Math 109 Calc 1 Lecture 26

Newton's Method

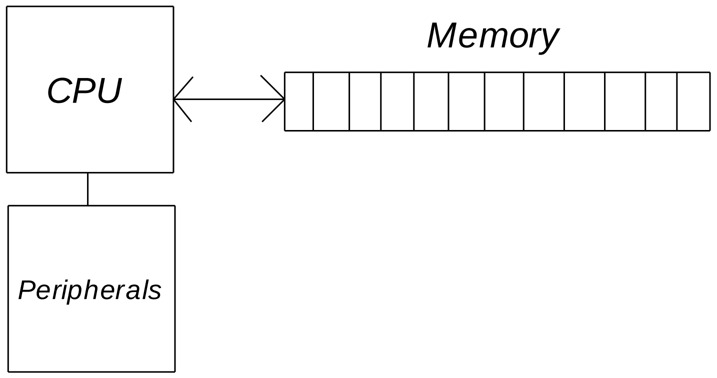
We're going to talk today about a method for finding the roots of an equation called Newton's method or The Newton Raphson method.

It's an iterative method.

If you have a calculator, it could be used. Without one it would be quite tedious, and even useless.

What makes it useful is the existence of computers. For that reason, I'd like to do a quick review of what a computer is and how it works.

Main Hardware Components of a Computer



CPU
The main component is the processor or CPU (Central Processing Unit) or sometimes a core if there are more than one. This is a device made up of electrical circuits.

The CPU's job is to execute instructions. Instructions are just a special form of data that live in the memory. These instructions can move memory around or process it in various ways, such as adding and multiplying it. The instructions can also check things like whether two pieces of memory are the same, or if the represent numbers, which is greater. Based on the results, different instruction paths can be executed. If you were to look at machine code, you would just see a bunch of numbers.

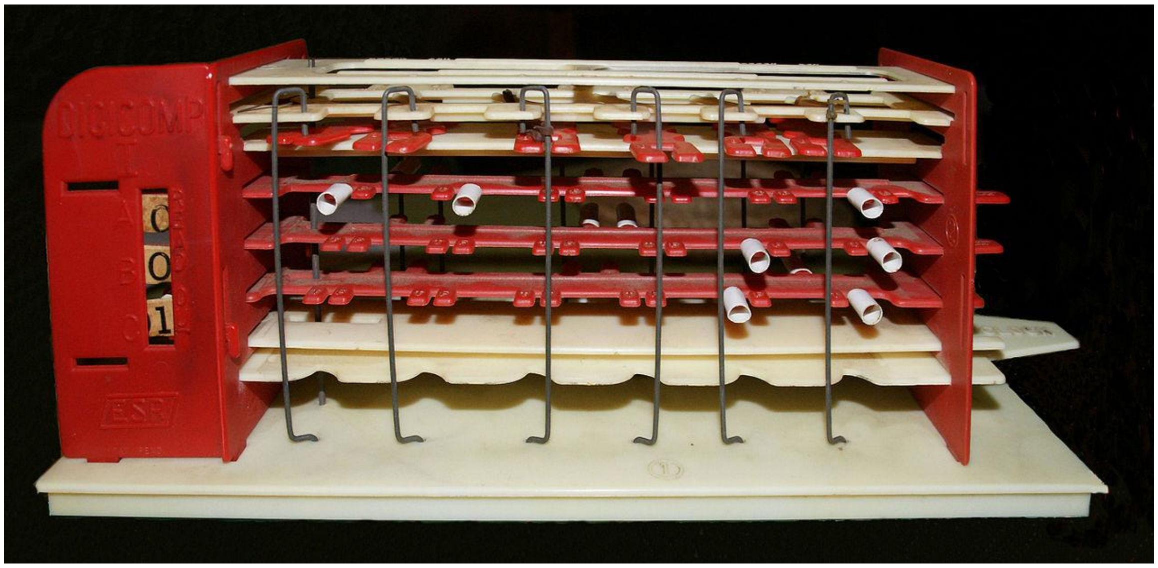
Memory
The memory is also a set of electrical circuits whose job is to hold binary information. Binary data is either a 0 or 1 which can be represented by an electrical voltage. These bits of date are group together in larger groups called bytes, words and double words. The CPU can manipulate memory in different size groups.

Peripherals
Peripherals is a large category of devices that the CPU is connected to that allows it to communicate to the outside world. Examples are
Keyboards
Mice
Monitors
Hard drives

One of the key features of a computer is that the CPU can execute programs, which are groups of instructions. The instructions that a computer executes are in a language unique to that processor. This is called machine code. The most common processor/languages in use today by numbers are x86 x64 and ARM. Within a specific generation of processor these will differ slightly.

Generally speaking, people, who I will call programmers, do not write programs using machine code. Instead they usually write in a higher-level language. These languages are easier for people to understand. Common examples are C++ and Python. There are hundreds of computer languages, although some are rarely used anymore. It's even possible to write your own language.

A Very Simple Computer



This was my first computer. It's barely a computer at all. Instead of electrical circuits it was made of plastic and it worked mechanically. It had only 3 bits of memory which you can see on the left-hand side. To program it you inserted short pieces of plastic straw in either side. The front pieces controlled whether the current data configuration would cause a change. The back pieces controlled what change would occur. So by cycling the computer, the digits could count up, down or do other things.

Execution of Programs

Computers/CPU's do not understand higher level languages. They only understand machine code. There are three main ways that computers execute higher level language programs.

1. Compilation

This could also be called translation. Before a compiled language is executed, it must be translated into the machine code of the CPU that it will be executed on. The translation is usually done by a program called a compiler.

Imagine you have a book written in French and you want to read it, but don't know French. You could hire a translator who would translate it for you and give you a new written copy in English. That's what Compilation is like.

Examples of compiled languages are 'C' and C++.

2. Interpretation
Instead of compiling a program once, the computer can run a program called an interpreter which reads the program and executes each instruction step by step. Scripting languages like Python work this way. Sometimes they do a preliminary scan and convert the language into a nicer format, but not into machine code. A program will typically run slower when interpreted than if it were compiled.

3. Hybrid
Some languages are compiled into a virtual machine code. An example is Java. Once compiled it is then interpreted, however it can run on any machine with a "Java Virtual Machine".

Once thing all computers are good at, regardless of the language they use is do the same thing over and over many times. This brings us back to Newton's method.

Newton's Method

The purpose of Newton's method is to find a root of an equation.

A root is value *a* for which *f* (*a*) = 0.

We start with a guess which we will call a_1 which defines a point on the curve

$$\left(x_1, f(x_1)\right)$$

We find the tangent at this point by using the derivative $f'(x_1)$ and using a linear equation find where the line crosses the *x*-axis.

This gives us a value x_2

This is followed by finding $x_3, x_4, x_5 \ldots$

We can then repeat this process until it converges to a value with sufficient accuracy.

You can see why a computer would be good at this.

It can calculate the function and the derivative quickly and repeat.

The formula for finding values is as follows:

$$x_{n+1} = x_n - \frac{f\left(x_n\right)}{f'\left(x_n\right)}$$

Example:

Starting with $x_1 = 2$, find the third approximation x_3 to the equation
$f(x) = x^3 - 2x - 5$

We have

$$f'(x) = 3x^2 - 2$$

$$x_2 = 2 - \frac{8-4-5}{12-2} = 2 - \frac{-1}{10} = 2.1$$
$$x_3 = 2.1 - \frac{2.1^3 - 4.2 - 5}{3 \cdot 2.1^2 - 2} = 2.0946$$

You can see on the 2nd step we already find a calculator quite handy.

Plugging this value into our equation we see that

$$f\left(2.0946\right) = 2.0946^3 - 2 \cdot 2.0946 - 5 \approx 0.00054$$

A note here, this was a setup problem and we don’t always get a good value so quickly.

Typically, you stop when two successive values are the same to the accuracy you seek.

So, what would this look like in a computer program.

Here is what a programmer would call pseudo code.   It’s not meant to be a specific programming language, but it describes what needs to be done.

```
ACCURACY = 0.000001

a1 = 2
count = 1;

a_cur = a1

LOOP

    a_new = a_cur – calculate_function(a_cur)/calculate_derivative(a_cur)

    if (absolute_value(a_new – a_cur) < ACCURACY)
        break

    a_cur = a_new
     count = count+1

END_LOOP

Print Root is a a_new after count steps

calculate_function(x)
    return = x^3-2*x-5

calculate_derivative(x)
    return = x^2-2
```

So, the crucial steps are checking the accuracy and either stopping or setting the current value to the new value.

Disclaimers

A bad first guess can cause a program to take a very long time to find a root.

In some cases, the algorithm will not converge at all.

So, the program needs to check and see if the estimates are getting better or not.

Programming Example

http://www.schoenbrun.com/usf/math109/php/newton.php

Show some examples using this script