Calculus: Mastering The First Five Objectives

Nicholas Caudill, Andrew F. Rich

Manchester University, North Manchester, Indiana, USA

Abstract

The mastery approach to Calculus requires each prior objective to be mastered before moving on to the next objective.

1. Introduction

The late 1990s saw a big breakthrough in computer algebra systems where affordable and powerful handheld devices came to fruition as seen in calculators such as the Texas Instruments TI-83[3]. Real-world Algebra is hand-written less and less each day. This is mostly due to the developments in computer hardware and software which has created open-source, reliable, and stable computer algebraic systems that any low-income student could afford. This improves the progress of science by giving more people more tools for symbolic mathematics and at a lower barrier of entry than the current convention of TI-83/84 calculators would cost (as of 2021: $80 at Amazon.com). Despite being 20 year old technology, TI calculators can be helpful for problem solving and can be very reliable, but better systems now exist. Are universities across the globe going to adapt to today's technology?

Of course, with these added layers of abstraction, instructors want to make certain their students understand all the underlying working mechanisms for their mathematical operations by testing the students ability to hand-write algebra. This is okay in learning the basic concepts of branches of mathematics such as Calculus; but when applied to real world problems such looking at the rate of change in massive sets of weather data, the majority of that work is automated on a computer algebra system with code optimized for performance.

With computer algebra systems comes these layers of abstraction the software is built upon. Like handwriting algebra, are the underlying mechanics of these systems important for a student to learn? Is it essential for one to understand how the engine of a car is built to be able to drive a care safe and reliably? The following will be an explanation of how these layers are built up from low levels of the machine.

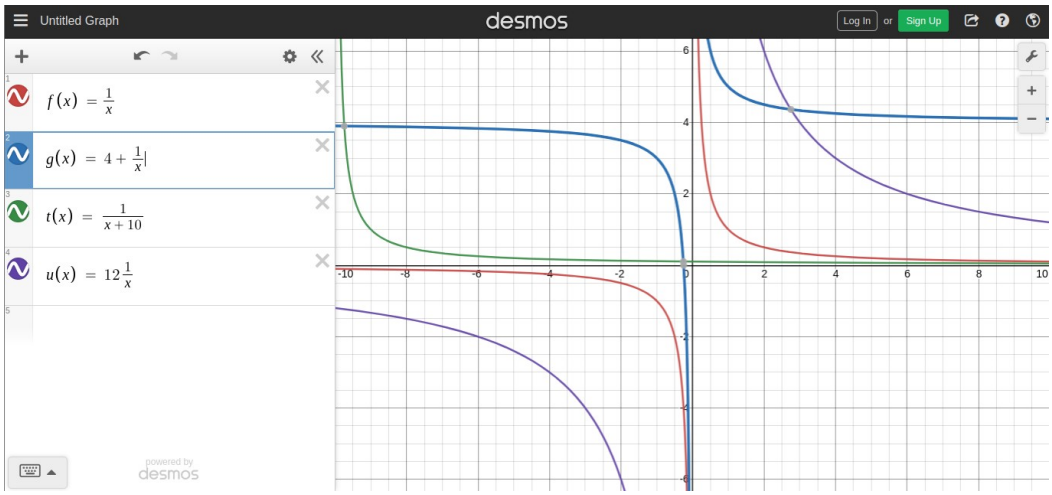
Assembly language talks indirectly to a machine's Random Access Memory through Registers and is a standard of engineering [5]. The C language is then built from this Assembly language [4]. "The C language as well as Assembly are not pieces of software but defined standards. One would not say that they are not open-source, but rather that they are open standards [2]." A higher level language "Python" is then built off of C. "Python is an interpreted, object-oriented, high-level programming

language with dynamic semantics[0].” “All Python releases are Open Source (see https://opensource.org/ for the Open Source Definition)[1].”

This brief software rant brings us to the tool used later in this paper; Sympy: Symbolic Mathematics, which is a Berkeley Software Distribution (BSD) licensed Python library [6,7]. Sympy is built off of other libraries such as Matplotlib and Numpy but is standalone and can do everything in-house using all the contents of its library. In some examples, Jupyter Notebook will be used to display some code written in Python and print them in a pretty way using text codes like LaTex. LaTex printing does not always display correctly when ran through a computer command line interface which is why Jupyter Notebook can be a helpful tool.

1. Functions: definition, modifying, thinking graphically

A function is two sets together with a rule assigning to each element in the first set exactly one element in the second set. This objective includes being able to identity what translations are happening to a graph. In the example below, one should obtain an intuitive understanding for the result of operations inside the (x) term as well as outside the (x) terms. To elaborate, looking at f(x) = 1/x, we should know we could shift the whole function up vertically by adding a constant outside of 1/x. To shift the graph left we would need to do add inside the x term. Note here that the opposite occurs, one might assume the graph for t(x) = 1/(x+10) would shift the graph to the right 10 but it correctly moves the graph to left 10. The more difficult part of this objective might be identifying shrinks and stretches.



2. Limits: idea, graphically and numerically, one-sided, infinite

$$\lim_{x \to a} f(x) = L$$

“Limits give us a way to identify a trend in the values of a function as its input variable approaches a particular value of interest[8].” Since some f(0) might not exist, limits can be useful tools to find arbitrarily close values to f(0) such as f(0.01) or f(0.001). The most important use of limits in calculus is taking the limit of the difference quotient of two points on a curve to find the derivative (see objective 4), also known as the tangent line of a curve. This let one find the instantaneous rate of change for any point on a curve.

Unless specified, limits are taken from both sides and evaluated as true if both limits are the same. To specify which direction to take the limit we write a “+” or” –“ next to “a”.

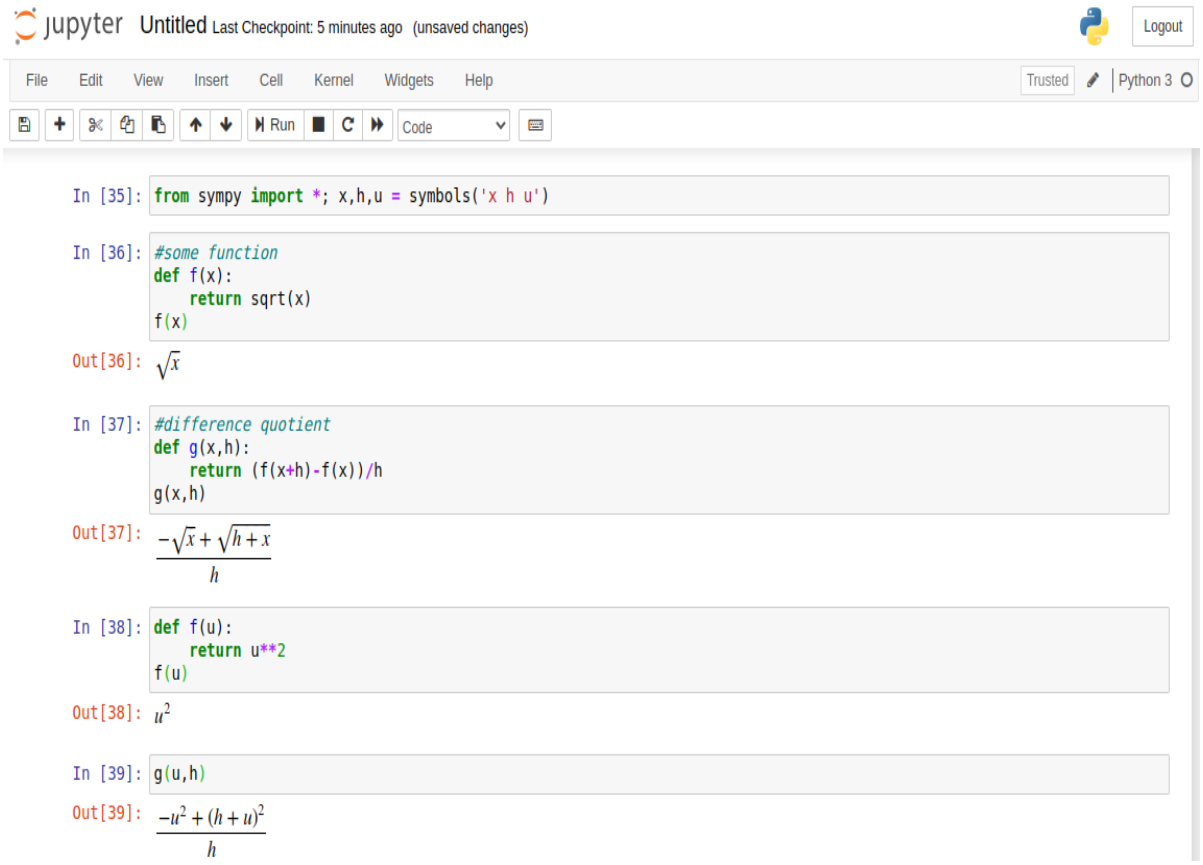
3. Limits: algebraically, including trig functions

Of course anyone could find the limit as x → 9 for f(x) = (2*sqrt(x) - 6)/(x – 9) and Sympy would spit out the limit as being 1/3 using sympy.limit() function. Is the student really mastering the conceptualization of limits when they do this though? Any employer would want their operations built to scale with as little human input as required, but there must be some value in understanding these operations and concepts by handwriting them out. The nuances of what is allowed and what is not is covered in the conclusion.

4. Derivative: definition and meaning, using limit to compute

The derivative of a function is found by taking the limit as h → 0 of the difference quotient. The difference quotient is written as: $\frac{f(x+h)-f(x)}{h}$

Attached below is an image expressing the difference quotient in computer code for a function. Of course we can substitute values in for x or h at any time using the .subs() Sympy function [9].



5. Derivative: numerical and graphical approximations

Finding the derivative f'(x) can be tricky if you are not a wizard at handwritten algebra. It is quite straightforward in Sympy.

In [2]:
```
from sympy import *; x, h = symbols('x h')
```

In [3]:
```
def f(x):
    return sqrt((6+9*x))
f(x)
```

Out[3]: $\sqrt{9x+6}$

In [4]:
```
def g(x,h):
    return ( f(x+h) - f(x) )/ h
g(x,h)
```

Out[4]: $\frac{-\sqrt{9x+6}+\sqrt{9h+9x+6}}{h}$

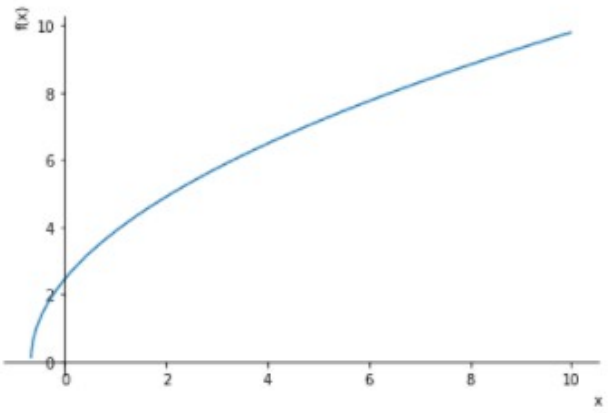
In [5]:
```
deriv = limit(g(x,h),h,0)
deriv
```

Out[5]: $\frac{3\sqrt{3}}{2\sqrt{3x+2}}$

In [6]:
```
print(deriv)
```

```
3*sqrt(3)/(2*sqrt(3*x + 2))
```

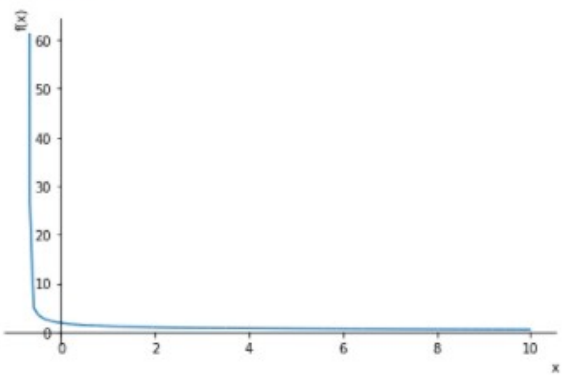
In [7]:
```
plot(f(x))
```



In [9]:
```
#solving the domain with
solve(f(x),x)
# giving us D(f) = [-2,3,infinity)
```

Out[9]: [-2/3]

In [26]:
```
plot(deriv)
```



6. Conclusion

The biggest advantage of the computer algebra system Sympy is that it saves you time by handwriting out equations. You can also plot graphs and find solutions much faster using a full keyboard rather than two thumbs with a Texas Instruments calculator. One could say the human-machine bandwidth of human/Sympy compared to human/TI is faster. Computers have near perfect memory and people often mess up and get lost in the algebra to be reliable computers.

The interesting thing about Calculus is some things are OK to be automated and others not. For example, when finding the derivative of a function with sin(x) or cos(x), that function sin() is condensed into a button on a TI-83 calculator. In a standard college level calculus course, you are not allowed to use Sympy on a laptop while taking a test and are limited to using a TI-83/84; despite Sympy being more readable, more powerful, open-source, and more accessible to low-income students. Even modern TI calculators have Python built into them. Why are we still requiring a ~$100 paywall to do calculus with TI calculators when it can all be done better at next to nothing with Python Sympy.

The underlying logic for the functions such as sin(x)or cos(x) is explained by source [11]. Nobody teaches the logic behind the TI calculator function sin(x) because its long, complex, and uses advanced calculus [12]. Why are calculus homework problems assigning problems that have sin(x) and cos(x) then? Since students are not required to hand-write the logic for sin(x) and thus do not understand the underlying logic of the buttons they are pressing, why are they given a free pass to use the TI calculator sin/cos buttons? To continue, a beginner at calculus is not allowed to do sympy.limit(f(x),x,0,'+') to find the limit of a function as x approaches 0 from the right side.

The sine function is one of two convenient solutions to the differential equation $\ddot{y} = -y$. The other convenient solution is the cosine function.

Specifically, the sine function is the solution with $y(0) = 0, \dot{y}(0) = 1$, and the cosine function is the solution with $y(0) = 1, \dot{y}(0) = 0$.

Using those three equations for the sine function, you can get the Taylor series:

$$\sin x = x - \frac{x^3}{6} + \frac{x^5}{120} - \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!}$$

and similarly for cosine:

$$\cos x = 1 - \frac{x^2}{2} + \frac{x^4}{24} - \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!}$$

If we allow students to use advanced calculus logic they do not understand that is shrunk down to a button on a calculator to aid students to learn about calculus, why stop there? If one automation of logic (sin(x)) is allowed, why not allow students to take their calculus tests with a laptops using Sympy? The calculus with Sympy approach could lead to universities being able to combine calculus I and calculus II courses into one course that is one semester in length. Do students really need to hand write anything to truly understand it? Could this "mastering calculus" situation be similar to the car engine situation where one does not need to understand the intricacies of how an engine is built from the ground up yet the common man is able to use it as a vehicle to travel between destinations.

2. References

https://softwareengineering.stackexchange.com/questions/20988/why-is-python-written-in-c-and-not-in-c

[0] https://www.python.org/doc/essays/blurb/

[1] https://docs.python.org/3/license.html

[2] http://www.open-std.org/jtc1/sc22/wg14/

[3] Campbell, Robert (2001). "TI-82/83/85/86 Mathematics Use". UMBC., https://en.wikipedia.org/wiki/TI-83_series#cite_note-2

[4] https://standards.ieee.org/standard/694-1985.html

[5] https://www.cs.princeton.edu/courses/archive/spr19/cos217/lectures/13_Assembly1.pdf

[6] https://docs.sympy.org/latest/index.html

[7] https://en.wikipedia.org/wiki/BSD_licenses

[8] http://activecalculus.org, ©2012–2019 Matthew Boelkins

[9] https://docs.sympy.org/latest/tutorial/basic_operations.html

[10] https://tutorial.math.lamar.edu/Classes/CalcI/TheLimit.aspx

[11] https://www.quora.com/What-does-sin-x-mean-How-do-we-derive-it-Does-it-have-an-equation-like-sinh-x-What-does-my-calculator-actually-do-when-I-type-in-sin-pi-3-for-example?share=1

[12] https://www.includehelp.com/c-programs/sinx-series.aspx

[13] https://github.com/NSC9/Sample_of_Work

[14] https://www.youtube.com/watch?v=qleGSnrnxgc