Sept. 12 2007

Lecture 4: Introduction to Mathematica III

Simplifying and Picking Apart Expressions, Calculus, Numerical Evaluation

A great advantage of using a symbolic algebra software package like MATHEMATICA® is that it reduces or even eliminates errors that inevitably creep into pencil and paper calculations. However, this advantage does come with a price: what was once a simple task of arranging an expression into a convenient form is something that has to be negotiated with MATHEMATICA® . In fact, there are cases where you cannot even coerce MATHEMATICA® into representing an expression the way that *you* want it.

A MATHEMATICA® session often results in very cumbersome expressions. You can decide to live with them, or use one of MATHEMATICA® 's many simplification algorithms. The "Algebraic Calculations" topics in the Tutorial Overviews section of the Helper Palette provides a nice summary of frequently used simplification algorithms. Another method is to identify patterns and replace them with your own definitions.

MATHEMATICA® has its own internal representation for rational functions (i.e., $\frac{\text{numerator expression}}{\text{denominator expression}}$) and has special operations for dealing with these. Generally, advanced simplification methods usually require a working knowledge of of MATHEMATICA® 's internal representations.

Lecture 04 MATHEMATICA® Example 1

Operations on Polynomials

Download notebooks, pdf(color), pdf(bw), or html from http://pruffle.mit.edu/3.016-2007.

There are built-in simplification operations, such as `Simplify`, but they will not always result in a form that is most useful to the user. Crafting an expression into a pleasing form is an art.

```
PaulENomeal = (1 + 2 a + 3 x + 4 z)^4                                    1
FatPEN = Expand[PaulENomeal]                                             2
Factor[FatPEN]                                                           3
PaulinX = Collect[FatPEN, x]                                             4
Coefficient[PaulinX, x, 2]                                               5
PaulSpiffedUp = Sum[
  Simplify[Coefficient[PaulinX, x, i]] x^i,
  {i, 0, 20}]                                                            6
Simplify[PaulSpiffedUp]                                                  7
RashENell = (x + y)/(x - y) + (x - y)/(y + x)                            8
Apart[RashENell]                                                         9
Together[RashENell]                                                      10
Numerator[Together[RashENell]]                                           11
Simplify[RashENell]                                                      12
Factor[RashENell]                                                        13
```

Simplfiying Expressions with Square Roots

```
RootBoy = Sqrt[(x + y)^2]                                                14
Simplify[RootBoy]                                                        15
Simplify[RootBoy, x ∈ Reals && y ∈ Reals]                                16
Simplify[RootBoy, x ≥ 0 && y ≥ 0]                                        17
Simplify[RootBoy, x < 0 && y < 0]                                        18
RootBoy /. Sqrt[(expr_)^2] → expr                                        19
```

1: We will use this simple expression to demonstrate some of MATHEMATICA® 's algebraic manipulations.

2: `Expand` performs all multiplication and leaves the result as a sum.

3: `Factor` has an algorithm to find common terms in a sum and write the result as a factor and a cofactor—but in this case, it will return the original form.

4: `Collect` will turn in an expression into a polynomial of a user-selected variable.

5: `Coefficient` picks out coefficients of user-specified powers of a variable—this will return the coefficient of x^2 in the polynomial.

6: This is an example of using `Simplify` together with `Coefficient` to simplify only the coefficients of each power of x, and then return the original result by multiplying by the appropriate power and summing.

7: `Simplify` tries to produce a simple result (based on an internal measure of simplicity). Here it returns the same result as `Factor`, but this will not always be the case.

8: Besides polynomials, other frequently encountered forms are *rational forms*—we will use this sum of rationals as an example.

9: `Apart` will re-express a rational form as a sum with simple denominators.

10: `Together` will collect all terms in a sum into a single rational form.

11: `Numerator` returns the numerator of a single rational form.

12–13: In this case, `Simplify` and `Factor` do not produce the same form.

14: MATHEMATICA® is fastidious about simplifying roots and makes no assumptions—unless they are specified— about whether a variable is real, complex, positive, or negative.

15: Many users become frustrated that `Simplify` doesn't do what the user thinks must be correct...

If you think it is obvious that $\sqrt{x^2}$ should always simplify to x, then consider that both $x = \pm 1$ satisfy $\sqrt{x^2} = 1$—picking only $x = 1$ will miss the minus-solution. Or, consider that $\sqrt{x^2} \neq x$ for $x < 0$

16: `Simplify` will accept `Assumptions` as a second argument, or as an option.

17–18: This demonstrates why it is not a good idea to automatically simplify $\sqrt{x^2}$.

19: This is brute force—and not really a good idea.

Lecture 04 MATHEMATICA® Example 2

A Second Look at Calculus: Limits, Derivatives, Integrals

Download notebooks, pdf(color), pdf(bw), or html from http://pruffle.mit.edu/3.016-2007.

Examples of `Limit` and calculus with built-in assumptions

```
AMessyExpression = Log[x Sin[x]] / x^(1/x)                                   1
Limit[AMessyExpression, x → 0]                                               2
DMess = D[AMessyExpression, x]                                               3
Integrate[DMess, x]                                                          4
DefInt1 = Integrate[DMess, {x, 0, e}]                                        5
(AMessyExpression /. x → e) -
 (AMessyExpression /. x → 0)                                                 6
DefInt2 = (AMessyExpression /. x → e) -
   Limit[AMessyExpression, x → 0]                                            7
DefInt1
DefInt2
DefInt1 == DefInt2                                                           8
Integrate[Sin[x] / Sqrt[(x^2 + a^2)], x]                                     9
Integrate[Sin[x] / Sqrt[(x^2 + a^2)],
 x, Assumptions → Re[a^2] > 0]                                               10
UglyInfiniteIntegral =
 Integrate[Sin[x] / Sqrt[(x^2 + a^2)],
   {x, 0, ∞}, Assumptions → Re[a^2] > 0]                                     11
N[UglyInfiniteIntegral /. a → 1]                                             12
Series[AMessyExpression, {x, 0, 4}]                                          13
FitAtZero =
 Series[AMessyExpression, {x, 0, 4}] // Normal                               14
Plot[{AMessyExpression, FitAtZero},
 {x, 0, 3},
 PlotStyle → {{Thickness[0.02], Hue[1]},
   {Thickness[0.01], Hue[0.5]}}]                                             15
```

1–2: This would be a challenging limit to find for many first-year calculus students (*try it!*).

3–4: Here, do a quick verification using differentiation and integration to check if MATHEMATICA® agrees with the fundamental theorem of calculus (`Integrate[ D[expr,x],x]==x`). Note, MATHEMATICA® does not add the arbitrary constant to the indefinite integral.

5: This definite integral *should* the value of `AMessyExpression` at $x = e$, but is not obvious by inspection.

6: Simply evaluating (via application of rules) the integral at the ends of the integration domain does not produce the correct result because of a possible division by zero.

7: Using `Limit` instead of direct evaluation produces the expected result.

8: Although they have different forms (and one can probably see that they are the same expression), testing equality shows that the two different forms of the definite integral are the same.

9-10: Some indefinite integrals do not have closed-form solutions as in **9**, even with extra assumptions as attempted in **10**.

12: But, in some cases even if the indefinite integral does not have a closed-form solution, the definite integral will have one.

13: `Series` is one of the most useful and powerful MATHEMATICA® functions; especially to replace a complicated function with a simpler approximation in the neighborhood of a point.

`Series` returns a `SeriesData`-form which is indicated by the *trailing order* function `O`. Subsequent operations, such as `Simplify`, won't work on a `SeriesData`-form, but `Normal` converts a `SeriesData` to a normal expression by chopping off the `O`.

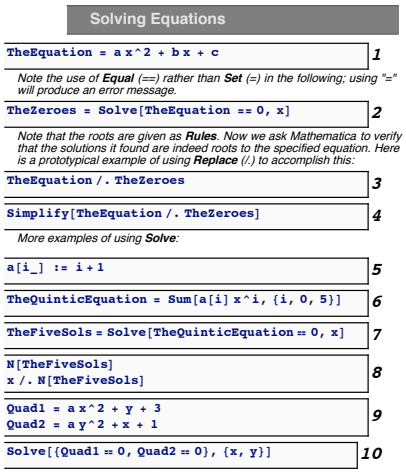
14–15: In this example, `FitAtZero` is a fourth-order approximation to `AMessyExpresssion` at $x = 0$ and has been converted with `Normal` so it can be plotted in **15** alongside the exact expression.

Lecture 04 MATHEMATICA® Example 3

Solving Equations

Download notebooks, pdf(color), pdf(bw), or html from http://pruffle.mit.edu/3.016-2007.

`Solve`, its resulting rules, and how to extract solutions from the rules.


Solving Equations

```
TheEquation = a x^2 + b x + c                                   1
```

*Note the use of **Equal** (==) rather than **Set** (=) in the following; using "=" will produce an error message.*

```
TheZeroes = Solve[TheEquation == 0, x]                          2
```

*Note that the roots are given as **Rules**. Now we ask Mathematica to verify that the solutions it found are indeed roots to the specified equation. Here is a prototypical example of using **Replace** (/.) to accomplish this:*

```
TheEquation /. TheZeroes                                        3
```

```
Simplify[TheEquation /. TheZeroes]                              4
```

*More examples of using **Solve**:*

```
a[i_] := i + 1                                                  5
```

```
TheQuinticEquation = Sum[a[i] x^i, {i, 0, 5}]                   6
```

```
TheFiveSols = Solve[TheQuinticEquation == 0, x]                 7
```

```
N[TheFiveSols]
x /. N[TheFiveSols]                                             8
```

```
Quad1 = a x^2 + y + 3
Quad2 = a y^2 + x + 1                                           9
```

```
Solve[{Quad1 == 0, Quad2 == 0}, {x, y}]                         10
```

1: We assign the familiar quadratic equation to `TheEquation` as a demonstration of how to solve equations and extract solutions.

2: `Solve` takes a *logical equality* (or a list of logical equalities for simultaneous equations) as a first argument. It returns a list of solutions in the form of rules. Here, the list of rules is assigned to `TheZeroes`. There will be one rule for every solution found—if no solutions are found then `Solution` will either return an empty list, or a symbolic list of pure functions that the solutions must satisfy for subsequent use in numerical functions (this case qualifies as an advanced topic). For the general quadratic case, `Solve` returns a list with two rules of the form `{{x->solution1},{x->solution2}}`—*it is a list of lists.* We will see why it is a list of lists when we examine the solution to simultaneous equations in two variables in **9**.

3: To evaluate the original equation at the values of x that solve it, one uses the rules (`TheZeros`) as a list of replacements: `TheEquation/.TheZeros` returns a list of the two values with x replaced by the solutions.

4: Using `Simplify` on this result produces the expected zeroes.

5–6: To see what `Solve` might do with higher-order polynomials, we set up a simple function for the coefficients of a particular quintic equation and create it using `Sum`.

7: The zeroes of a quintic polynomial do not have general closed forms. Here MATHEMATICA® will return a symbolic representation of the solution rules—which we assign to `TheFiveSols`. This representation indicates that the solution doesn't have a closed form, but the form is suitable for subsequent numerical analysis.

8: To extract the numerical solution to `TheQuinticEquation==0`, the first line will return a list of rules for `x`; the second line returns a list of `x` with those rules used as a replacement.

10: This is an example of a solution to coupled quadratic equations. There are four solutions with the form: `{{x->xsol1,y->ysol1},...,{x->xsol4,y->ysol4}}`. Each member of the list must contain a rule for each variable; that is why the solution has the form of a list of a list.

Sometimes, no closed-form solution is possible. MATHEMATICA® will try to give you rules (in perhaps a seemingly strange form) but it really means that you don't have a solution to work with. One usually resorts to a numerical technique when no closed-form solution is possible— MATHEMATICA® has a large number of built-in numerical techniques to help out. A numerical solution is an approximation to the actual answer. Good numerical algorithms can anticipate where numerical errors creep in and accounts for them, but it is always a good idea to check a numerical solution to make sure it approximates the solution to the original equation.

Of course, to get a numerical solution, the equation in question must evaluate to a number. This means if you want to know the numerical approximate solutions $x(b)$ that satisfy $x^6 + 3x^2 + bx = 0$, you have to iterate over values of b and "build up" your function $x(b)$ one b at a time.

The "Numerical Equation Solving" topic in the "Numerical Mathematics" within "Tutorial Overviews" section of the Helper Palette provides a nice summary.

Lecture 04 MATHEMATICA® Example 4

Numerical Algorithms and Solutions

Download notebooks, pdf(color), pdf(bw), or html from http://pruffle.mit.edu/3.016-2007.

Examples of numerical algorithms `NIntegrate` `FindRoot`

Numerical Solutions

```
Integrate[Sin[x] / Sqrt[(x^2 + a^2)], x]                                    1

Integrate[
 Sin[x] / Sqrt[(x^2 + a^2)], {x, 0, 1}]                                     2

NIntegrate[
 (Sin[x] / Sqrt[(x^2 + a^2)]) /. a → 1,
 {x, 0, 2 Pi}]                                                              3

Plot[
 NIntegrate[Sin[x] / Sqrt[(x^2 + a^2)],
  {x, 0, 2 Pi}], {a, 0, 10}, PlotStyle → Thick,
 BaseStyle → {Large, FontFamily → "Helvetica"}]                             4

Plot[{AMessyExpression, FitAtZero}, {x, 0, 3},
 PlotStyle → {{Thickness[0.02], Hue[1]},
   {Thickness[0.01], Hue[0.5]}}]                                            5

NSolve[AMessyExpression == 0, x]                                            6

FindRoot[AMessyExpression == 0, {x, .5, 1.5}]                               7

FindRoot[FitAtZero == 0, {x, .5, 1.5}]                                      8

FindRoot[AMessyExpression == 0, {x, 2.5, 3}]                                9
```

3: `NIntegrate` can find solutions in cases where `Integrate` cannot find a closed-form solution. It is necessary that the integrand should evaluate to a number at all points in the domain of integration (it is possible that the integrand could have singularities at a limited set of isolated points). Thus, a rule and replacement for `a` has to be used for the integrand that appears in **2**. Along with the numerical integrand, the bounds of the definite integral must also be specified. *Like most numerical algorithms,* ***`NIntegrate`*** *can return wrong results (viz* ***`NIntegrate[1/x,{x,1,∞}]`****).* However, in practice these cases are rare; but, be wary.

4: `NIntegrate` is sufficiently fast that we can treat the integrand in **2** as a function of `a`. Here, we let plot vary `a` like the x-axis and plot the results of the numerical integrand from 0 to 2π as a function of `a`.

5: Here we use `Plot` to compare our previous fourth-order polynomial approximation (`FitAtZero`) to the exact result (`AMessyExpression`).

6: `NSolve` will find roots to *polynomial forms*, but not for more general expressions.

7: `FindRoot` will operate on general expressions and find solutions, but additional information is required to inform where to search.

1. You will want to save your work.
2. You will want to modify your old saved work
3. You will want to use your output as input to another program
4. You will want to use the output of another program as input to MATHEMATICA® .

You have probably learned that you can save your MATHEMATICA® notebook with a menu. This is one way to take care of the first two items above. There are more ways to do this and if you want to do something specialized like the last two items, then you will have to make MATHEMATICA® interact with files. Because an operating system has to allow many different kinds of programs to interact with its files, the internal operations to do input/output (I/O) seem somewhat more complicated than they should be. MATHEMATICA® has a few simple ways to do I/O—and it has some more complex ways to do it as well.

It is useful to have a few working examples that you can modify for your purposes. The examples will serve you well about 90% of the time. For the other 10%, one has to take up the task of learning the guts of I/O—hopefully, beginners can ignore the gory bits.

The "Files and Streams" overview within the "Tutorial Overviews" section of the Helper Palette is useful. Data reading is also integrated into MATHEMATICA® —see the "Data Handling & Data Sources" section at the top level of the help browser.

Lecture 04 MATHEMATICA® Example 5

Interacting with the Filesystem

Download notebooks, pdf(color), pdf(bw), or html from http://pruffle.mit.edu/3.016-2007.

Reading and writing data directly and through the use of a *filestream*. A user should check and (sometimes) change the *working directory* to interact with files using `Directory` or `SetDirectory`. Otherwise, the full path to a file must be given.

File Input and Output

```
Directory[]                                                          1
AMessyExpression >> AFile.m                                          2
Clear[AMessyExpression]                                              3
<< Afile.m                                                           4
  The previous statement reads in the expression, but it is not assigned to its previous symbol
AMessyExpression                                                     5
AMessyExpression = << AFile.m                                        6
AMessyExpression                                                     7
FilePrint["Afile.m"]                                                 8
Close["ANewFileName"]                                                9
AFileHandle = OpenWrite[
  "ANewFileName", FormatType → OutputForm]                           10
RandomPairs =
 Table[RandomReal[{0, 1}, 2], {i, 10}]                               11
Write[AFileHandle, RandomPairs]                                      12
FilePrint["ANewFileName"]                                            13
Write[AFileHandle, MatrixForm[RandomPairs]]                          14
FilePrint["ANewFileName"]                                            15
Close[AFileHandle]                                                   16
```

1: `Directory` will print the *current directory* into which, and from which, files will be read (if that directory is writable and readable). To change MATHEMATICA® 's current directory, use `SetDirectory`.

2: Simple redirection of an expression into a file is achieved with `>>` The working directory must be *writable*. Selected symbols can be saved in files all at once using `Save`.

4: A file containing a MATHEMATICA® expression can be read in with `<<` The file must be *readable*.

5: Only the expression was saved using `>>`, not the symbol it was assigned to.

8: The contents of a file can be displayed using `FilePrint`.

10: This opens a filestream for subsequent use. Note that the filestream (here called `AFileHandle`) is associated with a filename (here `ANewFileName`). Filestreams give the user much more control over the way the file is written. The use of filestreams is useful for cases where data is written incrementally during a calculation and this method can be generalized to different kinds of devices. Another use of file streams is when the user wants to have the program compute the file name as a string value, and then use the filestream to write to a file with a meaningful string (e.g., name the file from a computed string "`x=3_y=2.dat`")

11: We use `RandomReal` to create some example data (a list of ten pairs of random numbers) to write to the filestream.

12: An example of writing data directly with a filestream.

13: Subsequent writes to the filestream get appended to the end of the file. Here we write the `MatrixForm` of the data.

16: It is good practice to close open file streams when writing is finished.

Lecture 04 MATHEMATICA® Example 6

Using Packages

Download notebooks, pdf(color), pdf(bw), or html from http://pruffle.mit.edu/3.016-2007.

There are a number of packages that come with MATHEMATICA® (and more that can be bought for special purposes). The packages contain functions and data that can be added to a MATHEMATICA® session as desired, and not loaded beforehand. This helps regulate the amount of memory required to run MATHEMATICA® . You should look through the various packages in the help browser to get an idea of what is there—it is also a good idea to take a look at the inside of a package by editing a package file with an editor. By doing this, you will see some of internal structure of MATHEMATICA® and good examples of professional programming.

Using Packages

Fortunately, others have gone to the trouble of writing files full of useful stuff--and you can load this stuff into Mathematica for your very own use. Some people produce useful stuff and you can buy it, which is nice if you find it valuable--and you can write stuff and gain value by selling it, which might be even more nice.
Mathematica comes with a group of Standard Packages, that you can load in to do special tasks. The Package documentation can be found with the Helper Palette, available at http://puffle.mit.edu/3.016/Help-Pallette-Builder.nb. For example, take a look at the specialized package Calendar:

```
<< Calendar`                                                    1
```

```
DayOfWeek[{1929, 9, 30}]                                        2
```

```
DateList[]                                                      3
```

```
CalendarChange[DateList[], Gregorian, Islamic]                  4
```

```
DateString[CalendarChange[
  DateList[], Gregorian, Islamic]]                              5
```

1: A package is read in using the input operator `<<` or with `Needs`. Here is an example of how `Calendar` is read.

2: `DayOfWeek` is one of the functions available in `Calendar`.

3: `DateList` is part of the standard MATHEMATICA® kernel, without arguments it returns the current date and time.

4: We use the Gregorian calendar–here is the current date in the Islamic calendar.

5: Here, we print the Islamic date in a more readable form. It would be nice to have a little function to translate the day and the month into Arabic...