MATH 330: Computational Analysis

Andrew Hofstrand

Contents

1 Introduction

1.1 Course Description

The major goal of this course is to introduce the student to numerical algorithms, along with their implementation on a computer, based on the mathematical material covered in the NYIT calculus sequence (MATH 170, 180, and 260) and some elementary linear algebra (MATH 310). We do not assume the student has any coding experience, and students who do have some familiarity with data structures, arrays, conditional statements, data input/output, etc. may skip these introductory sections.

Here, we will be using *Mathematica*, due to the simplicity of having symbolic computation, small-scale data manipulation, and visualization capabilities located on a single platform. Interested students may write numerical algorithms in other languages, such as *C* or *Python*.

Even though the student will acquire a significant amount of coding experience during the course of the semester, this is a math course and not a computer science course. Indeed, the major emphasis will be placed on the *conceptual* and *mathematical* underpinnings of algorithms, with the peripheral goal of computational proficiency.

1.2 Numerical Algorithms

The development of numerical algorithms used to solve problems in science, engineering, and mathematics greatly predates the advent of the first electronic digital computers built in the 1940s and 50s. However, the subsequent development and refinement of earlier algorithms, adapted on programmable modern machines, underlies our current computational age. Indeed, almost every aspect of modern life is built on large-scale numerical algorithms: from signal processing in smart devices to predictive weather and climate simulations to medical imaging and detection.

Even though computational analysis plays a vital role in applications, the research area of computational or numerical methods is a separate branch of applied mathematics. There are many well-known journals dedicated only to this area (*Acta Numerica*, *SIAM Journal of Numerical Analysis*, *Journal of Numerical Mathematics*, etc.). Indeed, in many situations the underlying mathematical algorithms remain the same and only the application area (finance, biology, engineering, etc.) and interpretation of results change. Thus, the objective of most texts on numerical methods is independent of any particular application. For the most part, we follow this format here, with a few exceptions discussed later.

1.3 Concepts and Projects

In these notes each section is spit into a *Concept*, which introduces a particular numerical algorithm with a discussion of general mathematical aspects, and

a *Project*, which the student completes to master the material presented in a hands-on manner. The concepts we consider in these notes are commonly used in building more advanced numerical techniques in data analysis, numerical linear algebra, and in the simulation of differential equations (these topics have significant overlap). Students may find more detailed descriptions of the algorithms presented here and their extensions in numerous textbooks and online resources.

2 Symbolic Calculus and Visualization in Mathematica

In this section we give a brief introduction to symbolic manipulation in Mathematica. We also introduce a few built-in Mathematica functions and basic plotting. The concepts in this section will be important for completing the projects later in the course.

2.1 Concept

The following serves as a "bare-bones" introduction to a few important aspects of symbolic computation in Mathematica. There are numerous resources for gaining more proficiency at Mathematica. Importantly, a complete list of functionality is given in the Wolfram documentation under the Help tab and the answer to many specific questions can be found online at the *Stack Exchange*.

2.1.1 Defining a Function

A Mathematica file is called a *notebook* (.nb). The basic building-blocks in a notebook are evaluation *cells*. Opening a new notebook, in a single cell, we can define a mathematical function as follows:

```
f[x_]=x^2-5*x+4;
```

After typing the above statement, by pressing the shortcut keyboard command *Shift+Enter* the cell evaluates (once executed, the f should turn from blue to black). f is now stored in the notebook for future use. Notice that the independent variable, x, is written with an underscore in the function definition (with square-brackets) and the line ends with a semi-colon.

Now we can evaluate the function at whatever point we choose. By typing the following inputs and again pressing Shift+Enter, we get the output

```
f[4]
out[#]=0
```

and

```
f[Pi]//N
out[#]=-1.83836.
```

By not including the semi-colon at the end of the line, we *see* the output. In the second expression above, we note that π is denoted by Pi. In Mathematica, mathematical constants and built-in functions are typically capitalized. Also, to obtain a numerical expression, instead of a symbolic expression, we use double-backslash N at the line end.

Mathematica also understands inputting other symbolic variables into a function. For instance,

```
f[1+y]
```

gives the output: $4 - 5(1 + y) + (1 + y)^2$. This assumes we have not previously defined the variable y in our notebook. If we did, we could write:

```
y=3;
f[1+y]
out[#]=0.
```

To erase a set variable value from memory in Mathematica we can use

```
Clear[y];
```

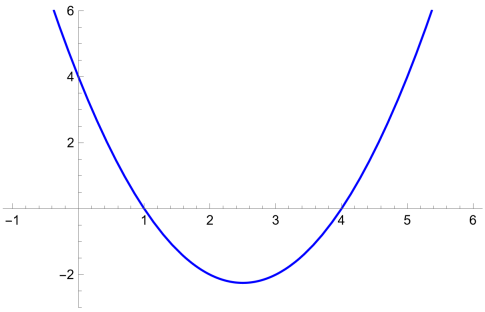
and the variable y turns from black back into blue in the notebook.

2.1.2 Plotting Functions and Data

To plot the above function, we can simply input

```
plot1 = Plot[f[x], {x, -1, 6}, PlotRange -> {-3, 6},
  PlotStyle -> Blue]
```

which gives the output



Notice that we can define a plot like a variable in Mathematica, in this case called plot1. The syntax for the Plot-function is: 1) the function to be plotted; 2) {independent variable, left-most value, right-most value}; (optional) 3) PlotRange $- >$ {display range}; 4) PlotStyle$- >$plot color. There are many more options in Plot[], which you can read about in Wolfram documentation.

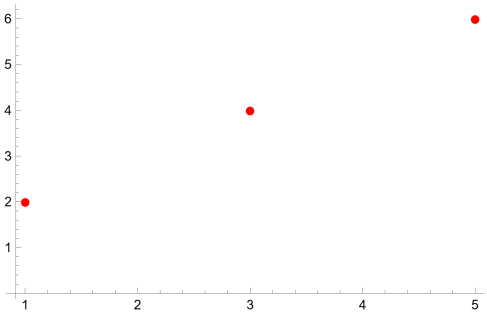
In Mathematica, an *array* or *list* of objects (numbers, strings, plots, etc.) is defined using curly-brackets {}. For example, a set of three data points in $\mathbb{R}^2$ can be stored under the variable name *data* with the input:

```
data={{1,2},{3,4},{5,6}};
```

Another useful visualization tool is ListPlot[], which plots a set of data points, like the one defined above, as a scatter-plot

```
plot2=ListPlot[data, PlotStyle -> Red,
PlotMarkers -> {"\[FilledCircle]", 10}]
```

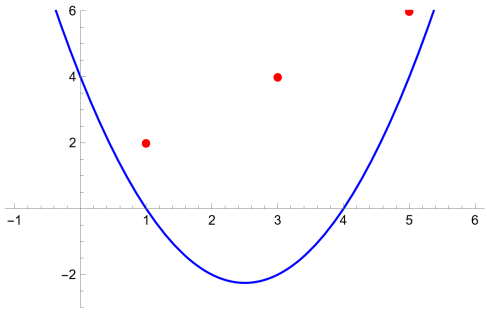
gives the output



Finally, you can show two (or more) plots, like those above stored separately as plot1 and plot2, overlaid on the same graph by inputting

```
Show[plot1, plot2].
```

which gives the result



2.1.3 Symbolic Differentiation and Integration

Mathematica can perform algebra and calculus tasks symbolically. For instance, to take the derivative of the function, f, we defined above, we input and execute

```
g[x_]=D[f[x],x]
```

to output -5+2x. Now the function g stores $\frac{df}{dx}$. We can find higher derivatives, for instance the tenth-derivative of f, by

```
h[x_]=D[f[x],{x,10}]
out[#]=0
```

Similarly, we can symbolically integrate the function f by evaluating

```
r[x_]=Integrate[f[x],x]
```

which gives the output $4x-(5/2)x^2+(1/3)x^3$. We can also symbolically integrate a function over a given interval, say [1,5], as follows

```
Integrate[f[x],{x,1,5}]
out[#]=-8/3
```

Mathematica's symbolic tools are powerful. However, there are many situations where closed-form integrals are not possible or available. One of the largest areas in numerical methods is the approximation of solutions to *differential equations*. In order to effectively solve such problems, simple discrete approximations of both the derivative and integral are required. All problems *cannot* be solved with the same techniques and some problems are more efficiently solved with certain methods. This is what makes numerical analysis such a rich topic.

2.2 Project

2.2.1 Part I: Tangent Lines

- First, plot the function, $f(x) = \ln(x)$, in blue in the interval $-1 \leq x \leq 6$ and $-2 \leq y \leq 3$.
- Now find the tangent lines of $f(x)$, using D[], at the points $x = 1$ and $x = e$. Plot the tangent lines in dashed-red and dashed-gray, respectively, on the same domain and range as above.
- Plot the two points above on $f(x)$ where you evaluated the tangent lines in black.
- Now overlay each of the plots you made above on a single graph. Make sure you can see everything you plotted.

2.2.2 Part II: Integrating a Discontinuous Function

- The so-called *Heaviside* piece-wise function is defined as

$$H(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$$

  Plot the function $G(x) = 2H(x) - 1$.
- Now find the indefinite integral of $G(x)$, using Integrate[] and Piecewise[], and plot the resulting function.
- Integrate the function you found above another time.
- Plot all three functions above, including $G(x)$, on a single graph for $-2 \leq x, y \leq 2$ in different colors.
- What do you notice about the *continuity* and *smoothness* of the indefinite integrals as you keep integrating $G(x)$? Conversely, what happens when you go backwards and differentiate each expression?