Engineering Calculus 1
Mini Project 1: Coding Alternative

Lina Fajardo Gómez

September 2022

This project will help you explore what we have learned about functions, limits and derivatives in a hands-on application. Most of the code is done for you, and it is heavily commented in some places to talk you through what each part is doing. Your task will be to interpret what the code is doing, connect it to what you have learned about calculus so far, and draw some conclusions.

Before We Begin...

You should have read through the tutorials to understand

- how objects are created in `Python`,
- data structures such as lists,
- functions, and
- loops.

You have a week to complete this project. Do not wait until the last day. The sooner you start, the more opportunities you will have to ask for help if you need it.

Use an online interpreter like trinket.io/python3 if you don't want to install `Python` in your computer. You can copy and paste code from this document as needed. As you work through the file, you will be asked to show outputs from your code, type some of your own code, and write out formulas.

When plain text or code are required, text boxes will be provided for you. For formulas, empty spaces are left. You can use a form editor to copy and paste your outputs and code snippets into the text boxes. Typesetting formulas over the white space may not be easy, so you can print the document after filling all text boxes and hand-write anything that needs mathematical expressions.

1 Functions and Limits

Remember functions as rules that pair inputs to outputs. This one is takes one argument, `input_val` (which can be any number) and outputs its square. Note that integers (`int` type) can't have decimals, while `float` type numbers can.

```python
# This function can be modified to represent different functions.
def fun(input_val):
    return input_val**2
```

Important to note here is that the way to type x^2 in `Python` is `x**2`. For the most part, all other operations work the way you would expect them to. You just have to remember that computers will do what you *tell* them to do, not necessarily what you *want* them to do: when you write by hand $2(3)$ you understand the parentheses to mean multiplication. When you want `Python` to multiply the numbers, though, you need to add the `*` symbol between them: `2*3`. If you try to type `2(3)`, `Python` will think you are treating the number `2` as a function with input argument `3` and will point out that integer objects cannot be used that way.

In class, we talked about limits as the mental math of evaluating a function "close to" a particular value. In the following function, `L` is a `list` type object and `v` is a `float` type number. Read the code carefully to answer the questions below.

```python
def eval_fun_left(v, L):
    # Initialize the output of the function as an empty list
    ret = list()
    for elem in L:
        # Fill the output list with the result of applying a function to a
        # number that approaches v from the left
        ret.append(fun(v - elem))
    return ret
```

1. Create a list `L` of five progressively smaller numbers and choose a value `v`. Type them out here, remembering to use the correct syntax for the creation of a list:

   `L =`

   `v =`

   Use the `print` command to show the results obtained from calling the function `eval_fun_left(v,L)`.

   Output:

   Write down a mathematical expression to represent the trend in values of your output list.

   Insert some code of your own corresponding to `eval_fun_right`.

Write down a mathematical expression for the trend in values in the output of `eval_fun_right`.

2. Type out a `Python` expression for the function

$$f(x) = \frac{5x^2 - 4x + 3}{2x^2 + 1}$$

using `x` as an input argument.

f(`x`) =

How would you evaluate $\lim_{x\to\infty} f(x)$ with the `Python` functions above? What would you have to change in the code? Explain.

After making all appropriate changes to the code, print out a list of output values and paste them here.

Output:

Compute the limit algebraically to corroborate your results.

2 Continuity

We can call on `Python` libraries to have access to more tools. Here, we'll make use of the `numpy` and `matplotlib` libraries. Because they can be cumbersome to type out completely every time, we often abbreviate them for easier reference. For example, we can call on `numpy` to access the factorial function ($n! = n(n-1)(n-1)\cdots 3\cdot 2\cdot 1$, read as "$n$ factorial") as follows

```
import numpy as np
print(np.math.factorial(5))
```

When we need to use it multiple times, this is shorter than typing

```
import numpy
print(numpy.math.factorial(5))
```

After importing it, we can use the `numpy` library to create a series of sample points that are equally spaced between a start and end value as follows:

```
# The numpy module can be used to create a set of evenly spaced points
# between a start and end value. Here, 100 points between -10 and 10
import numpy as np
sample_points = np.linspace(-10, 10, num =100)
```

The `matplotlib` library will allow us to plot functions at a series of sample points. For example continuing from the code above we can import the library, evaluate the function at our sample points, and then plot it.

(continues from the code above)

```
import matplotlib.pyplot as plt
curve_points = list()
for p in sample_points:
    curve_points.append(fun(p))

# Plotting the function
plt.plot(sample_points, curve_points, label = "function")
# Set the x and y axis labels.
plt.xlabel("x - axis")
plt.ylabel("y - axis")
# Set a title of the current figure.
plt.title("Graph of the function")
# Show a legend on the plot
plt.legend()
# Get the current axes
ax = plt.gca()
# Adjust the viewing window by setting limits for x and y values
ax.set_xlim([-10, 10])
ax.set_ylim([-5, 5])
# Display a figure.
plt.show()
```

Warning: If you are using `trinket`, it may be the case that you need to click run more than once to generate plots. Sometimes it says there is an error, but it should work if you just try again.

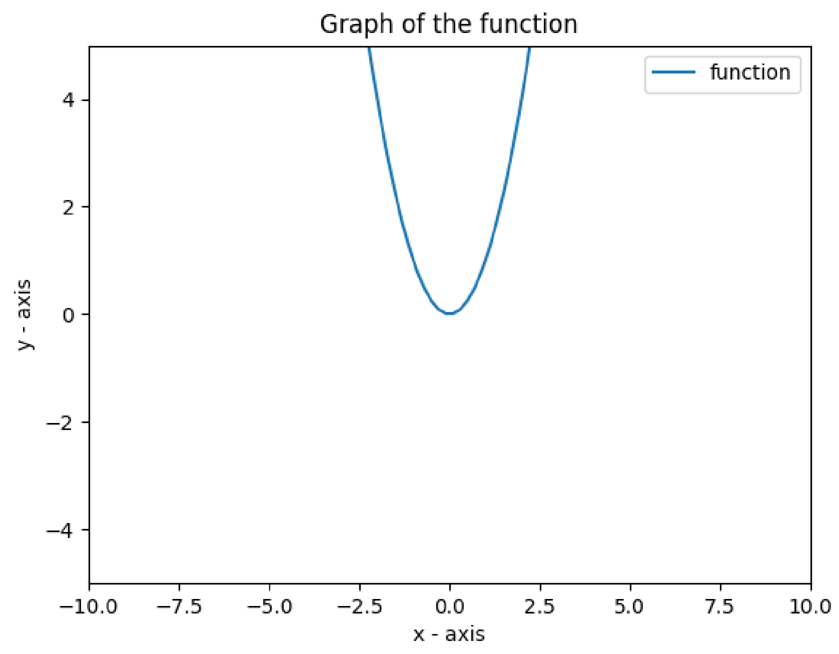

Figure 1: Figure of the function $f(x) = x^2$ created by `matplotlib`.

3. Write down the definition of continuity of a function f at a point c.

Read the following code carefully and run it. Analyze the plot produced.

```python
import numpy as np
import matplotlib.pyplot as plt

def fun(input_val):
    return 1/(input_val-2) + 5

sample_points = np.linspace(-10, 10, num =40)

curve_points = list()
for p in sample_points:
    curve_points.append(fun(p))
plt.plot(sample_points, curve_points, label = "function")
plt.xlabel("x - axis")
plt.ylabel("y - axis")
plt.title("Graph of the function")
plt.legend()
```

```
ax = plt.gca()
ax.set_xlim([-10, 10])
ax.set_ylim([-1, 9])

plt.show()
```

Write down a formula for the function that was plotted:

Find the function’s intercepts and evaluate any appropriate limits to sketch it below. Show all your work.

Describe in detail all inaccuracies you see in the plot produced by the `Python` code. Use key words related to what we have learned in class. Why did the plot come out that way? Explain based on what the code is doing.

4. Modify some of the code used so far to plot the function in a viewing window around the origin

$$f(x) = \frac{\sin(3x)}{\sin(2x)} x^2$$

Paste it in an image below:

Use your plot to visually estimate the limit

$$\lim_{x \to 0} f(x)$$

Calculate the limit by hand and compare your answers. **Make sure to show all work and cite any known limits or theorems you use.**

3 Derivatives

We can modify a number object by redefining it. For example,

```
num = 6
num = num + 5
```

initializes the object **num** with the value 6 and then adds 5 to it, so that next time **num** is called it has the value 11. Another way of writing this is

```
num = 6
num += 5
```

We can also modify lists by redefining their elements. This code initializes a list `L` with four elements and then modifies the first one (note that in `Python` counting begins with the number zero).

```
L  = [1,2,3,4]
L[0]+= 3
```

so that next time we pring `L` we see `[4, 2, 3, 4]`.

We can use something similar to set notation to select items from a list. For example,

```
four = [x for x in L if x == 4]
```

is the list of elements of `L` that have value equal to 4, which would print as `[4, 4]`. Here, the notation `==` is used for a logical test (is this equal?) and not for an assignment (this is equal). We could also define `less = [x for x in L if x < 4]` which would print as `[2, 3]`.

We can also use these logical tests to find the positions in a list where we can find particular values. The code

```
L = [-3, 1, -2, 0, 4, -5]
print(L.index(0))
```

outputs 3 to indicate that the value 0 is stored in list `L` in position 3 (which is really the fourth entry).

5. Write down the definition of the derivative of f at a point c.

   Analyze the code below and explain how it can be used to plot the derivative of a function defined by **fun**.

```
def approx_derivative(fun, h_value):
    derivative_points = list()
    for p in sample_points:
        derivative = (fun(p + h_value) - fun(p))/h_value
        derivative_points.append(derivative)
    return derivative_points
```

One way to add optional arguments to a function is to assign them in the definition. Here, the optional argument `max_n` has default value 5 but can be changed to anything else.

(continues from the code above)

```python
# This function defines an alternating polynomial sum up with up to
# max_n odd degree terms. Here we default to let max_n = 5
def alt_poly_sum(input_v, max_n = 5):
    ret = 0
    for i in range(max_n+1):
        # To use the factorial function we need to call numpy
        # Here we update our output by adding terms to it
        ret += (-1)**i*input_v**(2*i+1)/(np.math.factorial(2*i+1))
    return ret
```

Typing

`alt_poly_sum(3.1, max_n=4)`

evaluates the function at 3.1 using `max_n = 4`, which changes how many terms are added in the `for` loop.

Write a mathematical expression representing what the code does for different values of `max_n`. Use x to represent your input value. (*Hint: these should all be polynomials.*)

`max_n` = 1: $p(x) =$

`max_n` = 3: $p(x) =$

`max_n` = 5: $p(x) =$

Use the following code (added to the above) to plot the polynomial $p(x)$ when `max_n` = 5 along with its approximate derivative.

(continues from the code above)

```
import numpy as np
import matplotlib.pyplot as plt

sample_points = np.linspace(-10, 10, num =100)

curve_points = list()
for p in sample_points:
    curve_points.append(alt_poly_sum(p))

plt.plot(sample_points, curve_points, label="alternating polynomial")

plt.xlabel("x - axis")
plt.ylabel("y - axis")
plt.title("Graph of the alternating polynomial")
plt.plot(sample_points, approx_derivative(alt_poly_sum, 0.0001),
         label = "approximate derivative")
plt.legend()
ax = plt.gca()
ax.set_xlim([-10, 10])
ax.set_ylim([-5, 5])
plt.show()
```

Having graphed the polynomial, we can try to approximate its zeros using the Intermediate Value Theorem as follows. First, we find the points with positive and negative y values:

(continues from the code above)

```
# We can try to find some of the zeros of the alternating polynomial
# by finding some of the plotted points with positive and negative
# y values
positives = [x for x in curve_points if x > 0]
negatives = [x for x in curve_points if x < 0]
```

Then, we find the indices where these values are in the list of x values:

(continues from the code above)

```
# To find the x values where these positive and negative y values
# occur, we use the index method for lists.
# If idx is in positive_indices, sample_points(idx) is an x value
# that resuts in a positive y value.
positive_indices = [curve_points.index(x) for x in positives]
negative_indices = [curve_points.index(x) for x in negatives]
```

Examine the lists of indices carefully to select an x value in the interval [2,4] where the alternating polynomial function is approximately zero. Type your chosen value here and then evaluate it.

x =

$p(x)$ =

How did you choose your x value? What properties or theorems are you using? Why should $f(x)$ be approximately zero at that point?

Change the viewing window to $-4 \leq x \leq 4$ and $-1.5 \leq y \leq 1.5$ by adjusting `xlim` and `ylim`. The function $g(x)$ should very closely resemble an essential function within these values. Which function is that?

$g(x)$ =

What is an x-intercept of that function in the interval [2,4]?

x =

⋆ Create more alternating polynomials by letting `max_n` get larger and larger against the function $g(x)$ you found in the last problem. Plot three of them where `max_n` > 5 in the same graph, adjusting the viewing window as necessary.

If $p_n(x)$ is the alternating polynomial created by the code when `max_n` = n, for a fixed natural number n, then make a prediction for the function approximated by

$\lim_{n\to\infty} p_n(x) =$