

DeAn (Derivative Angles) - Documentation

Bakr Al-rawi

January 2024

Contents

| | |
|---|----------|
| Contents | 1 |
| 1 Introduction | 1 |
| 2 Loading and Preparing for the Program | 1 |
| 3 Using a pre-existing function | 3 |
| 3.1 Introduction to FunctionMode | 3 |
| 3.2 Part 0: Preamble for the code | 3 |
| 3.3 Part 0.5: Settings | 4 |
| 3.4 Part 1: Identifying the Original Function and its Derivative . . . | 4 |
| 3.5 Part 2: Computing the Derivative at an x-value and Constructing the Tangent Line | 5 |
| 3.5.1 Extra: The Mathematical Background | 6 |

1 Introduction

DeAn is a multi-script GNU Octave program that, in essence, deals with calculating the angle between the tangent line of an equation and the x-axis. This file goes over intricacies, design theory, and nuances within the program. Though this program was initially made as a way to learn how to use GNU Octave and not specifically made for applications in physics, this documentation will focus the most on its applications to physics, as well as the parts that most concern themselves with physics.

2 Loading and Preparing for the Program

First, you must have a compiler for your code. The best method to compile GNU Octave is by downloading the application Octave from octave.org, there are also online compilers available on the internet, however, there are often

limitations such as inability to download packages (necessary for this code) or limited time for computations unless a premium feature is purchased. Octave is completely free with all the options you will need for this program. (The rest of this documentation continues using Octave as the program of choice. If you happen to use a different compiler, you will have to determine the parts of your compiler analogous to Octave's as this documentation progresses.)

After you have downloaded Octave, download [DeAn](#). Once you download DeAn, open Octave. In the interface (Figure 1), find the File Browser, then, navigate to the directory where the DeAn folder is and open it. This way, the Command Window can find and run the scripts.

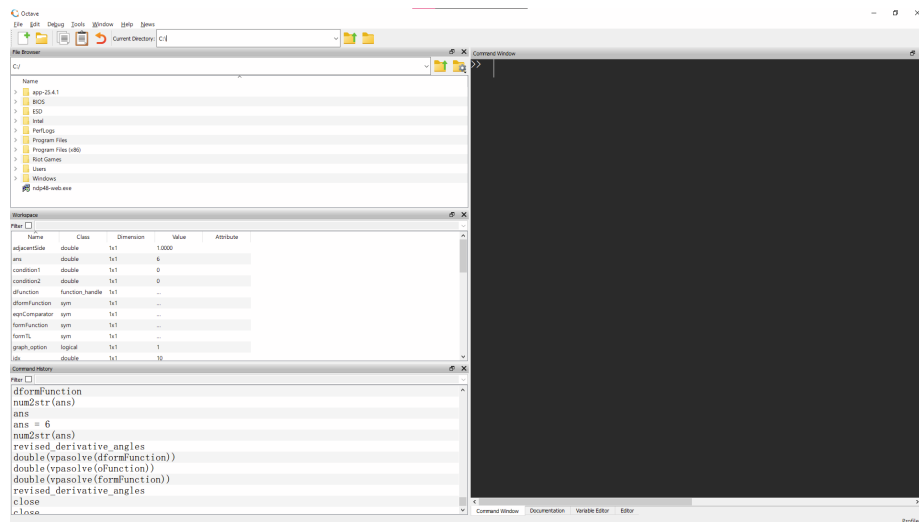


Figure 1: Octave's interface

If you want, you may double-click and open any of the *.m* files on the Editor and see the lines of code yourself (the "Editor" can be accessed under the Command Window). Before the program is run, the packages SYMBOLIC and INTERVAL should be loaded (Figure 2). In the Command Window, type:

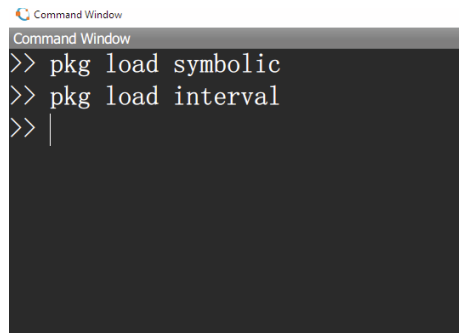
```
pkg load symbolic
pkg load interval
```

And now, we can begin to use our program. Run the program by typing into the Command Window:

```
run DeAnStartup
```

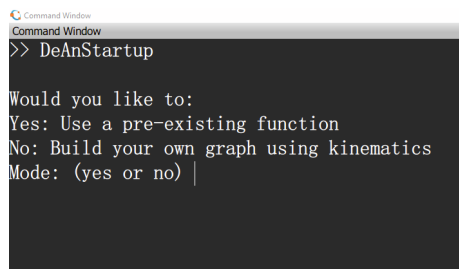
When the program first runs, you will be met with the following prompt (Figure 3):

Let us discuss the first choice.

A screenshot of a 'Command Window' with a dark background and light text. The title bar says 'Command Window'. The text inside shows three lines of input: '>> pkg load symbolic', '>> pkg load interval', and '>> |' with a cursor at the end of the third line.

```
Command Window
>> pkg load symbolic
>> pkg load interval
>> |
```

Figure 2: The Command Window after loading the packages

A screenshot of a 'Command Window' with a dark background and light text. The title bar says 'Command Window'. The text inside shows the command '>> DeAnStartup' followed by a prompt 'Would you like to:' and two options: 'Yes: Use a pre-existing function' and 'No: Build your own graph using kinematics'. The prompt 'Mode: (yes or no) |' is at the bottom with a cursor.

```
Command Window
>> DeAnStartup

Would you like to:
Yes: Use a pre-existing function
No: Build your own graph using kinematics
Mode: (yes or no) |
```

Figure 3: The startup part of the program

3 Using a pre-existing function

3.1 Introduction to FunctionMode

When this program was created, it was intended specifically to be used for this task: to insert a specific function that can be used to make a tangent line out of. `FUNCTIONMODE` is the name of this mode, and can be run by typing "yes" into the startup prompt.

3.2 Part 0: Preamble for the code

This part of the code sets up necessary stuff for the code to function.

```
1 pkg load symbolic
2 pkg load interval
3 close
4 close
5 fflush(stdout);
6 syms x;
7 x1 = -10000:0.1:10000;
```

Lines 1 and 2 are for loading the package (despite it being a part of the code, you still had to load it from the beginning). We will get to what lines 3 and 4 are about as we discuss figures. Line 5 uses the `fflush()` function—this is

standard "good practice" for code to make sure that all output makes it out before a new event occurs. In here, the event would be an input function (which we will see in a bit). Line 6 identifies `x` as a variable symbol. As to not go off on a tangent, just know that **sym** is a class of data (like **double**) and is used by the functions provided with the SYMBOLIC package for future calculations and inputs. Finally, line 7 is a vector variable that provides a set of defined points for the x-axis (200,001 points!). This will be the defined space for the x-axis, meaning if a tangent line's x-intercept exceeds 10000 in the magnitude of its value, the figure will face errors.

3.3 Part 0.5: Settings

This part of the code provides options for the user.

```
1 display("Would you like to have a graph to display the tangent
   point?")
2 graph_option = yes_or_no("Plot the graphs?: ");
3 value_option = yes_or_no("Would you like angle values to be
   displayed in degrees? (radians by default): ");
4 display("Enter your function!", "Make sure the independent value is
   expressed as 'x'")
5 display("Also, make sure that any coefficients be separated from
   their term with an * symbol (unnecessary for a - sign)")
```

Line 2 asks whether the user wants to plot the graph (Line 1 is a display code to further contextualize Line 2). Line 3 asks whether values be displayed in degrees or radians (radians mode does not show the value as multiples of Pi and will simply be a close rational approximation). Lines 4 and 5 are display code providing context for our next part...

3.4 Part 1: Identifying the Original Function and its Derivative

Part 1 consists of the following code:

```
1 str1F = input("Your Function: ", "s");
2 str1F = strrep(str1F,"e^x","exp(x)")
3 symb1F = sym(str1F);
4 oFunction = function_handle(symb1F);
5 formFunction = formula(oFunction(x));
6 symbDF = diff(formFunction);
7 dFunction = function_handle(symbDF);
8 disp("The derivative has been calculated!")
```

Line 1 initializes the variable **str1F**, which stores our function in the form of a string provided by the **input()** function. Line 2 is a tiny consideration for the function e^x , since in GNU Octave, Euler's number e is not **e** but is rather **exp()**, where between the parenthesis lies the power. Though this consideration can only deal with e^x (as the **strrep()**, string replace function, is being asked to replace the text **e^x**, and no other forms of the exponential function), it is placed for the sake of convenience. Line 3 converts the value of **str1F** into a **sym**

class and stores it in `symb1F`. Line 4 creates a function handle for our inputted function. Function handles are an extremely useful tool, and we will re-call this function handle `oFunction` when we get to plotting our graphs (provided the user requested them) later. Line 5 stores the function into a `formula` type, which is useful for Line 6, to take the derivative. Finally, Line 8 is a display code to let the user know that "The derivative has been calculated!".

3.5 Part 2: Computing the Derivative at an x-value and Constructing the Tangent Line

For Part 2 of the code, the following is shown:

```

1 dformFunction = formula(dFunction(x));
2 disp("What is the x-value for which you want to evaluate the
   derivative at? (For initial velocity, you can type 'vo')")
3 xo = input("The x-Value: ", "s");
4 double(vpasolve(formFunction, x, -10000));
5 xo = strrep(xo,"vo",num2str(ans))
6 subs(dformFunction, x, xo);
7 slope = vpa(ans);
8 xo = vpa(xo);
9 subs(formFunction, x, xo);
10 yo = vpa(ans);
11 formTL = formula(slope * x - slope * xo + yo);
12 tLine = function_handle(formula(slope * x - slope * xo + yo));
13 idx = index(disp(formTL), "x")
14 condition1 = 0;
15 condition2 = 0;
16 eqnComparator = formFunction == formTL;
```

Let's break it down.

Line 1 converts the `dFunction` into a `formula`, so as to solve for the `slope`. Line 2 is a display code. Line 3 asks the user for the x-value at which the derivative should be evaluated. This line, alongside Line 1, help in Line 7, as the `subs` function uses the `dformFunction` and `xo` from to solve the derivative at a specified x-value.

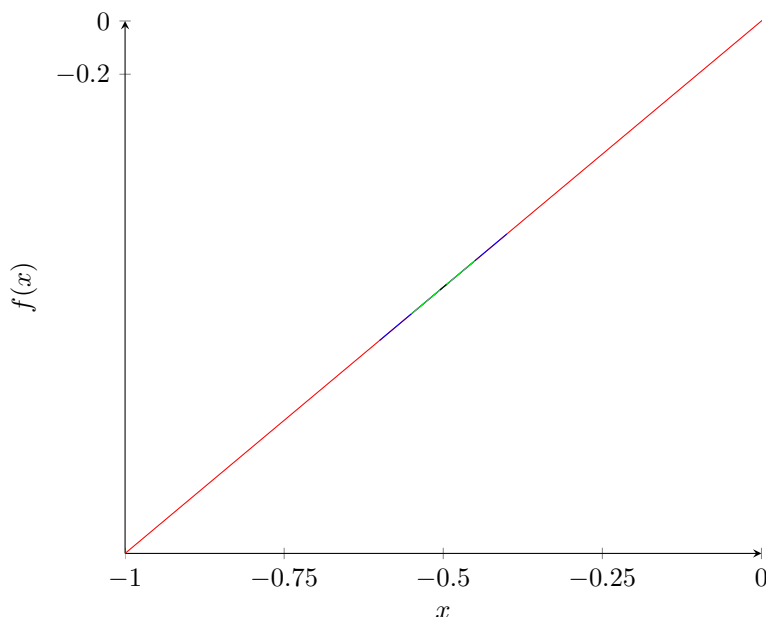
You may notice that the display code for Line 2 adds that there is an option to calculate the initial velocity, the value of x at which, assuming the graph is for some sort of motion, motion starts. In mathematical terms, this would be the x-intercept. Line 4 does this by using `vpasolve`, which helps find the x-intercept of a function. `vpasolve`'s parameters ask to provide the function, the value of a variable for which to solve $y = 0$ (where y is the output variable), and the third parameter is the guessing value. In the case that a graph has more than two solutions, like graphs of parabolic trajectories, the guessing value helps pick the solution closest to it. Since this program works for the interval $[-1000, 10000]$, setting it at -10000 makes sure that it's most likely that the user receives the value of the initial velocity, as the initial velocity would be closest to the leftmost point in the domain (set of x-values).

3.5.1 Extra: The Mathematical Background

(You may skip this subsection if you are familiar with the basics of derivatives and tangent lines.)

The derivative is, simply put, a function that displays the rate of change between two infinitesimally close points. It is like the slope of a function, but with the two points used being extremely, extremely close to each other. When the function is linear, the derivative is always a constant value. This is because in a linear equation, there are no changes in the way that values change: in other words, every point that comes after another, arose in the same fashion as any other previous time in which a point came after another:

An example of a linear function $f(x) = 2x$



Evidently, we can see that the graph provides new values in the exact same way each time, which defines what a linear equation is: the "line" shape comes from having a uniform change for the entirety of its domain's outputs (its x-values' y-values). The highlights in different colors show that no matter how much we "zoom in" (from the red line, to the smaller purple, to the green, and the tiniest being black), the behavior is the exact same: a constant behavior in how the values change as we move forward on the x-axis. The derivative of $2x$:

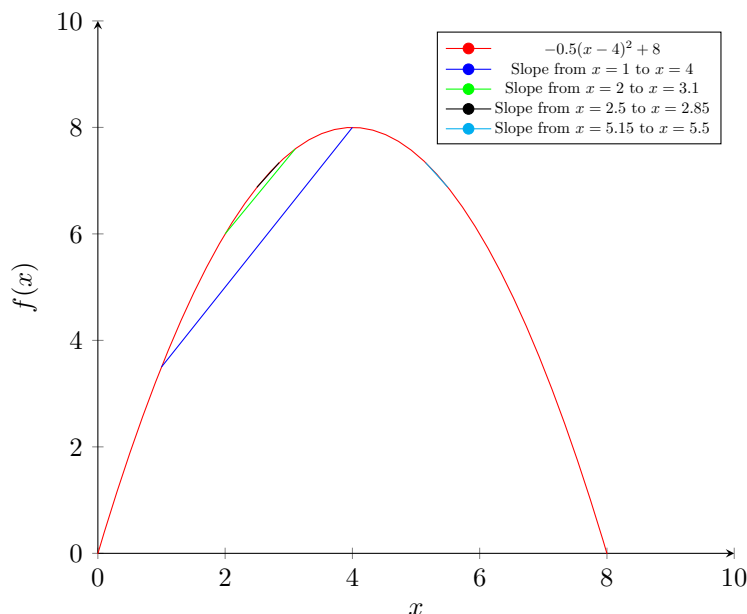
$$\frac{df}{dx}(2x) = 2, \text{ per the power rule}$$

If you are not familiar with what exactly is the "power rule" (or how to find a derivative), do not worry, as it is not necessary to know here. Just know that it *makes sense* for this derivative to be a constant value, losing the x variable as there is, literally, no *variety* in the derivative: the infinitesimally small change

in y with respect to an infinitesimally small step in x is always 2.

Things, however, change when considering non-linear functions. In such functions, the rate at which a step in the x -value changes the y -value differs based on a non-linear fashion:

An example of a non-linear function $f(x) = -0.5(x - 4)^2 + 8$



In here, we see that our graph not only changes from increasing to decreasing, but the "way" that the graph increases/decreases gradually changes, from a steep increase, to a nearly-flat increase, plateauing at $x = 4$, then a nearly-flat decrease, then a steep decrease. Thus, we can understand that the derivative of this function will likely not be a constant value but instead a function that, when given a specific x -value, gives a description of a slope (rate of change between x and y) that, for a non-linear graph, is not found *everywhere*. The derivative of this function is:

$$\frac{df}{dx}(-0.5(x - 4)^2 + 8) = \frac{df}{dx}(-0.5x^2 + 4x + 16) = -x + 4$$

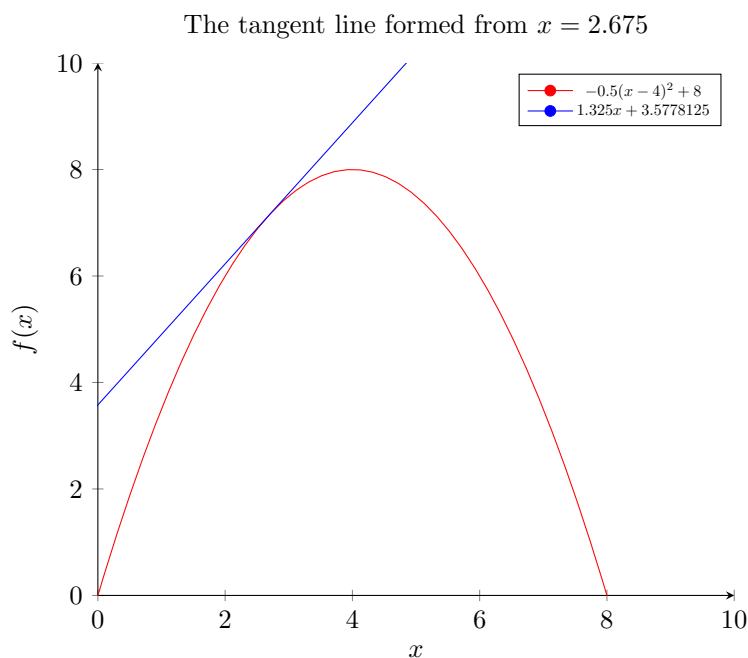
What this means is that the derivative itself is also not a constant, just like the original function. This means that the rate of change in the y -value between two infinitesimally close x -values is not the same at each point, like the linear function previously shown. So, imagine a slope that is taken between two points within the black and cyan lines that are much, much closer than the two points with whom the slope was taken for the black and cyan lines. Indeed, this line would show that the black and cyan lines are somewhat of an approximation of the derivative of the point in the middle between their respective pair of selected x -values: as in, the black line's slope that is between $x = 2.5$ and $x = 2.85$, for example, is a good approximation of the exact derivative of $x = 2.675$, the x -

value in the middle of $x = 2.5$ and $x = 2.85$. The derivative of $-0.5(x - 4)^2 + 8$, where $x = 2.675$, is:

$$\frac{d}{dx}(-0.5(x - 4)^2 + 8) = (-x + 4)$$

$$\xrightarrow{\text{plug in the specific x-value}} (-x + 4) \Big|_{x=2.675} = -2.675 + 4 = 1.325$$

Thus, the "slope" between y and x at *exactly* that point is 1.325. This is then a constant value, and this makes sense because the variability of the derivative was fixated by substituting the one variable, x , that was in the equation. There are many applications for what a derivative can do; the application this program focuses on is constructing the tangent line. The tangent line is a linear equation whose slope is the derivative of x at some point in the original function. For example, here is the tangent line for $x = 2.675$:



As you can see, the tangent line and the original function intersect at that point in which we took the derivative. You will also notice that, as previously stated, the slope is the same as that of the derivative of $f(x)$ at $x = 2.675$. But graphing the constant value 2.675 alone does not grant the tangent line but simply a flat line that does not exhibit the same behavior. The way we construct a tangent line is using the following equation:

$$y - y_o = m(x - x_o)$$

Where:

m is the slope

x_o is the x value at which we took the derivative

y_o is the y value of the original function at x_o

The value of the original function at $x = 2.675$ is:

$$-0.5(x - 4)^2 + 8$$

$$-0.5(2.675 - 4)^2 + 8 = 7.1221875$$

$$\text{so, } 2.675 = x_o \text{ and } 7.1221875 = y_o$$

We already found m through our slope calculation, therefore:

$$y - 7.1221875 = 1.325(x - 2.675)$$

$$y = 1.325x - 3.544375 + 7.1221875$$

$$y = 1.325x - 3.5778125$$

And thus, that is what a tangent line is. But thankfully, all that math is done in a couple of seconds or less by the computer and the code.