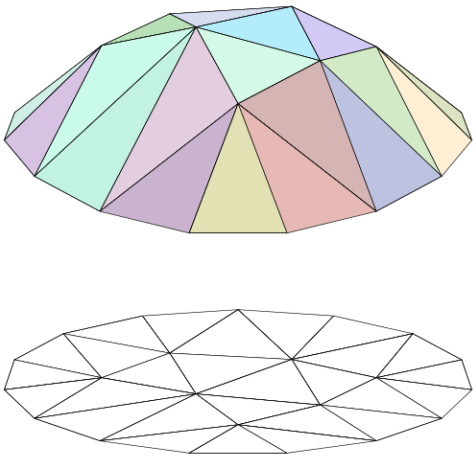
An Introduction to



Mathematica Programming

for Undergraduate Research

Robert D. French Dr. Samuel Jator

December 21, 2012

Contents

Preface

This book is written especially for the students of MATH 3120 and 3130 at Austin Peay State University in Clarksville, TN. It is intended to introduce the reader, whether versed in programming or not, to the basic elements of *Mathematica*.

Computer algebra systems and symbolic programming provide a valuable means of exploring research topics at the undergraduate level. In addition, *Mathematica*'s facilities for functional programming provide a means for students to learn to solid programming habits.

This book features an introduction to basic usage of *Mathematica* for those who have never used it before, and for those who have never done any programming before. It is our aim that this preliminary section "If you have never used *Mathematica* ..." will provide enough background to get started with the remainder of the book.

We hope that this book will serve to clarify and make *Mathematica* a more accessible programming language. We hope this book illuminates the joy of programming, and that those who read this book will go on to publish research of their own.

Samuel N. Jator & Robert D. French
Austin Peay State University
December 21, 2012

If you have never used *Mathematica* ...

This pamphlet assumes you have done at least a little programming before, possibly in C, C++, FORTRAN, Visual Basic, or some other language. If not, *that is completely fine*. We will walk you through some simple programming concepts here so that you can get your feet wet, and then you will be ready to take on chapter 1.

If you have done some programming before, but have never used *Mathematica*, it will still benefit you to spend some time with this section, because it will walk you through some tricks to make using *Mathematica* more pleasant.

1. Notebooks and Cells

When you start *Mathematica* for the first time, it will create a new *notebook* for you. A notebook is just a type of document, like a Microsoft Word document, but it is specially designed to make *Mathematica* programming easier.[1] Notebooks are broken into cells, which you can think of like paragraphs. Each cell can have code in it, and they can be evaluated[2] independently.

You can create new cells by clicking in a blank region of the docoument. You will see the cursor displayed sideways, and that is *Mathematica*'s way of telling you that the cursor is in between cells or that it is not inside a cell. As soon as you start typing anything, a new cell will be created, your cursor will hop inside, and anything you type will be displayed there.

Cells can have different "Styles" depending on what you want to use them for. By default, when a new cell is created it will be an "Input" cell, which means you can type code in it and evaluate it. By right-clicking on the vertical bar on the right side of a cell, you can bring up a menu that will allow you to change the style of the cell. Changing your cell style to "Title", "Section", "Subsection", etc. as appropriate can help you organize your work and will make your presentations look more professional.

When you type some code in a cell, you can evaluate it by pressing "Shift+Enter"[3]

2. Fancy Typing and the Palette

When you start *Mathematica*, it should load a set of buttons in a thin vertical window to the right of your notebook. If it does not, you can access this palette

[1] Note that for most programming languages, you type your code in a *plain text* document, or one that has no special formatting in it. This is not the case in *Mathematica*; if you open a notebook in Vim or Notepad, you will see that your code is surrounded by lots of formatting directives that you probably don't want to type by hand!

[2] To "evaluate" some code simply means to run it.

[3] In case you are not familiar with this convention, it means to hold the "Shift" key and then press the "Enter" button

Symbol	*Mathematica* code
π	"Esc" pi "Esc"
α	"Esc" alpha "Esc"

TABLE 1. A table of symbols

by going to "Window" → "Palettes" → "Basic Math Input" in the menu bar. The palette gives you ways to make your code look more mathematical. For example, there are buttons for writing integrals, summations, exponents, matrics, greek letters, etc.[4]. If you press any of these buttons, it will insert the necessary symbols into the current cell and you will see several small boxes in the positions where one would expect symbols. Pressing "Tab" will allow you to move between these boxes, and type any valid *Mathematica* expression in them.

2.1. Subscripts, Exponents, and Fractions. There are also keyboard shortcuts for many of the symbols in the palette. For example, to type a fraction, you can press "Ctrl+/" and your cursor will automatically be placed in the numerator. To type the fraction $\frac{2}{3}$ in *Mathematica*, you simply need to press "Ctrl+/" and then "2", "Tab" (which moves you to the denominator), and then "3".

Likewise, to type x^3, you simply type "x" and then "Ctrl+6" and then *Mathematica* will give you a black box above and to the right of the "x" character and move your cursor there. Then type "3", and you will have x^3.

2.2. π, θ, and *alpha*. Using the "Esc" button, you can generate these symbols. "Esc+pi+Esc" will get you the π symbol. You can look at Table 1 to see a longer list of symbols.

2.3. Sums, Integrals, and Derivatives.

3. Functions

Mathematica has tons of built-in functions. It has trig functions like `Sin` and `Cos`. It also has functions for plotting graphs and generating lists or tables of data (see "Plotting" and "Lists and Map").

[4]Indeed, many of the basic features of LaTeXare available in *Mathematica* to help you typeset mathematical expressions, but you will still need the full power of LaTeXin order to prepare your research for publication

CHAPTER 1

A Review of Some Stuff You've Probably Seen Before

Before beginning this chapter, it is assumed that you know a few things about programming already. Specifically, you will need to know how to

(1) assign values to variables
(2) evaluate *Mathematica* cells

and if you don't, that's quite alright. Just see the section "*If you have never used* Mathematica *...*" on page iii, and even if you have done a bit of programming before, we will walk you through some basic *Mathematica*.

1. Variable Assignment

The first thing we need to know in any language is how to assign variables. Probably you are thinking "*I learned this in CSCI 1010!*", but *Mathematica* is a subtle language, and does not always work as you might expect if you are coming from C++ or FORTRAN.

```
(* Cell 1 *)
i = 1;
j = 2
k := 3;
l := 4
```

Looking at this example, we can see that there are four slightly different ways to "assign" values to a variable, so let's discuss this a bit. If you put this code into a *Mathematica* cell, you will see that, upon evaluating the cell, it will output:

```
2
```

So, if we assigned four numbers to four variables, why do we only see one output? The are two key items here. One is the semicolon (;) at the end of the first and third lines. This tells *Mathematica* to suppress the output of that calculation. Generally, it is appropriate to put a semicolon at the end of every line of code in a cell except for the last one. This is because you *usually* want to group your code into cells in such a way that each cell achieves one result, computes one item (or related set of items), or builds one data structure. When debugging, it can sometimes be handy to remove individual semicolons in order to investigate whether each line behaves as you expect.

The second key item is the `:=` operator. This is called the *Set Delayed* operator. This is different from the standard `=` sign[1] in one important way: it does not assign the value to that variable immediately. Rather, it tells *Mathematica* to wait until `k` or `j` is used and then evaluate the right hand side of that expression.

[1] also called the *Set* operator

What that means, in terms of the above example, is that, at the moment of evaluation, `i` and `j` are numeric variables that contain the values 1 and 2, but `k` and `l` are just symbols that do not yet contain any value. This might be hard to see with the previous example, so let's look at one that's slightly more involved:

```
(* Cell 2 *)
i = 1;
j = i + 10
k := i + 10
```

When we evaluate this code, we see that the output is simply `11`[2]. Now we set up a short experiment: We create a few separate cells, and examine what happens to `j` and `k` when we change the value of `i`.

```
(* Cell 3 *)
{j,k}

(* Cell 4 *)
i = 2;
{j,k}

(* Cell 5 *)
i = 20;
{j,k}

(* Cell 6 *)
i = x;
{j,k}
```

When we evaluate these cells, we note that they each produce an ordered pair (also called a *list*) as output. This is just to help us see what happens to `i` and `j` when we change the value of `i`.

In Cell 3, nothing interest happens. We note that `i` and `j` are both `11` like we expected them to be. But now let's evaluate Cell 4 and see what happens. The output from Cell 4 tells us that `j` is still `11`, but that `k` has been updated to reflect the new value of `i`. This is because the definition we gave for `k` in Cell 2 is re-evaluated *every time* we use `k` in an expression.

If we evaluate Cell 5, we see the same thing has happened: `j` is still `11`, but `k` is now `30`, reflecting the fact that we changed the value of `i` again.

When you evaluate Cell 6, you will see that is does something slightly different. The explaination is simple, but we leave it as an exercise so that you are forced to think about it!

Delayed Evaluation is most frequently used when defining functions (See Defining Your Own Functions). For now, this is as far as we need to go with this topic. You might be thinking: "*Then why did we even bother?!*" but, the misuse of this operator is the cause of many bugs, and much of the *Mathematica* code you are likely to find on the internet contains a wild `:=` when it shouldn't. Straightening this out now will spare you headaches, and will promote friendship between you and your code.

[2]Using what we discussed earlier, can you figure out why this is? See problem 1

2. Invoking Functions

In *Mathematica*, the *invocation operator* is `[]`.

Exercises.

(1) Explain why the code in Cell 2 produces only the single output of `11`
(2) Explain why Cell 6 produces {`11,x + 10`} as an output.

CHAPTER 2

The Kernel, Variable State, and Scope

Before beginning this chapter, it is assumed that you know a few things about *Mathematica* programming already. Specifically, you will need to know how to

(1) assign values to variables
(2) evaluate *Mathematica* cells

and if you don't, that's quite alright. Just see the section "*If you have never used* Mathematica ..." on page iii, and even if you have done a bit of programming before, we will walk you through some basic *Mathematica*.

1. What is the Kernel?

In *Mathematica*, all of your coding is done in a *notebook*, and all of the output of your code is displayed there as well. However, the calculations themselves are done in *an entirely separate program*[1], and this program is called a *kernel*. There are many reasons for doing these calculations in a separate program from your notebooks:

(1) You can still edit your notebooks while long computations are running
(2) You can share variables and data between notebooks
(3) You can manage multiple kernels, (and thus multiple long-running computations) from a single notebook
(4) You can run computations on multiple kernels *on other computers*

So we see that this separation of kernel and notebook is very powerful. But what does it mean in terms of your research? Specifically, while you are working on your code, the values you calculate and the variables you assign them to will be stored in your "Local Kernel". This assignment of values to variables is called "State", and it's just a fancy computer science term for "The values of your variables at a given time".

1.1. Quitting the Kernel. . This is kindof like an emergency reset for your program. Quitting the Kernel will basically erase the values for all the variables in your notebook (because they are stored in this separate program which you are about to quit).

[1]This is an example of a Service in a Service Oriented Architecture, and if you are interested in software engineering, you should check this out.

CHAPTER 3

Defining Your Own Functions

1. Plotting Your Functions

CHAPTER 4

Lists and Map

This chapter introduces what computer scientists refer to as *functional programming*. To understand some of the impact of this, we begin by discussing the fundamental data structure of functional programming, the List.

A List is similar to an *array* that you might have encountered in other programming languages. One of the main differences is that Lists are designed to grow, whereas arrays are designed to take up a fixed amount of memory.

Lists in *Mathematica* can be constructed very simply by the following statement:

```
A = List[];
```

or equivalently

```
A = ;
```

however, the former style should be preferred as it is more explicit[1]. Lists can be grown by *appending* to elements to them. For example, in order to create the list $1, 2, 3, 4$, we could do the following:

```
AppendTo[A, 1];
AppendTo[A, 2];
AppendTo[A, 3];
AppendTo[A, 4];
```

Of course, we could also define this list explicitly as follows:

```
A = 1,2,3,4;
```

and this is of course much more concise. Generally, if a list can be defined without doing any calculations, i.e. if it is a constant, you will define it all at once as we have done here. However, if the list must be built up programmatically, it is necessary to use the `AppendTo` function as described above.

[1] Good programmers always aim for their code to be *clear* and *explicit*. This makes it easier for others to read their code and understand its meaning, and that is good for friendship

CHAPTER 5

Plotting

1. **ListPLot**
2. **3DListPlot**

CHAPTER 6

Symbols, Expressions, and Replacement Rules

1. Replacement Rules

. When you solve a system of equations ...

2. Transforming Expressions

. You can apply a set of replacement rules to an expression...

3. Performance Issues

. If part of your code that uses replacement rules seems to be running slowly, you can speed things up by preparing a *dispatch table*.

APPENDIX A

Finding Help in *Mathematica*

1. Documentation Center

Shift+F1.

2. Stack Overflow

One of the greatest resources for *Mathematica* help on the internet is StackOverflow/Mathematica. Also becoming very popular is mathematica.stackexchange.com, which is a separate site dedicated enitrely to *Mathematica* issues. Both of these should serve you well in your quest to find answers to your problems.

StackOverflow is a forum in which users are encouraged to give insightful answers in order to receive *points*, which amount to social capital. Thus, the answers you will find on StackOverflow are consistently of a higher quality than those you will find on other programming-related websites.

On StackOverflow, all questions are *tagged* according to which programming language or platform they pertain to. The link above will take you directly to the *Mathematica* questions, and from there you can search for more specific information about your question.

Generally, the probability that the issue you have run into when programming in *Mathematica* (or any language) is unique is very low, so the odds are in your favor that someone has encountered a very similar problem before you. Thus, the challenge is to alter your search terms judiciously until you stumble on a problem that seems to fit the issue you are dealing with.

One way to help with this process is to speculate a few guess about what the problem might be. For example, are you using some functions whose behavior you don't quite understand? Maybe you getting a weird output that doesn't look like what you think it should? Searching on StackOverflow for things like "ListPlot no graph" is more likely to get you useful results than "no graph".

2.1. Asking Questions on Stack Overflow. Accounts on StackOverflow are free, and joining this community will help you learn a great deal about both *Mathematica* and programming in general. Searching through other people's questions and the suggested answers can be very informative, and it is a good intellectual exercise to try to solve some on your own (and you may even be rewarded with profile points!).

Of course, membership on StackOverflow also allows you to post questions. Understand though that the community expects you to have done some work before you post a question. Generally, before posting new questions, it is advisable to:

(1) Try your query on a major search engine like Google or Bing. At least try all the links on the first page to see if they have anything helpful to offer.

(2) Try searching WolframAlhpa. While this is not exactly a search engine in the usual sense, its results usually contain *Mathematica* code which is occasionally helpful.
(3) Try the *Mathematica* documentation. You can access this by pressing Shift+F1 while running *Mathematica*. Specifically, you will want to look at the examples that are available on the documentation page for each function that you might have questions about. Frequently functions can take different arguments, or give different output depending on parameters or options, so it may be that you need to invoke your function in a slightly different way.

If you have done these things and still not found the answer to your question, then it will be cool to post your question on Stack Overflow. One thing to note is that while people on Stack Overflow are usually very eager to help, they need enough information about what you are trying to do to be able to understand where you might be running into trouble. Generally, posting a single line of code may not be enough. Also, keep in mind that folks on the internet are very unlikely to be familiar with your research, so it is important to track down the issue as specifically as possible.

APPENDIX B

Using GitHub to Collaborate with Teammates

If you are working on a research project with a team, you may decide simply to email the mathematica notebooks you are using back and forth every time either of you makes a change. This seems like a good idea, and indeed it is the simplest way to share changes, but it can lead to some surprisingly hairy problems very quickly.

For example, let us suppose that Nell, Ben, and Raman are on a team together. They start with one document, *research1.nb*, which contains all of their initial code that they plan to use for their project, and each person takes a copy of the document home with them. Later that week, Nell and Ben both make separate changes to the document and email their new documents to Raman. When Raman checks her email, she sees that Nell and Ben have both come up with neat ideas, but in order to get them to work together, she has to cut and paste their changes by hand. This seems straightforward enough, until Raman tries to run the code and she encounters a bug!

What might have happened? Did she make a typo while rearranging the code? Do Nell and Ben's changes conflict in some subtle way? Did the documents they emailed her match their latest changes (that is, did either Nell or Ben accidentally send an old document)? Emailing documents leads to many, many difficulties of this nature. Fortunately, programmers before us have encountered this same issue, and developed tools to solve it.

1. The Value of Version Control

Version Control is, in general, a way of keeping track of changes to files across a team of people (even if it is only a team of one!). Version Control systems are programs that you can install on your computer that will note the changes you make to your code in a way that allows you to access specific versions of your code at any time.

2. Using Git

Git is awesome.

3. Sharing Repositories on GitHub

Use GitHub or git left behind.

APPENDIX C

A Sample *Mathematica* Project: Deriving Euler's Method

Here we will look at some sample code and understand how one can derive Euler's Method for solving 1st-order ODEs. We will go through this piece by piece.