Understanding Calculus through Linear Maps: A Computational Approach

Robert D. French

Contents

A Note for Students

Math is fun and exciting! However, many calculus texts are not. I hope this one will be.

Acknowledgements

The author would like to thank Mr. Olivier Verdier for his excellent Python syntax highlighting package [2]. This is a valuable open source contribution, and has certainly enhanced the readibility of this text.

CHAPTER 1

Functional Function Fun

Mathematicians are *really* fond of definitions. Often, mathematicians will spend large parts of their day worrying about the precise definition of some concept, whereas a scientist or an engineer may just take it on intuition. Sometimes in this book, we'll take the route of the engineer, and just go with what makes sense. However, if we want to get started on the right foot, we'll need to make sure that we know exactly what we mean by the word "function".

1. What is a function?

In pre-calculus, you may have discussed a function as an "input-output machine", meaning that it takes one number in, and puts one number out. A slightly better way to put it is to say that a function is a strict rule for associating an input number with an input number. By *strict*, we mean that it is totally unambiguous; there should be no room for guessing about the interpretation of a function. For example,

$$f(x) = \frac{x^2}{6}$$

is a strict rule. For any number x we can think of, it is absolutely clear how to calculate $f(x)$: we multiply x by itself, then divide that number by 6. On the other hand,

$$g(x) = \frac{x^2}{\text{Number of seconds since the author's last Peanut Butter sandwich}}$$

is quite ambiguous; do we count the seconds since the author finished his sandwich, or the number since he began? Further, suppose the author was interrupted from his sandwich by the sudden arrival of an infinite gaggle of Canadian Geese, each requesting an equal-sized crumb. Clearly it would be quite impossible to interpret this expression without opening a hole for philosophers to crawl in and shout at us about the follies of loose reasoning. No, better to wash our hands of it and stick to polynomials for the time being.

2. Polynomials

In this book, we shall limit our discussion of functions to polynomials of real numbers, such as x^2, $2x^3 + 5x^2$, and even $\pi x^3 + 0.001x + 6$. We will not (at first) consider any exponential functions (e^x), nor any trigonometric functions($\sin(x)$, cos(x), etc...), nor any other form of high-brow transcendental lavishness (hyperbolic functions, Bessel functions, respiratory functions...). In general, polynomials look like this:

$$f(x) = a_0x^0 + a_1x^1 + a_2x^2 + \cdots + a_nx^n$$

Where each of the a_i are just whatever real numbers we choose. Every different choice of the a_i's gives us a different polynomial. For example, let's pick two polynomials of degree[1] 1:

$$p_1(x) = 5x^0 + 4x^1$$
$$p_2(x) = 3x^0 + 4x^1$$

These are different polynomials, which should be obvious enough. Given the same input, say $x = 0$, they will give different output values: $p_1(0)$ will equal 5, wheras $p_2(0)$ will equal 14[2]. If we plot these two functions, we will see that while they may overlap in some places, mostly they are different.

Fortunately, we do not have to check polynomials by hand or plot them to see if they are different: polynomials can be entirely characterized by their *coefficients* – that is, the specific values of a_0, a_1, etc. That means we could represent $p_1(x)$ as $(5, 4)$ and $p_2(x)$ as $(3, 4)$, which we can then treat as points in the plane.

In fact, it is a central idea in this book that there is a connection between points in the plane and polynomials of degree 1. The same connection also holds between polynomials of degree 2 (such as $12x^2 - 100x^1 + 17x^0$) and points in a three-dimensional space, like $(12, -100, 17)$. This connection allows us to use a neat tool from geometry, namely *linear maps*, to study properties of polynomials having to do with rates of change. That's actually what Calculus is all about.

3. Functions in Python

The tutorials in this book use the IPython [1] package. Please take a moment to install IPython on your computer so that you can follow along. Install instructions are available online: `http://ipython.org/install.html`.

Let's take a look at a simple polynomial function in Python:

```
def f(x):
  return x + 2
```

This code does nothing more than to declare that `f` is the function $f(x) = x+2$. In the same fashion, if we wanted to define the function $g(x) = x^2 + 3$ in Python, we could do that just as easily:

```
def g(x):
  return x*x + 3
```

[1]The *degree* of a polynomial is the highest number to which x is raised: $x^5 + x^4$ is a fifth-degree polynomial, $x^2 + x + 1$ is a second-degree polynomial. If we're being general, we say that $x^n + x^{n-1} + \cdots + x^1$ is an n-th degree polynomial

[2]Are you working these by hand as we go along? Because $p_2(0)$ is definitely not 14...

Let's talk about what each statement does in a little more detail, so that these code examples become a bit clearer.

Do you remember earlier when we discussed how much mathematicians love definitions? Well, computers are equally obsessed with definitions, so if we want to get anything done in Python, we will need to begin by *defining* what we are talking about. We can introduce a new function definition with the `def` keyword. In fact, the Python expression "`def g(x):`" simply means "define a new function `g` that takes a single input variable `x`".

The lines below the "`def g(x):`" statement are called the *body* of the function, and they tell Python what the function is all about. In this case, we want to calculate the value $x^2 + 3$, so we tell Python to multiply x by itself (`x * x`) and then add 3. The `return` keyword simply tells python "what comes after is the *output* of this function". Once the `return` statement has been called, the function will end.

One more note about the bodies of functions... they must always be indented by 1 tab relative to the `def` statement. This is because Python assumes that the function body extends down until the first line that isn't indented. So, for example, if you were to omit the tab and write the following code:

```
def h(x):
return x*x + 3
```

you would get the following error at runtime[3]:

```
  File "<mycode.py>", line 2
    return x*x + 2
         ^
IndentationError: expected an indented block
```

4. Exercises

Ex. 1 — Follow the instructions given at http://ipython.org/install.html to install IPython on your workstation. If you are using a shared workstation, such as a computer in the library or university commons, ask your professor to harass the campus IT department until they IPython for you (again, using the above instructions). Should the IT department intransigently refuse to assist you, engage in an act of non-violent civil disobedience such as: staging a teach-in in front of the Help Desk, railing against their fascist behavior in pastel sidewalk chalk, or perhaps asking the custodial union to . . . hold off a while on their next IT department trash pickup. Write an essay about your experience.

Ex. 2 — How can you tell Python to import the module `polynomial.py`?

Ex. 3 — Given two polynomials f and g, does it make sense that $f(x) + g(x)$ should always be the same as $(f + g)(x)$? That is, if we evaluate the functions first and then add their sum, will we get a different answer than if we add the polynomials first (i.e. create a new polynomial by adding their coefficients) and then evaluate the new polynomial?

Ex. 4 — Explain why the *bodies* of functions must always be indented.

[3]The phrase *at runtime* is programmer speak for "while your code is running". Using this phrase will make you sound savvy and professional, like you *know what you're on about.*

Ex. 5 — Describe in words what each line of the following Python function does:

```
def distance(x1,y1,x2,y2):
  deltaX = (x1 - x2)
  deltaY = (y1 - y2)
  return pow(pow(deltaX,2) + pow(deltaY,2),0.5)
```

APPENDIX A

Projects

1. Solving the Heat Equation

1.1. Finite Difference Methods.

Bibliography

[1] Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007.

[2] Olivier Verdier. Python highlighting in LaTeX. `https://github.com/olivierverdier/python-latex-highlighting`, June 2012.