing `matlab` at the prompt. Within Matlab, open the file `sample.mdl`. You should see a diagram consisting of five blocks, labeled 'Dice', 'Event', 'Probability', 'P(E)', and 'Bar Graph'. The meaning and purpose of this system should be intuitively clear: We simulate an experiment (roll one die), define an event (the result is greater than 4), compute the probability of the event, and display the result. Also, the raw results of rolling the die will be displayed as a bar graph.

*Copyright © 2003, Peter Brinkmann (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>.

1.1 Running the simulation

In order to run this simulation, click on the item labeled **Start** under the **Simulation** menu (on some computers, you can also start the simulation by hitting **CTRL-T**). You'll see some activity for a while, and then the simulation will stop, displaying a value that should be reasonably close to the probability of our event (0.333...). The bar graph will show integer values between 1 and 6. Run the experiment a few times and notice how the displayed probability value changes.

Here's what's happening: When you click on **Start**, Matlab will simulate 50 rolls of the die, keep track of the number of times our event occurred, and compute an estimate of the probability of the event using the relative frequency interpretation of probability [Ros02, Section 2.3]. Increasing the number of rolls will usually improve the quality of the estimate. In order to do this, click on the **Configuration Parameters** item in the **Simulation** menu. You'll see a dialog window that lists, in particular, a **Stop Time** of 50. Change this to, say, 200 and click on **OK**.

If you start the simulation now, you'll see that it runs for a longer time, and the resulting values are (most likely) closer to 0.333... than before. Try this a few times.¹

1.2 Modifying the simulation

Double-click on the icon of the dice block. You'll see a dialog window that lets you modify the properties of the dice block. The first entry is a list of relative weights of the sides of the die. By default, they are all equal, meaning that the block simulates a fair die. You can create a biased die by changing the values.² E.g., if you change the last entry to 3, you create a die with $P(\{6\}) = \frac{3}{8}$, and $P(\{1\}) = \dots = P(\{5\}) = \frac{1}{8}$. Try this, run the simulation again, and notice how the displayed result changes.

Now, double-click on the **Event** icon. You'll see a dialog window that allows you to change the event. E.g., if you replace the expression `u>4` by

¹Updating the bar graph after every single step slows the simulation down quite a bit. If you want to speed things up, you can double-click on the **Bar Graph** icon and increase the number of iterations between updates, or disable updates altogether, leaving only one drawing of the graph at the end of the simulation.

²You can also change the number of sides of the die by changing the length of the list. For instance, `[1111]` defines a fair tetrahedral die.

`u==2` and click on **OK**, then you've defined the event that 2 comes up. Run the simulation a few times and see whether the resulting probability estimates agree with the exact value. (Keep in mind that the die will still be biased because of the previous step.) Look at the appendix (especially the logical expressions in Appendix A.2) if you want to learn more about Matlab expressions.

Finally, you can modify the labels of the blocks by clicking on them. For example, you might click on the label 'Dice' and replace it by 'Biased dice'. Click somewhere else within the diagram when you're done modifying the label.

So far, you have learned how to run as well as modify an existing simulation. The next step is to learn how to build a new simulation from scratch.

2 Building a new simulation

Now, we will recreate the same simulation from scratch.

Open the IPROM library by typing `iprom` at the Matlab prompt. You'll see a window filled with blocks, some of which you already know from `sample.mdl`. This is the library of blocks that we will assemble into various simulations throughout the semester. Open the file `blank.mdl`. This file is the empty canvas on which we will create all our simulations. Before you do anything else, you should assign a new name to the simulation using the **Save as...** item. The file extension must be `.mdl`.

Warning 1. When creating a new IPROM model, *always* start with the file `blank.mdl`. *Do not* use the **New Model** menu item from the **File** menu. Although `blank.mdl` looks just like a new model, it comes with a number of default settings that IPROM requires. If you start with the **New Model** menu item, chances are you'll encounter some very strange error messages regarding mismatched types and such.

In order to recreate the simulation `sample.mdl`, you first need to copy five blocks ('Dice', 'Probability', etc.) from the library to your empty window. You do this by dragging the blocks from their place in the library to your window.³ When looking for the right blocks in the library, keep in mind that

³On my machine, there's an annoying little bug (I'm not sure whether to blame it on Matlab or the window manager or something else) that fills up the display with some junk

labels of blocks can be changed (e.g., the block labeled 'Event' in `sample.mdl` is labeled 'MATLAB Expression' in the library).

After you've copied all the necessary blocks to your window, you need to connect them. You connect two blocks by dragging the output of one of them (i.e., the little wedge pointing away from the block) to the input (i.e., the little wedge pointing toward the block) of the other. Move the mouse carefully to make sure you don't miss any of your targets. A connection between two blocks looks like an arrow whose tip is a solid triangle. If the line you've drawn does not end in a solid triangle, you have missed the input port of your target and need to draw the line again.⁴

You still need to draw the line that sends the output of the dice block to the bar graph. In order to draw this line, hold down the **CTRL** key while dragging the mouse from the origin of this line to the input port of the bar graph block. (Alternatively, you can right-drag the mouse.) This completes your recreation of general structure of `sample.mdl`.

Now, in order to fully duplicate `sample.mdl`, you still need to enter the Matlab expression that defines the event that you wish to study, such as `u>4`. That's it — you've built your first IPROM simulation! Run it a few times in order to make sure that it behaves the way it's supposed to.

3 What's next?

If you have closely followed the instructions up to this point, then you have picked up the basic skills you need to work with IPROM. I suggest that you simply start experimenting with the blocks in the IPROM library. Many of them are self-explanatory, and you can get a help message for any block if you just double-click on it and then click on **Help** in the resulting dialog. Also, it may be instructive to look at some of the simulations (files with extension `.mdl`) that are included in the IPROM distribution. Have fun!

lines when I drag blocks around. If this happens to you, you can clean up your window by hitting **CTRL-D**, or by minimizing and then restoring the window.

⁴If you need to get rid of a block or line, just click on it to activate it, then choose the menu item **Clear** under **Edit**, or hit **Delete**.

A Mathematical expressions in Matlab

For simple expressions, we use the usual keyboard characters, e.g., $(x^3-1)/6$ means $\frac{x^3-1}{6}$, `pi` means π , and `2*u-1` means $2u - 1$.

The last function is a useful one to keep in mind since we sometimes want to translate the result of a coin toss (IPROM's Coins block returns 0 or 1 for heads or tails) into the values +1 or -1. The function $2u - 1$ accomplishes just that.

A.1 Built-in functions

The following is a of some of Matlab's built-in functions. It is by no means complete.

<code>exp(x)</code>	exponential, e^x
<code>log(x)</code>	natural logarithm, $\ln x$
<code>abs(x)</code>	absolute value, $ x $
<code>sqrt(x)</code>	square root, \sqrt{x}
<code>sign(x)</code>	signum function, which equals $\begin{cases} +1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$
<code>sin(x)</code>	
<code>cos(x)</code>	trigonometric
<code>tan(x)</code>	functions
<code>rem(a,b)</code>	remainder of a under division by b

Example 1.

<code>sin(exp(y))^4</code>	means $\sin^4(e^y)$,
<code>rem(n,2)</code>	is the remainder of n under division by 2.

A.2 Logical expressions in Matlab

Expressions like `u>=2` are treated as logical functions, and return a value of either 1 (true) or 0 (false). Matlab knows the following logical operators:

<code>==</code>	equals,
<code>~=</code>	does not equal,
<code><</code>	less than,
<code>></code>	greater than,
<code><=</code>	less than or equal to,
<code>>=</code>	greater than or equal to,
<code>&</code>	and,
<code> </code>	or,
<code>~</code>	not.

Example 2.

<code>(u==3) (u>=5)</code>	means $u = 3$ or $u \geq 5$,
<code>(u ~=6) & ~(u<10)</code>	means $u \neq 6$ and not $u < 10$.

In the Matlab Expression block, you can use any Matlab function. In particular, you can use logical expressions as above. This is extremely useful since we frequently define events in terms of logical expressions as above.

References

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