Engineering Calculus II
Mini Project 4: Computing work left to computers

Lina Fajardo Gómez

April 2023

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...

This project assumes you're familiar with `Python` as a programming language. If that is incorrect, you should at least review

- how objects are created in `Python`,
- data structures such as lists and dictionaries,
- 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. The USF Application Gateway also has it, though I'm not even linking to it because I can't guarantee it will work properly. Source files will be available on Canvas for you to copy and paste from. The .pdf render doesn't respect spacing and indentation (which `Python` very much cares about), so it's better not to copy and paste from it. 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.

One way to get an appreciation for the hard work that went into making a useful module like `numpy` or `sympy` open source is attempting to recreate even a small fraction of its functionality. We will recreate, more or less from scratch, many of the functions of a scientific calculator. This means, whatever you do, don't fall for the temptation to use ready-made modules where the functions we want already exist.

1 Polynomials

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.

```
# This function can be modified to evaluate 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. 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. That said, the function described above can easily enough be modified to evaluate any polynomial. Now, the beauty of Taylor series is that we won't need much more than polynomials for anything. All we need is a good way to store them, preferably using the native data structures.

1. Propose a data structure that can be used to represent the polynomial

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

   Explain how you would retrieve information about the degree of $p(x)$ and how you would deal with any missing terms (e.g. $p(x) = x^3 - 4$ has no x^2 terms or any x terms).

2. Using whatever convention you adopted above, use `Python` notation to define a polynomial $p(x) = 4x^5 - 15x^2 + 3x - 8$.

   `p=`

3. Write some code for a function that takes `p` (the polynomial as defined above) and `a` (an integer) to evaluate $p(a)$.

4. Explain how you would define functions to integrate and differentiate `p`. What kind of inputs would you need? How would you store the outputs? What kind of rules would you need?

5. Using your outline, write out functions for `differentiate(p)` and `integrate(p)`. Paste the code below.

6. Discuss: how easy was it to write a function to integrate/differentiate polynomials? How easy would it be to integrate/differentiate an arbitrary (read: not necessarily polynomial) function $f(x)$? Would it be easier to

find $f'(x)$ or $f'(a)$ for a specific value of a? What about $\int f(x)dx$ compared to $\int_a^b f(x)dx$? Here we are still assuming that a and b are strictly integers.

2 Elementary Functions

Starting with integers, the easiest numbers to build are rationals. These will just be turned into `float` type objects. That is, even though 2 and 3 are `int` type objects, `a=2/3` is a float. It is worth pointing out that this is not the same as the symbolic version of the fraction $\frac{2}{3}$, as explained here. Since we won't have access to things like radicals or transcendental constants unless we create them and to create them, we need more functions.

7. Define a function to evaluate $n!$ for a positive integer n. Paste the code below.

8. What would happen if you try to use your function on $n = -2$? Assume users are not savvy enough to read documentation. Paste the code again here with any modifications to reduce the likelihood of user error.

9. Write $\sqrt{x}$ as a power series.

10. Define a function to evaluate T_n, the degree n Taylor polynomial approximating a function $f(x)$.

11. How would you approximate $\sqrt{x}$ with a polynomial of degree n using your previously defined functions?

12. How would you approximate $\sqrt[7]{x}$? Paste the code for your function below.

Given that there is a native way to evaluate $2^{1/2}$ as `2**(1/2)`, it seems like a waste to write it up the way we just did. The goal of the last few exercises is to get an idea of how the expression is interpreted (i.e. these power series expansions, or something like it, is what probably runs in the background when you evaluate exponentiation).

Key word *probably*. We can test that hypothesis. The `timeit` module lets us measure how long it takes something to run. For example the code below compares the running times of a function that raises a number to the power of 4 and a function that square roots a number.

```
import timeit

def f(x):
    return x**(1/2)
def g(x):
    return x**4

print(timeit.timeit('f(42)', globals=globals()))
print(timeit.timeit('g(42)', globals=globals()))
```

The outputs are informative:

```
0.1712149130180478
0.322200040332973
```

It took almost twice as long to compute $\sqrt{42}$ than 42^4. This may be dependent on the input though.

13. Fill out the table below (round your output times to the nearest thousandth):

x	10	100	1000	$10,000$	$100,000$
$x+2$					
$2x$					
x^2					
$\sqrt{x}$					

14. Does the time depend on how large the input is? What makes a bigger difference: the operation carried out or the input?

15. We can use this to compare our "made from scratch" algorithm for roots against the native exponentiation. Design an algorithm to find an approximation that is accurate to 10 decimal places when compared against the exponentiation. That is, the input should be the function to run and the output should be the degree n of the polynomial that produces that level of accuracy.

16. Fill out the table again, this time with the degree of accuracy needed, n.

x	10	100	1000	$10,000$	$100,000$
n needed to compute $\sqrt{x}$					
time to evaluate `x**(1/2)`					
time to run $\sqrt{x}$ with power series					

17. Do you think `Python` is using an algorithm like the one you designed? Why?

3 Transcendentals

Algebraic numbers are those that correspond to solutions of polynomial equations with integer coefficients(at most, they just involve repeated use of the standard operations $+, -, \cdot, \div$). *Transcendental* numbers are irrational numbers that are not algebraic. We won't prove it here, but constants like e and π are transcendental.

It is impossible to list all transcendental numbers (there are more of them than algebraic numbers, and there's more of those than rational numbers, which is already saying something). Beyond fractions or powers of e and π, other familiar transcendental numbers include outputs of logarithms and trigonometric functions. We'll work on approximating those next.

17. Write an expression to obtain e as a power series.

18. Write an expression to obtain $\ln(2)$ as a power series.

19. Write **two expressions** to obtain π as a power series.

20. Find the sum

$$4\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots+(-1)^n\frac{1}{2n+1}+\cdots\right)$$

21. Write an algorithm to evaluate the first 1000 terms of the sum above. Write the value of s_{1000} here:

$$s_{1000} =$$

22. To evaluate the accuracy of that partial sum approximation, find the difference between it and the exact

value and write down the error here:

$$\text{error} =$$

23. We could (but will not) prove that

$$\arctan(1) = \arctan\left(\frac{1}{2}\right) + \arctan\left(\frac{1}{3}\right)$$

using angle sum formulas for $\tan\theta$. We will use this for a new approximation. Write out a power series expansion for

$$4\left[\arctan\left(\frac{1}{2}\right) + \arctan\left(\frac{1}{3}\right)\right]$$

24. Write some code to figure out how many terms of the series are needed to get the error you calculated above. Paste the code below.

25. Which of the two expansions converges faster? Explain.

26. If you had to approximate π to a high degree of accuracy, what would you use? Justify using the work above.

Mini Project 4: Python
Rubric

	1	2	3	4	5
Code analysis	All answers are missing or incorrect	A majority of answers are missing or incorrect	Some answers are missing or incorrect	Most answers are correct	All answers are correct
Code use	All answers are left blank	A majority of answers are left blank	About half of the answers are left blank	Most spaces are filled in correctly (some are missing or incorrect)	All spaces are filled in correctly
Calculus concepts	The work shows serious misunderstandings when using calculus concepts	The work shows some misunderstandings when using calculus concepts	The explanations given are generally correct but incomplete	Most explanations are correct and complete	All explanations are correct and complete
Math work	Many solutions are incorrect or incomplete	A few solutions are incorrect or incomplete	Some solutions are missing steps or have small errors	A few solutions are missing steps or have small errors	All solutions are correct and complete
Clarity	It is hard to read/follow the work	Some of the work is hard to read/follow	The organization/tidiness leaves room for improvement but is readable	The work is generally easy to read/follow	It is very easy to read/follow the work done
Bonus problem	The problem was not attempted	Some of the plots are missing and/or the prediction is incorrect or missing	The plots are complete but not on the same graph and/or the prediction is incorrect	The plots are complete and on the same graph but the prediction is incorrect	The plots are complete and ont he same graph and the prediction is correct