Sept. 7 2012

Lecture 2: Introduction to Mathematica

Expressions and Evaluation

There are very many ways to learn how to use MATHEMATICA® . Nearly all of the best ways involve performing examples from the very beginning. That is how we are going to start—with examples. Using MATHEMATICA® 's *FrontEnd* you may execute a command by pressing Shift-Enter; simply pressing Enter tells MATHEMATICA® 's that you merely wish to have a "carriage return" on the screen.

Mathematica's syntax will feel fairly natural after a while. Use the following notebook to get started. Execute a few commands until you get a sense for what output MATHEMATICA® will produce; try editing the commands; try to make MATHEMATICA® do something strange—just try playing with it and you will soon get the hang of what is going on.

One way to use MATHEMATICA® is simply as a calculator that allows symbols to get carried along. MATHEMATICA® will usually try to resolve every symbol and return precise information about it. If something is undefined to MATHEMATICA® , it simply returns it as a symbolic expression.

A number is not returned until all of the symbols in an expression are defined as numbers. MATHEMATICA® will try to be exact—it does not calculate $\frac{1}{3}+\frac{1}{2}$ by adding $0.33333\cdots+0.5=0.83333\ldots$, it has an algorithm for adding rational numbers and gives $\frac{5}{6}$.

Getting Started

There are a variety of ways to get MATHEMATICA® started and these are specific to the operating system your computer uses. A license must be purchased to run MATHEMATICA® code, but free MATHEMATICA® -display tools can be obtained from Wolfram.

The *FrontEnd* is the graphical interface between the user and MATHEMATICA® —you arrange your MATHEMATICA® input, sometimes with text-like comments, in the FrontEnd. The user must request the FrontEnd to pass something to MATHEMATICA® 's *kernel*, by pressing Shift-Enter. The kernel is the resident symbolic algebra software engine behind MATHEMATICA® .

The appearance of the FrontEnd depends on either provided or user-designed *StyleSheets*. Style sheets can be located under the *Format* menu.

Lecture 02 MATHEMATICA® Example 1

Basic Input and Assignment

Download notebooks, pdf(color), pdf(bw), or html from http://pruffle.mit.edu/3.016-2012.

The methods of assigning symbols (`SomeVariable`) to expressions via `SomeVariable = expr`. The `expr` can contain other symbolic variables, functions, programs, graphics, and many other things. There are important differences between exact (symbolic) objects and numerical objects. Logical equalities (`==`) are not assignments, but are Boolean operations.

Assigning values to symbols

```
a = 4 π / 3                                   1
UnitSphereVolume = a                          2
2 a                                           3
ANewVariable = (2 a + b) ^ 2                  4
ANewVariable^2                                5
b = 4 (3.14159265358979) / 3                  6
UnitSphereNumericalVolume = b                 7
ANewVariable                                  8
```

Differences between exact expressions and numerical expressions

```
UnitSphereVolume - UnitSphereNumericalVolume  9
a - 4 ArcCos[-1] / 3                          10
a - 4 ArcCos[-1.0] / 3                        11
2 Pi - 2 (3.141519)                           12
N[5 / 6]                                      13
```

Distinction between Equality (= =) and Assignment (=)

```
a == 4 ArcCos[-1] / 3                         14
a == 4 (3.14159) / 3                          15
```

1: A symbol is assigned to an expression with an equals sign `=`. Some symbols, such as π, are already defined—in MATHEMATICA® it is *exactly* the ratio of a circle's circumference to its diameter. Here, `a` is a symbol that could represent, for example, the volume of a sphere with radius 1—*and not an approximation depending on how many digits are used to numerically represent* π.

2: In my opinion, the variable `a` is not a very good name. We might forget what it represents, or try to use it again in a different context. I think it is much better to use descriptive names, such as `UnitSphereVolume`. Here, because there is an assignment in `UnitSphereVolume = a`, MATHEMATICA® tries to see if there are any other assignments associated with the right-hand-side, and if there are it uses them until all possible assignments have been made.

3: Because no assignment was made to `a` just above, its value is not changed.

4: The RHS in an assignment (here to `ANewVariable`) can contain unassigned symbols.

6–7: Here, the symbols `b` and `UnitSphereNumericalVolume` are assigned to an approximation to the unit sphere volume.

8: Note that, because `ANewVariable` contains `b`, the assignment of `b` above is reflected in the current value of `ANewVariable`: MATHEMATICA® will check to see if any symbol being output has been assigned.

9: To show the difference between the numerical approximation of π and the symbol π, subtraction shows that the difference is a very very small number.

10: Some functions can behave as *exact* if their values can be expressed exactly: here `ArcCos`[-1] is *exactly pi*.

11: Notice that the output here is different, showing that `ArcCos`[-1.0] has been replaced with a numerical representation because the function was executed on a numerical object.

14: The operator `==` tests to see if the LHS (left-hand side) and the RHS (right-hand side) can be determined to be equal, in which case it returns true.

15: If `==` can do so, it will return false if the two sides are not equal; otherwise if it can't say whether true or false, it will just return the statement itself.

Lecture 02 MATHEMATICA® Example 2

Building Expressions and Functions and Operations on Expressions

Download notebooks, pdf(color), pdf(bw), or html from http://pruffle.mit.edu/3.016-2012.

Sometimes it is easier to build up complicated expressions by entering shorter subexpressions beforehand. There are usually many ways to do the same thing in MATHEMATICA® , and this is demonstrated for functions. As you begin, pick the most simple method that works. Someday later you can pick up the alternative methods—they can be useful in advanced usage.

Mathematica Functions

```
a = 1 / Exp[x]                                          1
b = Cos[x]                                              2
c = (a + b) ^ 2                                         3
```

Alternative Syntax for Functions (There are many ways to do the same thing)

```
AnotherVersionofb = x // Cos                            4
YetAnotherVersionofb = Cos@x                            5
YetEvenAnotherVersionofb =
 Function[z, Cos[z]][x]                                 6
YetStillAnotherVersionofb =
 Function[Cos[#]][x]                                    7
FinallyAnotherVersionofb = (Cos[#] &)[x]                8
ANewVariable[x]                                         9
```

Mathematica Operations on expressions

```
c
AnotherVersionofC = Expand[c]                           10
c
Simplify[AnotherVersionofC]                             11
```

Calculus

```
IntegralofC = Integrate[c, x]                           12
Integrate[c / x, x]                                     13
```

Getting information (part 1)

```
? ExpIntegralEi                                         14
```

1–3: This is a simple example of building up an expression piece-by-piece. For very complicated expressions, this is much easier and less prone to typing errors.

4–8: One of the difficulties of learning MATHEMATICA® is that the syntax can appear to be very complicated and hard to remember. As you begin, just use functions in the form of `Cos`[x]. Here, just as a heads-up, other ways to do the samething are presented. We will use **4** sometimes in this course, because it is convenient. The most useful form is probably **8**, this invokes the concept of a *pure function*.

10–11: One of the powerful aspects of a symbolic algebra program is the manipulation of expressions. It's fast and it doesn't make mistakes as one might using pencil and paper. Here are examples of `Expand` (which expands all products) and `Simplify` (which uses an algorithm to choose among various forms).

12–13: Another powerful aspect is the ability to perform more advanced mathematics. The integral in **12** is one that *perhaps* you might have been able to do after one semester of calculus; **13** is one you would have to manipulate and look up in tables—the answer demonstrates that MATHEMATICA® knows about many many different functions.

14: If you see a symbol that you don't recognize you can either use the help-browser or ask the front-end directly. MATHEMATICA® has a fairly consistent function naming strategy The first letter of a word is always capitalized; compound words are concatenated together while maintaining the first letter capitalization; thus `InverseBetaRegularized`. A function is just another symbol—if a symbol is followed by square brackets `[]` the stuff inside the brackets become the argument(s) for the function.

Lecture 02 MATHEMATICA® Example 3

Calculus and Plotting

Download notebooks, pdf(color), pdf(bw), or html from http://pruffle.mit.edu/3.016-2012.

The derivative and integration methods are introduced. Simple plotting methods are demonstrated with an example of annotating a plot.

```
D[ANewVariable[x], x]                                                    1
Integrate[ANewVariable[x], x]                                            2
D[ANewVariable[x], z]                                                    3
tempvar =
 Integrate[ANewVariable[x], {x, 0, y}]                                   4
D[tempvar, x]                                                            5
D[tempvar, y]                                                            6
Factor[IntegralofC]                                                      7
IntegralofC
AnotherVersionofIntegralofC =
 Integrate[AnotherVersionofC, x]                                         8
c
D[IntegralofC, x]                                                        9
Factor[c]
Simplify[D[IntegralofC, x]]                                              10
Plot[IntegralofC, {x, 0, 10}]                                            11
Plot[{IntegralofC, c}, {x, 0, 10}]                                       12
Plot[c, {x, 0, 10}, PlotRange → {0, 0.0001}]                             13
Options[Plot]                                                            14
Plot[{IntegralofC, c}, {x, 0, 10},
 PlotStyle → {{Red, Thickness[0.005]},
    {RGBColor[0.2, 0.56, 1],
     Thickness[0.0075]}}, BaseStyle →
  {FontFamily → "Helvetica", FontSize → 24},
 PlotLabel → " A Function (Pretty Sky
    Blue)\nand Its Integral (Red)\n",
 AxesLabel → {"Value", "Argument"},
 ImageSize → 800]                                                        15
```

1: If MATHEMATICA® can't differentiate or integrate a function, it will be left in a symbolic form. `D` is MATHEMATICA® 's function to take derivatives.

2: If `Integrate` can't integrate a function, it will return the result in symbolic form.

3: MATHEMATICA® is rigorous about applying the rules of calculus...

4: Let `tempvar` be the result of integrating some function of `x` from `0` up to some arbitrary value `y`—the result should be a function of `y`.

5–6: MATHEMATICA® knows about the fundamental theorems of calculus...

7: The calculus operations will often create long and complicated expressions. That two expressions are equivalent can *sometimes* be shown with built-in functions such as `Simplify`, `FullSimplify`, `Factor`, `Expand`, `Collect`, etc., but sometimes it is an art to turn an expression into an aesthetic form.

8–10: Reassign `IntegralofC` to the *indefinite integral of c*—note, the constant of integration is set to zero. The integration-result is not necessarily left in the most simple form.

11: This is the simplest form of `Plot`. The second argument is a list giving the variable and its bounds. The first argument should have a numerical value at most of the points within the variable's bounds.

12: These two expressions ought to be the same; however, the output-result doesn't make this obvious.

13: Here, operations on the above expressions do show that they are same.

14: This is the simplest version of `Plot`—all it needs is the expression to plot and the range over which to plot a variable—in this case `x` from 0 to 10.

15: If we form a list of two expressions with {–}, then we get a curve for each expression.

16: MATHEMATICA® 's `Plot` has an algorithm to set the values of y-axis if it is not specified. To specify, one sends `Plot` and option in the form of a rule—here the rule is specified for `PlotRange`.

17: To find all the possible options for a function with their default values, the `Options` function provides a way to decipher what aspects of a plot can be changed easily.

18: Here is an example with a plot title, axes labels, different colors and thickness for the curves.

Lecture 02 MATHEMATICA® Example 4

Lists, Lists of Lists, and Operations on Lists

Download notebooks, pdf(color), pdf(bw), or html from http://pruffle.mit.edu/3.016-2012.

Lists are useful ways to keep related information together, and MATHEMATICA® uses them extensively. Lists could be created in MATHEMATICA® by using the `List` function, but they are usually entered in with curly-brackets {} and each *element* of the list is separated by commas. List elements can be about anything, even lists themselves.

```
AList = {a, b, 2, 7, 9, 1.3, π/2, 0}                                    1
Length[AList]                                                           2
Cos[AList]                                                              3
AList
AList[[2]]                                                              4
AList[[{3, 6}]]                                                         5
AList[[-2]]                                                             6
Sort[AList]                                                             7
Select[AList, NumberQ]                                                  8
Reverse[Sort[Select[AList, NumberQ]]]                                   9
Select[AList, EvenQ]                                                    10
Select[AList, PrimeQ]                                                   11
Perms =
 Permutations[Select[AList, ExactNumberQ]]                              12
Dimensions[Perms]                                                       13
Transpose[Perms] // MatrixForm                                          14
TranPerms = Transpose[Perms];                                           15
TranPerms[[3]]                                                          16
TranPerms[[1, 4]]                                                       17
IntList =
 Table[i, {i, 1, Length[TranPerms[[1]]]}]                               18
TranPerms // MatrixForm
TranPerms[[All, Select[IntList, OddQ]]] //                              19
 MatrixForm
```

1–2: Here is a simple assignment of a list to a variable and the operation of `Length` on the list.

3: Some functions, such as `Cos` here, are *threadable functions*; when called on a list-argument, they will produce a list of that function applied to each list element.

4: A list's parts (its elements) can be picked out in a variety of ways. The `Part` function has a shorthand double-bracket form.

5: If the argument inside the double bracket is a list of integers, the elements corresponding to those integers are returned.

6: Negative integers pick elements from the *end* of the list.

7: `Sort` returns a sorted list; it will also take a second argument to specify alternative sorting rules.

8–11: There are plenty of functions designed to operate on lists; here `Select` returns those elements for which the second argument (`NumberQ`, in **8**) evaluates to True. Functions can be applied sequentially, as in **9**, and the inner-most function is applied first.

12–13: A list's elements can be lists themselves. For example, a matrix is represented by a list of a list. In **12**, `Permutations` creates a list whose elements are all the permutations (which are list themselves) of the list on which `Permutations` was executed. And there even are higher-dimensional structures such as tensors. `Dimensions` is a useful way of learning about such structures.

14: Here, the *post-fix operator* for a function (`MatrixForm`) is used to change the way a matrix is displayed. *Note, the result is* ***not*** *a matrix, but a* `DisplayForm` *of a matrix.*

15: `Transpose` returns the result of taking a matrix and turning the rows into columns and vice-versa.

18: `Table` is a common way to produce a list; here, a list of integers as long as the first row of `TransPerms` is produced.

19: This is a fairly advanced example of extracting the odd-numbered columns of a matrix. The list `IntList` is simply the integers for each column; its odd-numbered members are selected and become the second (i.e., column) argument of the `Part` selection. The first argument is `All`, so the entire row is captured for each selected column.

Lecture 02 MATHEMATICA® Example 5

Rules (→) and Replacement (/.); Getting Help

Download notebooks, pdf(color), pdf(bw), or html from http://pruffle.mit.edu/3.016-2012.

A rule `leftvar` → `rightvar` is *similar* to assignment in that it associates a new symbol (`leftvar`) with something else, but the value is not assigned—it does not effect future values of the left-hand-side symbol. Rules are often used in conjunction with replacements and to set options in functions. Many of MATHEMATICA® functions, (e.g., `Solve`) return rules as a result.

Rules → and Replacement /.

```
ARule = a → π/3                                          1
a                                                        2
AList
AList /. ARule                                           3
SomeRules = {ARule, b -> π/12}                           4
AList /. SomeRules                                       5
a = SomeOtherSymbol;                                     6
AList                                                    7
StrangeRule = {Rational[x_, y_] :> y / x}                8
(AList /. SomeRules) /. StrangeRule                      9
```

Getting Help

Several methods of getting help are available.

1. Typing `?ExpIntegralEi` returned information about the symbol `ExpIntegralEi`. Typing `??FunctionName` gives even more information—try `??Plot`. Wildcards can also by used as demonstrated below. You can click on the resulting grid-list to pull up documentation.

```
? *Exp*                                                  10
```

Each of the above is linked to Mathematica's Help Browser.

2. Typing Options[Plot] returned a list of options that can be adjusted by the user until the result (in this case the appearance) of the plot is satisfactory.

Mathematica's Help Browser is a very useful tool and will probably become a primary resource for students. It contains good tutorials and demonstrations that can be copied and pasted. It has very good and concise descriptions of mathematics; in fact, exploring the Help Browser is a good way to explore mathematics as well as Mathematica. For instance , the discussion of "Combinatorial Functions" by typing "Combinatorial" at the help browser---you will get a list of results that points to tutorials and overvies.

1: The *rule* `a` → $\pi/3$ is *assigned* to the symbol `ARule`. The rule can be read as, "let `a` become $\pi/3$".

2: Note, the rule does *not* make an assignment to the symbol `a`.

3: A rule can be applied with the function `Replace`, but the syntax (.) is typically used instead; one can read `expression/.rule` as "what would `expression` become, if `rule` was applied to it."

4–5: Rules can be collected into lists, and then applied sequentially to an expression.

6: Assignment of `a` will change the form of `ARule`, because if MATHEMATICA® is asked for a symbol it will make any assignments that have been called—in this case, `ARule` will automatically become `SomeOtherSymbol`→ $\pi/3$.

7: Likewise, `AList` will change because it contained the symbol `a`.

8: This is a somewhat advanced example using patterns and delayed rules, which will be explained later, but the point is this: Rules are necessary for manipulations in MATHEMATICA® , but can be used to generate "mistakes." Think of Rule and Replace acting on an expression as "What would the expression be if a certain rule were applied to it?" If the rule is wrong, the resulting expression will be as well.

9: As an exercise, see if you can figure out why this list turned out like this.

10: Besides the help browser, there are ways to get help directly from the FrontEnd. Here, a list of hyperlinks to documentation for functions containing the string "`Exp`" is obtained.

Getting Help on Mathematica

With the implementation of MATHEMATICA® 6.0, the help-browser changed considerably. Many consider it an improvement, but I am not yet convinced—perhaps I am becoming inflexible and resistant to change.

In the old days, one would memorize large portions of the MATHEMATICA® book—which has grown continuously heavier since its first publication in the early 1990's—and rely on the useful "?" and "??" operators. The use of "?" with the wildcard "" enabled a beginning user to track down almost any MATHEMATICA® function. The Options function is also a very efficient way to discover alternate

ways of getting results.

Wolfram decided to stop updating the book as of 6.0, and it is no longer being published. When MATHEMATICA® first came out (I was already familiar with symbolic algebra packages like Macsyma, which predated MATHEMATICA®), I learned it by reading the entire book in one sitting and then I could quickly find and reread parts as I needed it. I found this very effective, and I am curious about what is the most effective way to learn MATHEMATICA® today. I'd be happy to hear suggestions.

In any case, I encourage you to idly explore the MATHEMATICA® Help-Browser. You will not only learn about MATHEMATICA® , but also about mathematics.