Matlab for the Absolute Beginner

By Arvind Ravichandran

7/12/11

"Do not worry about your difficulties in Mathematics. I can assure you mine are still greater."

- Albert Einstein

When first introduced to Matlab, with no previous programming experience, I myself was a bit nervy about using the software to solve and present solutions to complex projects. I have, however, attempted to allay these initial jitters to some extent with a simple guide for the beginner. I have collated these notes together from sources such as the help pages from the MathWorks website, notes posted by Matlab geeks on websites, and advice from my own experience. Also, I have appended some exercise problems at the end, and also have the solutions to them. So please try them!

Some of you may be entirely familiar with all that I talk about in the following pages and may even prefer to skip entirely what is pretty much a rudimentary instruction manual.

All I wish to convey is that Matlab is not an intimidating computer gimmick. But an incredibly useful tool that you should learn to use to solve engineering problems, quickly and efficiently. And I hope that I can be of service to help you master this software.

Let us begin!

1. The Command Window

There are two windows that you can type stuff into in Matlab: The editor and the command window. Knowing how to use these windows can make coding and debugging your programs much less painstaking.

As the name suggests, the command window is where you enter instructions for Matlab to perform. It is handy to know a few basic commands so that you can instruct Matlab to perform basic functions. Listed are some functions, which I find very handy to make moving around the Command Window much easier. Make sure that you try these functions out and are accustomed to them.[1]

- The `help` function is extremely useful when you are unfamiliar with a Matlab function. Typing in "`help`" followed by your query (and pressing enter of course) will give an output describing the function and the syntax required to use it. Try typing in "help plot". I find this particular help page very helpful for plotting, and you will see yourself using most of the tips listed on this topic.
- `clc` - clears all input and output from the Command Window display, giving you a clean screen.
- `clear` - clears all functions and variables from memory.
- `lookfor` – this can be quite an useful command. If all you remembered was a fragment of command or topic, you can use `lookfor`. For example, `lookfor plot` will list all occurrences of the string 'plot' within Matlab documentation.
- `shg` - brings the current figure window forward if you have plotted a graph
- Up button – The up button in the command window returns your previously entered statement.
- `home` – moves the cursor to the upper left corner of the window.
- `quit` - quits Matlab, although you can always do it the amateur way.

[1] Note that the words that are to be typed into Matlab are in a different font than the explanatory texts.

Note that you can also type in mathematical problems into Matlab to solve it:

e.g.
The polynomial $s^3 + 6x^2 - 72x - 27$ represented in MATLAB software reads

```
>> p = [1 -6 -72 -27];
```

The roots of this polynomial are returned in a column vector when you enter:

```
>> r = roots(p)
```

as

```
r =
   12.1229
   -5.7345
   -0.3884
```

Another very useful function of the command window is copying code from the editor window and watching variable values change. I shall explore this later in section 12.

2. Variables

A Matlab variable is essentially a tag that you assign to a value while that value remains in memory. The tag gives you a way to reference the value in memory so that your programs can read it, operate on it with other data, and save it back to memory. Variables will be the essential dynamic medium, which your program will access, change and produce data. In the following example, the `>>` sign shows what I have typed in the command window.

For instance:

```
>> monkey = 8

monkey =

     8
```

Now that you have “taught” Matlab that `monkey` = 8, every time you type in `monkey`, Matlab will understand that you are talking about the number 8 (until you enter `clear`, which clears the memory of all variables).

```
>> 2*monkey

ans =
        16
```

I find the use of an analogy very helpful to drive this concept home. Take the above example. We first enter `monkey = 8`. Now imagine that the variable on the *left* of the equal sign is a little box and that you are storing the stuff on the *right* into the variable on the *left*. You can never swap this order, unlike what you might be used to doing in math.

Anytime you call the variable thereafter, you call for the value stored inside. However, you can change the contents of the "box" by reassigning the variable (putting or taking out stuff from the box). For example, `monkey = monkey + 1`. Here Matlab will understand that you are assigning a new variable monkey (a value of 9) by using the old value of `monkey`, which is 8. Reassigning variables like this will come very much in handy when we deal with "for-loops."

You can also assign a variable to be an array[2]. We will look at manipulating arrays in another section.

Changing the name of variables is usually redundant but can be done if you type it in the following format: `new_name = monkey`. Now `new_name` will have a value of 8.

Note that Matlab will throw back the value of the variable unless you end the line with a semi colon, "`;`". Always remember to do this especially when working on lengthy pieces of code.

Also remember that Matlab variable names must begin with a letter, which may be followed by any combination of letters, digits, and underscores. Matlab distinguishes between uppercase and lowercase characters, so `A` and `a` are not the same variable.

When naming a variable, make sure you are not using a name that is already used as a function name, either one of your own functions or one of the functions in the Matlab language. If you define a variable with a function name, you will not be able to call that function until you either remove the variable from memory with the `clear` function. For example, if you enter the following command, `disp = 5`, you will not be able to use the Matlab `disp` function until you clear the variable with `clear disp`.

3. Editor window

The Editor window is very different from the command window because here you can make all the mistakes you want and Matlab will never shout at you. It is best to code in the editor window and run the program in the command window. Avoid coding in the command window, as it will be a huge pain to write programs like that.

[2] A series of values or a matrix

The advantage of coding in the editor window is that Matlab tells you when you make a mistake, indicating it by an orange or a red line. An orange line means that you can run the program (but Matlab is unhappy because of your bad coding practice) and the red line means that the program cannot be run because of an error in that line. I will try and explore common sources of error in Matlab in a later section.

You also necessarily have to make sure that you have saved your code in the same directory as what Matlab is looking in when running the code in the command window. I would recommend creating a Matlab folder in a recognizable region of your hard drive and saving all .m (read as "dot M files") files there. When obtaining codes from other people, you could just save it in this directory and run the file in the command window. We will explore running .m files in the command window in a later example.

If you have a very clear code in your mind, you can actually type it all out using notepad, in your email portal or even write it out on a piece of paper. The only reason you actually need Matlab is to see how it all works and to get your data.

4. How to write a program

To make our lives easy, we can write a program with user-defined variables. This means you can use it like how you use excel. In excel, you feed in a formula in a cell, and change the input in your "input cell" and the program spits out an output.

You can do the same thing with Matlab, and this section will teach you how.

To do this, we need to make use of the editor window.

Problem: Add 2 numbers, input 1 and input 2.

Solution:

```
%Type the following in the editor window

function [solution] = exmp_4(inp1,inp2)

solution = inp1 + inp2;

end
```

For the program to work you have to make sure you save the program as "exmp_4.m" in the directory that the command window is running in. You can see where Matlab is looking for the .m files under the "current directory" text field in Matlab.

Once you do that, you can go to the command window and type in `[solution] = exmp_4(1,2)` or `[solution] = exmp_4(124,268)` to get your answer. A new variable, "`solution`", will now appear in your Workspace window. This variable contains your answer.

When running programs in the command window you can change the variable name and it will just replace "`solution`". For instance, `[A] = exmp_4(inp1,inp2)` will give you the same result, but your answer will be saved under the variable A.

Alternatively, you can just type `exmp_4(inp1,inp2)` for Matlab to spit out the answer, given that you add the following line in your .m file: `disp(solution)`. Now, displaying the solution has become a function of your code, and you don't have to call for the answer. Try to accustom yourself with these techniques, because you will be using them for every code you write. See Exercise 1.

5. Presets

It is good coding practice, and also a good way to avoid orange lines, to preset values for the variables you are planning to use. Matlab doesn't like it (but at times will perform the program without complaining) when you show it variables that it has never seen before. You have to say what the initial value of the variable is. If it is an array, it is good practice to set up an array of zeros of the required dimension.

6. Commenting

Commenting your code is a very important yet undervalued practice. You can always type a `%` sign on your work in the Editor window (also possible in the command window) and Matlab will ignore whatever that comes after that. Note that you have to type in the `%` sign for every new line of comment you insert. This is a great way to keep track of what you are doing and it makes the lives of those who are looking at your code that much easier (like your TAs).

7. Arrays

You will have to use arrays extensively in your projects and it would be helpful if you were familiar with the different commands to manipulate arrays. There is a comprehensive list of tricks in the following website:

http://blinkdagger.com/matlab/matlab-tips-and-tricks-on-manipulating-1-d-arrays-vectors/

I would strongly advice you to play around with them before attempting to work on the projects.

Just to make this article complete, I have included a few basic array manipulation techniques here.[3]

In addition to doing operations on single numbers, Matlab allows us to perform operations on arrays. There are several ways to create an array, I have listed examples in the bullet points:

[3] I have modified the following material from: http://physics.gac.edu/~huber/matlab/mtlabfun.htm

- Explicitly listing terms
  - `array = [1 4 9 16]`

- Colon notation,
  - `array = 1:5`, creates an array with elements from 1 to 5 using intervals of 1.
  - `array = 1:2:10`, creates an array with elements from 1 to 20, using intervals of 2.

- Using the zeros and ones commands,
  - `array = zeros(1,3)`
  - `array = ones(1,5);`

We can do addition, subtraction, multiplication and division of an array and a value using the (+, -, *, /) operators.

Try this: Create an array T, for example, `T =  [1 4 9 16]`. And try out the following functions in the command window:

- `T*2`
- `T/5,`
- `T+20,`
- `T/10+5`

To operate on each element of the array individually you have to put a period in front of the operator.
- `.*`
- `./`
- `.^`

To demonstrate this, try the following example in the command window:

```
x=1:3;
y=2:4;
```

Now observe the results for the following:

- `x.*y`
- `x./y`
- `x.^y`

And now try to perform any one of those functions without the period. It doesn’t work because without the period, Matlab tries to perform a matrix operation. The dimensions of x and y are inappropriate for such a function.

A row vector can be turned into a column vector using the transpose (') operator. Using the definition of t above, note what happens for the command t' and t''.

We can also extract and modify elements of a list:

- Selecting an element of a list, use `x(3)`
- We can set an element using `x(3)=100`;
- To select a range of an array, use `x(3:5)` or `x(:3)` or `x(5:)`

See Exercise 5.

8. for loops

“for loops” will repeat a series of statements a specific number of times. They help us by making Matlab do all the repetitive work for you. Let’s solve a simple Matlab problem.

Problem: We want to find the sum of numbers 0 to 50. So we want to find 0 + 1 + 2 + 3 + … + 49 + 50. We can use the loop construct here.

Solution (an example of coding in the command window):

```
>> a = 0;

for i = 1:50
    a = a + i;
end
>> a

a =

       1275
```

Note that I have preset the variable “`a`” to be 0 *before* the loop begins. This means that on the first iteration[4], `a = 0` [5]. So the new value of variable “`a`” is given by “`a = 0 + i`”; where “`i`” refers to the iteration number so our first iteration number is 1, because we have assigned the “for loop” to run such that i increases by one for each iteration starting from 1. So Matlab will add 0 + 1, put it in the variable a, and reach end. The loop will make Matlab go back to the start of the loop and once again perform `a = a + i`, just that this time, `a` (on the right) `= 1` and `i = 2` (for the second iteration). This goes on 50 times and you get your answer of 1275. Note that if you put the preset inside the “for loop”:

```
>> for i = 1:50
a = 0;
a = a + i;
end
```

[4] I will use the term “iteration” to refer to the loop number. First iteration will refer to the first repetition.

[5] Note carefully the position of the variable, and the position of the value assigned to that variable.

Then we have for every iteration `a = i`. And we will end up with a = 50 (as the last iteration number).

The easiest way to add these numbers would be of course:

```
>> sum (1:50)

ans =

        1275
```

9. if constructs

The "if" construct is a simple, yet very useful function in coding. Let's look at another problem.

Problem: Identify if a number is below 25 or between 25 and 75 or above 75.

Solution:

```
function exmp_7(inp)
if inp < 25
    disp('less than 25')
elseif 25 <= inp && inp <= 75
    disp('between 25 and 75')
else
    disp('above 75')
end
```

Although this could be an insultingly simple example, there are a few important points to note. The "elseif" within the "if" construct will enable you to add another condition, in case the first one isn't satisfied. The "else" statement, gives the construct the option of performing the actions that follow "`else`", in the event that all the previous conditions weren't satisfied.

Now, note that Matlab does not accept statements like "`25<inp<75`". You need to specifically mention that "`inp`" must fall between exactly these two intervals using the "`&&`" symbols. You have to use the symbol twice because you are specifying a condition. Note that you will also have to use "`==`" if you want to say that something is equal to something in your condition. You can only use "`=`" when you want to store the value on the right hand side (RHS) into the variable on the left hand side (LHS).

In one of the exercises you are required to make an "if statement" work inside a "for loop". Think through your process carefully before proceeding with the coding for that.

I would strongly recommend you to take a look at the help section for "if" to make yourself accustomed to the various less than and greater signs that you can use to obtain your desired if statements.

See Exercises 2 and 3.

10. Switch Case

With the following example, I will illustrate how the "switch case" method can be used as an alternative to a series of "if elseif" statements. Let's say we have a similar problem as before.

> Problem: We are told that we will be given an input, which can either be 1, 2, 3 or something else. We need to make a program that will tell us which is which.
>
> Solution:

```
function exmp_10(inp)
switch inp
    case 1
        disp('one')
    case 2
        disp('two')
    case 3
        disp('three')
    otherwise
        disp('others')
end
end
```

Notice, how you could have also tackled the problem using a series of "if elseif" statements too, but the "switch case function" may prove to be easier and cleaner. You might want to think of the switch function as a train-track-directing switch. In the line "`switch inp`" you tell Matlab that you are looking for the variable "`inp`". In the following statements, you give a list of cases detailing where the train should be directed if one of the conditions are met. You may want to use this method to solve the final exercise problem.

11. while loops

A "while loop" is a combination of a "for loop" and an "if" construct. In a "while loop", before another iteration begins, your code will ask if a certain condition is satisfied before it continues with the loop. It can be articulated as "Keep running the loop *while* this condition is satisfied."

To give a very simple example, let's say, I ask you to count from 1 to a varying number (randomly assigned by me), and record the counted numbers in an array. You would probably write the following program:

```
function exmp_8(num)
array = 0;
```

```
i = 1;
while array(1,i) < num
    i = i+1;
    array(1,i) = array(1,i-1)+1;
end
disp(array)
end
```

Although there is a much easier way to generate this array:

```
function exmp_8-2(num)
array = [1:1:num]
end
```

I have used the first solution for a few reasons. It demonstrates that you need an artificial iteration number in a while loop, unlike the "for loop". Since you have to make your own iteration number, you also have to manually make the number go up by one each time the program goes through the loop. Note that, this isn't always necessary in a "while loop", and that I had to make use of this little trick just to make a coherent array.

Also, I chose this example because you can't always expect to have such nice increments. You might face a problem with varying increments, like a Fibonacci sequence. In fact that is an exercise included in this article.

Since this isn't a fantastic example, I also had to commit a subtle crime of creating an array indefinitely increasing in size (which triggers the orange line in the editor window). But, I get away with this because I know that this isn't exactly indefinite, and that I know the loop will stop when I have counted enough numbers.

See Exercise 4.

11. Strings

You can make Matlab recognize, store and recall text. This will come in handy when you are analyzing long strings of text. For example, we could count the number of times the word "the" is used in a storybook using strings[6]. I think the following website does a great job of explaining how to make use of strings.

http://en.wikibooks.org/wiki/MATLAB_Programming/Strings

You might be at one point required to make use of the Map data structure. A Map is a type of fast key lookup data structure that offers a flexible means of indexing its individual elements. Unlike most array data structures in Matlab that only allow access to the elements by means of integer indices, the indices for a Map can be nearly any scalar numeric value or a character string (words).

[6] Remember that strings is just a fancy way of saying, I have stored a word in Matlab.

Indices (or elements) of a Map are called *keys*. These keys, along with the data *values* associated with them, are stored within the Map. Each entry of a Map contains exactly one unique key and its corresponding value.[7]

At this point you might want to look up what containers.Map does. You will be able to use this to make a dictionary to reference a certain string to another string.

Don't worry too much about this section as I plan to make another set of more detailed notes if and when you are faced with such a project.

12. Debugging

No matter how careful you are with your codes, you are at some point bound to have made some mistakes in them. This can be either in the form of an error, in which case Matlab will not perform the task, or in the form of nonsensical results. There are a few ways you can solve the problem.

- Running codes in your head

  You can go back to the editor and scan for the mistake and try running the program in your head a few times. This involves mentally running through the loops.

- Using the command window for debugging

  Another technique follows from what I mentioned in the first section. You can run the entire code in the command window.

  Copy everything after the "`function`" line until just before the last "`end`" line, and paste it in the command window and press enter. But before this, remember to define your input variables.

  In reference to exmp_8-1, you will have to define `num` to equal something before pressing enter.

  This will enable you to see all the different variables, like `array` and `i`. They will appear in the workspace window so that you know what values are being used to generate the output that you are getting.

  This will give you an idea of how Matlab is dealing with your data. This can sometimes be quite tedious, but unfortunately it is the only way to be sure that your code is good.

13. Common Sources of Error

You will, unfortunately, be seeing little red bars on your editor window on the right. As mentioned before, this means the program cannot be run. I

[7] From http://www.mathworks.com/help/techdoc/matlab_prog/brqqo5e-1.html

have tried to explore some common sources of Matlab errors that you might want to check for first in the event that you are faced with this problem once you have completed your coding.

Firstly, don't try to code up something that looks smart enough to work, without completely understanding how it works. I have tried this before, and trust me, that's not how you code, and you will have no idea if the data you are producing is right or wrong.

- Omitting operators in expressions

  Remember that MATLAB does not understand a statement like "`b(3 + a)`" to mean "`b` times the quantity `(3 + a)`". Rather, MATLAB interprets it as "the `(3+a)`th element of `b`". This interpretation is not likely to give you the result you are expecting. To produce the desired result, be explicit about the multiplication operation: `b*(3 + a)`.

- Presetting

  As you might have already noticed in my example on the previous page, it is good practice to preset your variables. It isn't good to scare Matlab by suddenly introducing new variables in the middle of the code.

  For example, let's say you suddenly realized that you want to include a variable "`X`", an array, who will change with each iteration of a for loop. Typing in "`X(1,i) = i*2`" inside a for loop, for example, will work. However, you need to tell matlab how big X is first. I usually include presets immediately after the first line of my code. In this case, if you are doing a hundred iterations of the for loop, you should preset X to be, "`X = zeros(1,100)`". This isn't always a fatal error, but you will get orange lines in the editor page.

- Keeping track of iteration variables

  At times, you will have to use a "for loop" within a "for loop" within a "for loop", and so on. It requires some mental gymnastics to keep track of such things so I would suggest you reserve the letters: i, j, k and l for this purpose. Also note exactly where you are using these letters in your code.

  It is best to mentally map out a few iterations to see how the for loop goes before you run the program to get an overall feel for it. <u>Remember that you are using Matlab only for repetitive work and for remembering things. The brain of the coding is still you! If you don't understand it, Matlab won't work.</u>

- Brackets

  When, inserting complicated equations, Matlab might not be the most helpful thing in the world. If you have had experience inserting complex equations in wolfram alpha, you know what I am talking about. Keep track of your brackets, because sometimes you will have to open 6 or 7 pairs of them.

- Memory

  Try to keep your programs as simple as possible. The more complicated they are, the longer Matlab will take to process the data. Get rid of redundant lines, and look for the simplest methods. Matlab's command window will appear busy (on the bottom left) when it is running the code. If a project that we assign you is going to take a long time to run, we will let you know in advance.

  Although Matlab is a very powerful tool, you need to remember that it has a limited amount of memory. It is foolish to keep record of redundant data. Always begin with the end in mind and remember what final data you are interested in. Try to make Matlab "forget" as much useless data as possible and only keep the essence of your experiments. This will speed up processing time significantly and help you write succinct codes.

  Be on the lookout for lengthy periods of busyness. This usually means that you have gone about the problem in a long-winded way and Matlab isn't going to give you the solution anytime this week.

- The semi colon

  This is usually a very annoying problem until you get used to Matlab. You have to make sure that you type a colon at the end of each line of "action lines" (unlike "conditioning lines") so that Matlab doesn't display all the values that it has changed. You will be alerted that you are missing a semi colon with a yellow bar in the editor window.

  You have to be especially careful of these semi colons, when dealing with large arrays. Or else, you will see a "ginormous" array, containing, thousands, maybe even millions of elements, printed on your command window screen.

14. Elegance

This isn't a programming course but a course on data analysis. You have to remember that Matlab is only one amongst the many skills we teach you here, and this course isn't about mastering Matlab.

So we don't care how you obtain your findings as long as you present it in such a way that you demonstrate knowledge of the underlying concepts. There are very many solutions to each problem you will be assigned and there is no "one right answer".

With that said, there are of course only a few *elegant* solutions. And I, personally, am a big fan of elegant coding. I mention this here because this is what has helped me really enjoy data analysis. Working on solutions over and over again searching for a faster and better way to process data, even after a solution has been obtained, will help you hone your skills in programming. The more time you invest in each problem, the quicker you will get when faced with brand new problems.

We will try as much as possible to reward elegant solutions, but don't worry too much if you still prefer the brute force methods. This section is just a note to encourage you to do that extra bit more to become a better engineer.

***

And that is about all that I have for you in these introductory notes. You won't need these notes past the first couple of weeks of using Matlab. Once you have gotten accustomed to the coding platform, solutions to your projects and the ease of solving will no longer be a matter of your Matlab knowledge. Rather it will come down to you understanding the problem, the concepts taught and a bit of coding ingenuity. It will not be a matter of the amount of cool functions you know in Matlab.

Also, I have tried my best to get rid of mistakes in this article, but by no means is it perfect. Please don't hesitate to let me know if you find any mistakes or inconsistencies.

Feel free to contact me or the other TAs if you have any further questions about the course. We will be happy to help you.

All the very best!

Exercises[8]:

1. Gas[9]

An equation of state for a non-ideal, which is commonly used, is the van der Waals equation for 1 mole of gas:

P = (R*T)/(V-b) - a/(V^2)

where,
P = Pressure in Atmospheres
R = 0.0821 Atm*Liters/Mol*Kelvin
T = Temperature in Kelvin
V = Volume (in Liters)

The a and b values are[10]:
Ideal Gas: a = 0, b = 0 (so it reduces to ideal gas equation)
Nitrogen: a = 1.408 b = 0.0386
Oxygen: a = 1.378 b = 0.0318

· For T=300 K and V=10L, calculate the pressure for nitrogen and oxygen and compare the results with ideal gas.

· Plot the pressure for oxygen & nitrogen for a range of volumes from 0.1L to 10L (in steps of 0.05L) and compare to ideal gas

(To get you guys warmed up, below is the solution to the first problem.)

Solution:

```
%Part 1
function [P] = ex_1_1(T,V)

a = [0 1.408 1.378]; %ideal gas, Nitrogen, Oxygen
b = [0 0.0386 0.0318];

P = (0.0821*T)./(V-b) - a/(V^2);
end


%Part 2
function ex_1_2(T)
i = 0;
a = [0 1.408 1.378];
b = [0 0.0386 0.0318];
array = zeros(3,199); %preset

for V = 0.1:0.05:10
    i = i+1; %artificial counter
```

8 These problems are somewhat in order of ascending difficulty.

9 http://physics.gac.edu/~huber/matlab/mtlabpr1.htm

10 http://en.wikipedia.org/wiki/Van_der_Waals_constants_%28data_page%29

```
        P = (0.0821*T)./(V-b) - a/(V^2);
        array(:,i) = P;
    end
    x = 1:199;
    plot (x,array(1,:),'k',x,array(2,:),'y',x,array(3,:),'b')
    %see help plot
    end
```

2. Simple coin toss

   Find the percentage of heads occurring in a fair coin thrown 50, 100, 1000 and a million number of times respectively.

3. For and if

   Generate an array of 10 random numbers between 1 and 100. Then find out how many of the elements are between 1 and 25, how many between 25 and 75 and how many between 75 and 100.

4. The Fibonacci Finder

   Create a program that will distinguish a Fibonacci number from a non-Fibonacci number. *Hint: This is a review of while loops and if constructs.*

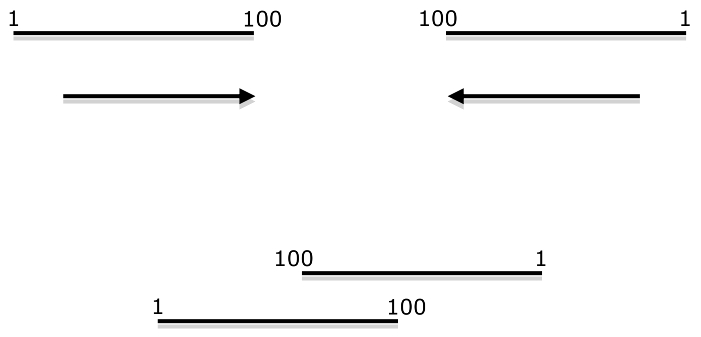
5. Discover Pi

   Imagine the first quadrant of a circle in a Cartesian vector space. Pick a few random points within this space and calculate the ratio of the points that lay within the circle to those that are outside the circle. Using this, determine the area of the circle, and calculate pi. Increase the number of points within the space and see how it affects your value of pi. You should get a fairly accurate value after picking about a billion points. (The method is known as the Monte Carlo Simulation)

6. Fold Mountains

   (a) Imagine two arrays, each given by: [1:1:100], overlap each other (See figure below). The "overlap depth" is random: a number, between 1 and 100.[11] Overlapping elements should be added together. Generate a plot of the resulting array from such a collision. You will notice something remarkable. This should be an exercise in array gymnastics.

[11] Overlap depth of 1 would mean, the two 100s at the ends overlap, and overlap depth of 100 would mean the two arrays overlap entirely.


1
100
100
1
100
1
1
100