I. Problems

1. Evaluate the following Big O and Little o expressions:

- Assumptions: (whole problem) We will assume the standard definitions of "big O" (O) and "little o" (o) and thus use them to determine the validity of the problem statements. We will also assume that, for the following problems, the functions given are **continuous** and **differentiable** around the approached value ($x, \varepsilon = 0$ for a,b and d and $t = \infty$ for c), thus giving us access to *L'Hopital's Rule*.

- Problem(s): Prove the following statements below using the following definitions of "big O", O, and "little o", o.

$$\lim_{x \to a} \frac{f(x)}{g(x)} = 0 \text{ then,}$$

$$\boxed{\text{f(x)=o(g(x))}}$$

and

$$\lim_{x \to a} \frac{f(x)}{g(x)} = C \text{ where } \{C \in \mathbb{R}; C \neq 0\} \text{ then,}$$

$$\boxed{\text{f(x)=O(g(x))}}$$

i. Prove that $(1 + x)^n = 1 + nx + o(x)$ as $x \to 0$

- The work for this problem is listed on the next page (on paper):

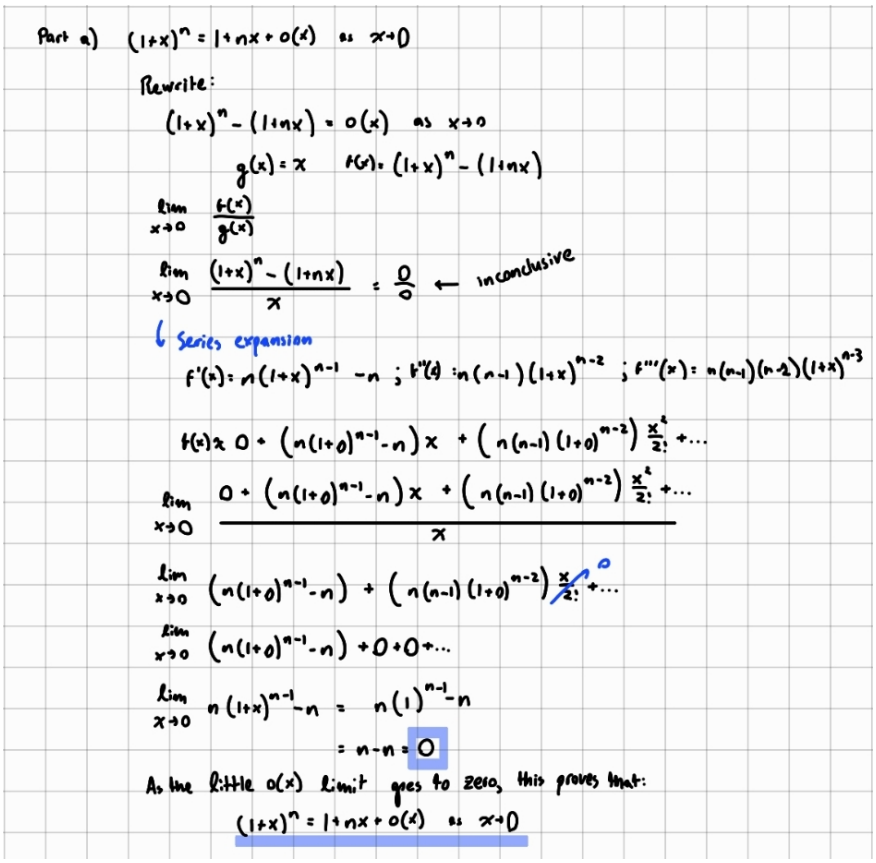

Figure 1: Problem 1.a - Written proof for Part a

ii. Prove that $x\sin(\sqrt{x}) = O(x^{3/2})$ as $x \to 0$

- The work for this problem is listed below (on paper):

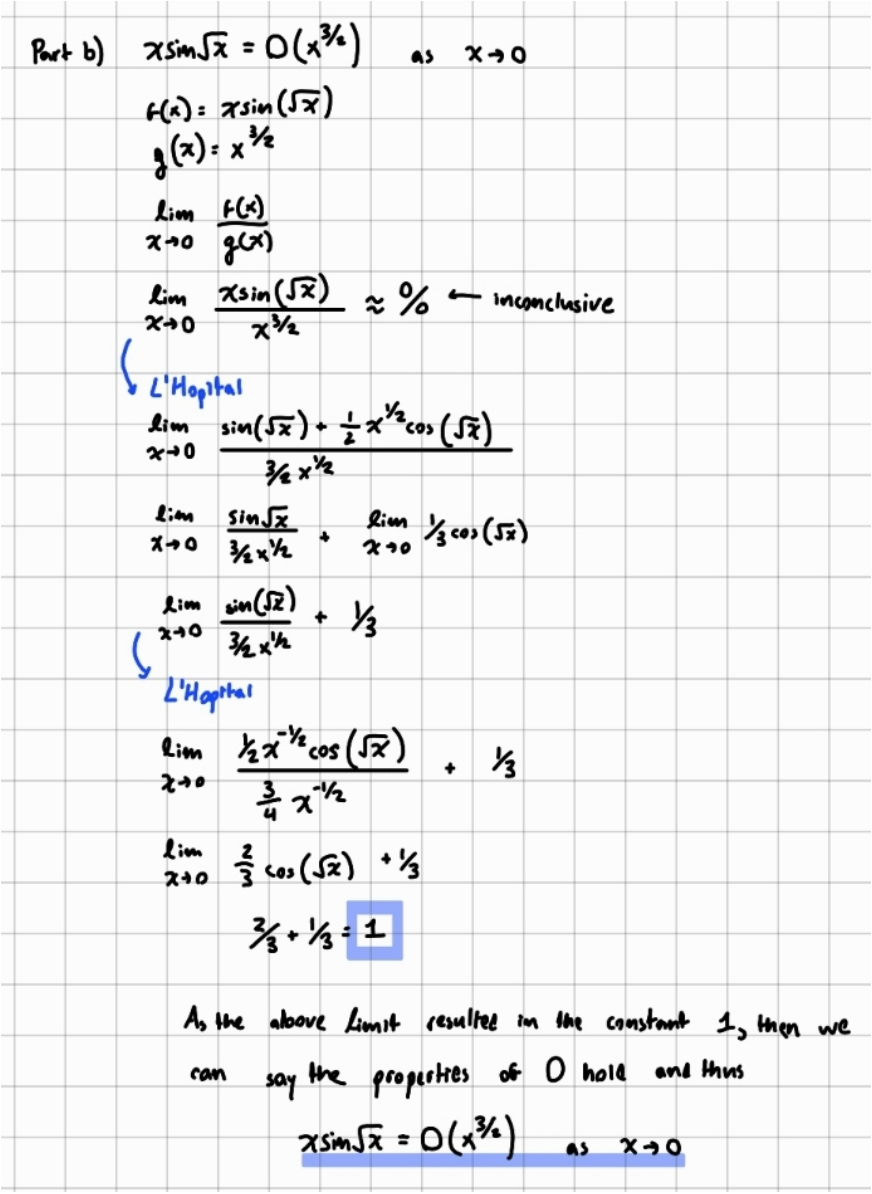

Figure 2: Problem 1.b - Written proof for Part b

iii. Prove that $e^{-t} = o(\frac{1}{t^2})$ as $t \to \infty$

- The work for this problem is listed below (on paper):

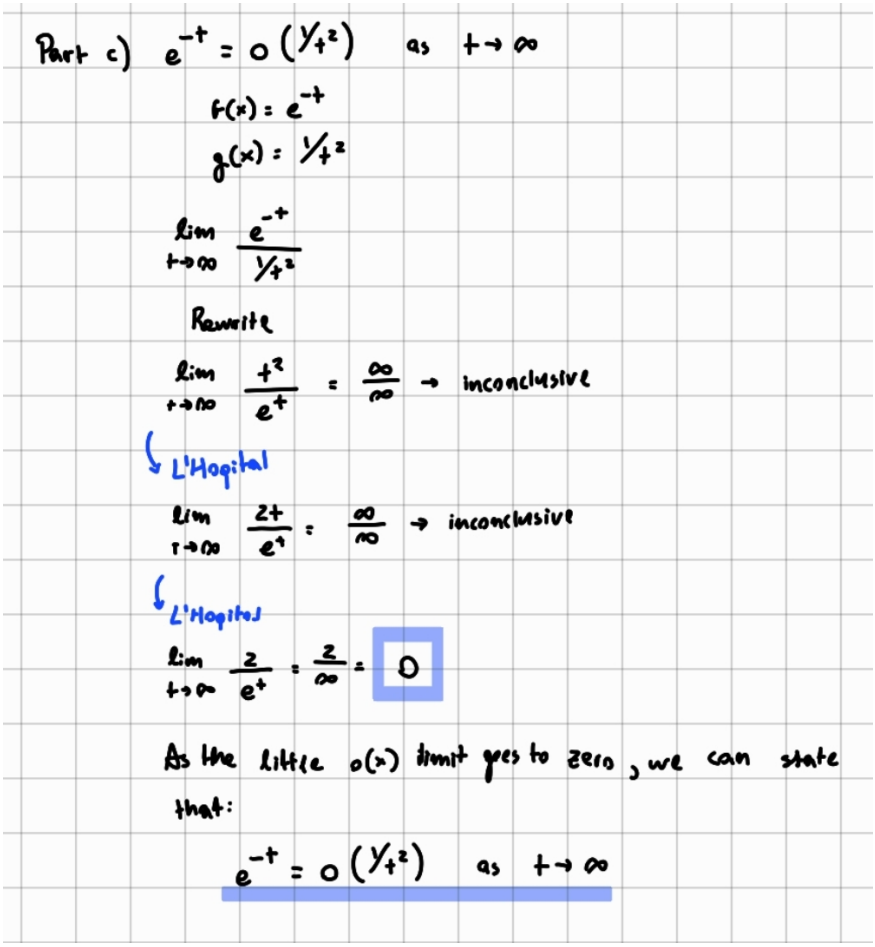

Figure 3: Problem 1.c - Written proof for Part c

iv. Prove that $\int_0^{\varepsilon} e^{-x^2} dx = O(\varepsilon)$ as $\varepsilon \to 0$

- The work for this problem is listed below (on paper):

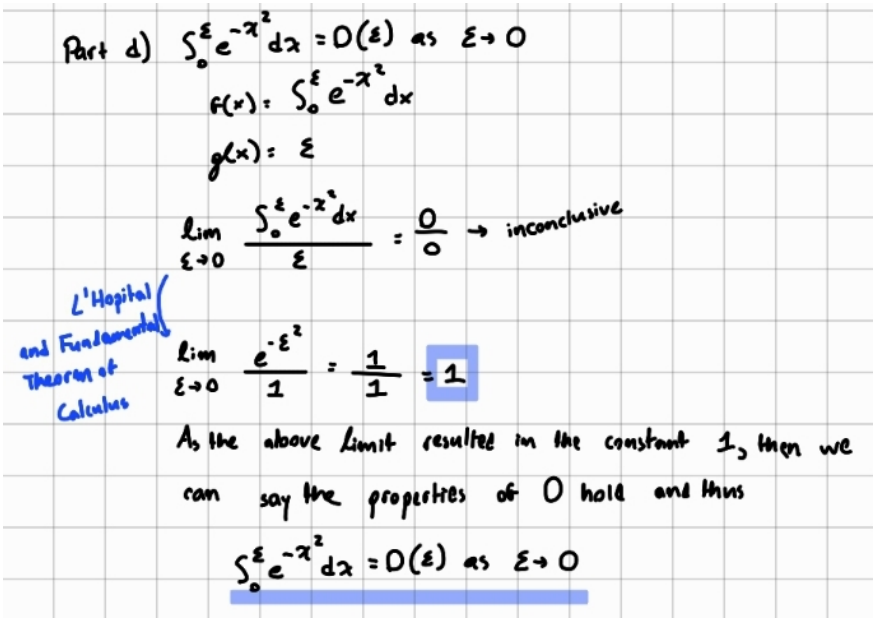

Figure 4: Problem 1.d - Written proof for Part d

- Discussion: All four problems above came to the expected conclusion based on the given "Show that" statements. Using *L'Hopital's Rule* made these problems relatively eloquent and each came to a definitive, finite limit that proved the O/o statements.

2. Given the equation $Ax = b$ where $A = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1+10^{-10} & 1-10^{-10} \end{bmatrix}$ and $b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. We know that the exact solution is $x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and that the inverse of A is $\begin{bmatrix} 1-10^{10} & 10^{10} \\ 1+10^{-10} & -10^{-10} \end{bmatrix}$. We will look at a perturbation in b of $b = \begin{bmatrix} \Delta b_1 \\ \Delta b_2 \end{bmatrix}$ and the effects of the Condition Number.

- Assumptions: For this problem we will make the assumption that the matrix A given is invertible and thus the inverse given is accurate for deriving solutions. Further, we will be assuming that the perturbations in b will be small, thus not affecting the behavior the solution.

i. Determine an exact formula for the change in the solution between the exact problem given above an the perturbed problem of Δx

- Problem: The change in solution can be determined through using simple matrix math and the calculation is listed below:

$$Ax = b$$

$$x = A^{-1}b$$

Then, we will consider a perturbation in b and thus the b vector would take the form:

$$b = \begin{bmatrix} 1 + \Delta b_1 \\ 1 + \Delta b_2 \end{bmatrix}$$

The equation solved for x would then be:

$$\begin{bmatrix} 1 + \Delta x_1 \\ 1 + \Delta x_2 \end{bmatrix} = \begin{bmatrix} 1 - 10^{10} & 10^{10} \\ 1 + 10^{-10} & -10^{-10} \end{bmatrix} * \begin{bmatrix} 1 + \Delta b_1 \\ 1 + \Delta b_2 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 - 10^{10} & 10^{10} \\ 1 + 10^{-10} & -10^{-10} \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

And:

$$\begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} = \begin{bmatrix} 1 - 10^{10} & 10^{10} \\ 1 + 10^{-10} & -10^{-10} \end{bmatrix} * \begin{bmatrix} \Delta b_1 \\ \Delta b_2 \end{bmatrix}$$

So,

$$\Delta x = A^{-1}\Delta b$$

- Discussion:

- Problem: As discussed in lecture, the condition number for A for the above equation is:

$$\kappa_A(x) \leq ||A||_2||A^{-1}||_2$$

Which can be simplified to:

$$\kappa_A(x) \leq \frac{\sigma_{max}}{\sigma_{min}}$$

We can then use *linalg.svd* within numpy to find the singular values of the matrix A.

ii. What is the condition number of A?

- Problem: For this problem we were given multiple methods in lecture to determine the condition number of a matrix. The method that was decided upon (based on ease of calculation within Python's numpy library) was using the max singular value and the minimum singular value of the A matrix to calculate the condition number. The formula I used was as follows:

$$\kappa(f) \leq \frac{\sigma_{max}}{\sigma_{min}}$$

And with the formula I found that the condition number of A equaled: $\boxed{19999975363.813427}$ or approximately $\boxed{2 \times 10^{10}}$

iii. Set Δb_1 and Δb_2 to a magnitude of 10^{-5}. Given those, determine the relative error in the solution? Also, what is the relationship between the relative error, condition number, and perturbation? Does this behavior change if the perturbations are the same? Which is more realistic: changed value of perturbation or the same value.

- Problem/Discussion: For this problem, I choose Δb_1 and Δb_2 to be equal to 10^{-5} and 3×10^{-5} respectively. This gave me a relative error in the solution to be 12.0×10^{10}. After running more tests with the code (changing the values of Δb_1 and Δb_2) it was determined that the relative error was always about one order of magnitude larger than the condition number. This varied with more disturbed inputs, but the relationship that began to appear (even with greatly disturbed inputs) was the following:

$$(RelativeError)_{outputs} \leq \kappa \times (RelativeError)_{inputs}$$

The relation closely aligns with what we were shown in lecture and is shown in the data when tested with different disturbances. To elaborate more, the direct answers to the three questions asked in the prompt are listed below:

- Relative error, condition number and perturbation are related to each other via the following inequality: $(RelativeError)_{outputs} \leq \kappa \times (RelativeError)_{inputs}$.
- When the perturbations are the same the relative error of the outputs is almost exactly equal to the relative error of the inputs, thus making the above inequality simplify to: $1 \leq \kappa$.
- A different value of perturbation is more realistic as it is very rare for both elements of a vector to be disturbed to the same magnitude in the same direction at the same time.

3. Let $f(x) = e^x - 1$

- Assumptions: We will assume for this problem that the function above is both differentiable and continuous for all x. Further we will assume that we are operating with double precision and thus cannot get more precise than an error of 10^{-16}.

i. What is the relative condition number $\kappa(f(x))$? What values of x would this be ill-conditioned

- Problem: The calculation for the relative condition number of this function was determined by hand and is attached below:

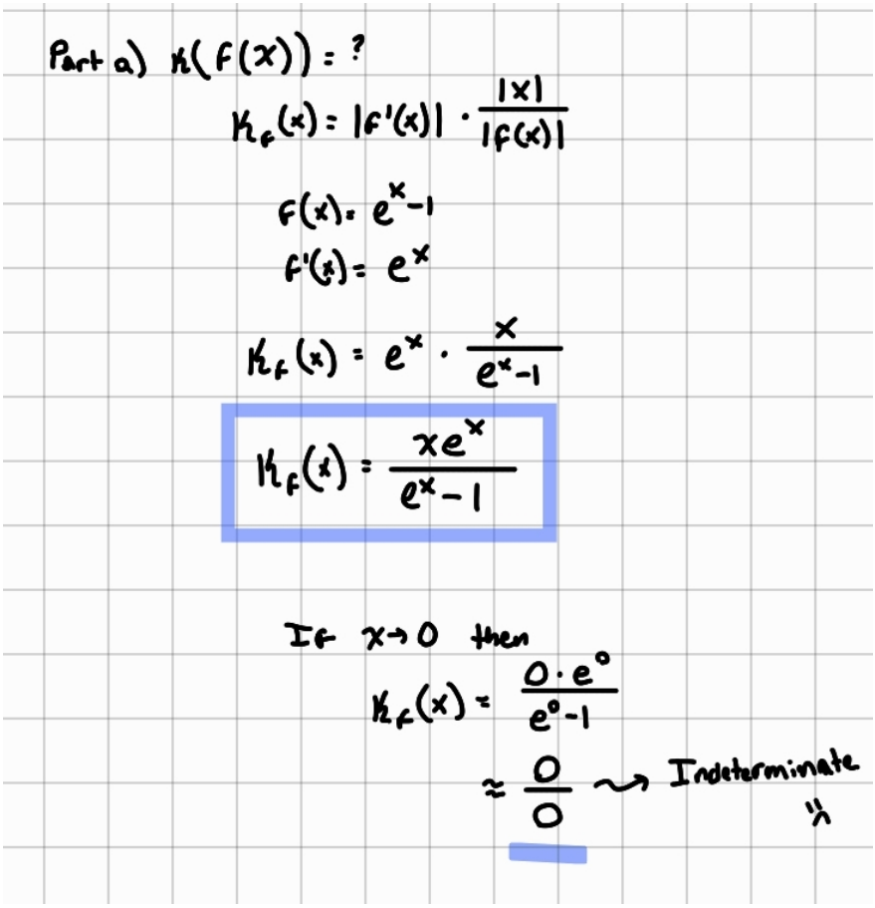

Figure 5: Problem 3.a - Relative Condition Number Calculation

- Discussion: As can be seen above, this is a relatively eloquent solution for the condition number, albeit for the point at zero or approaching zero, as the denominator and the numerator will both be forced to zero, resulting in an ill-conditioned scenario.

ii. Compute the algorithm for f(x) given. Is it stable?

- Problem/Discussion: As can be seen in the code pushed to GitHub for this problem, the algorithm was determined to be stable. This is because, despite the subtraction, the e^x, around points approaching zero, will not be affected by the -1 as small changes in x will not be close enough in relative scale to 1. That being said, there is a chance of breakdown around points quite close to $x = 0$, but for most practical applications this will reside well within the margin for error.

iii. Let x have the value $9.99999999500000 \times 10^{-10}$, thus $f(x)$ is equal to 10^{-9} up to 16 decimal places. What is the number of correct digits that the algorithm gives you? Is this what we would expect?

- Problem: Using the algorithm listed in the problem statement, it was determined and can be seen in the code that the algorithm was correct up to 8 digits ($1.0000000082740317 \times 10^{-9}$).

- Discussion: This is expected as x in this case is extremely close to zero and close to the potential cancellation case noted in the previous question, where it gets increasingly close to $1 - 1$ and thus results in an error.

iv. Find a potential polynomial approximation of $f(x)$ that is correct to 16 digits for $9.99999999500000 \times 10^{-10}$.

- Problem: The Taylor polynomial used to approximate this problem was calculated on paper and is listed below:

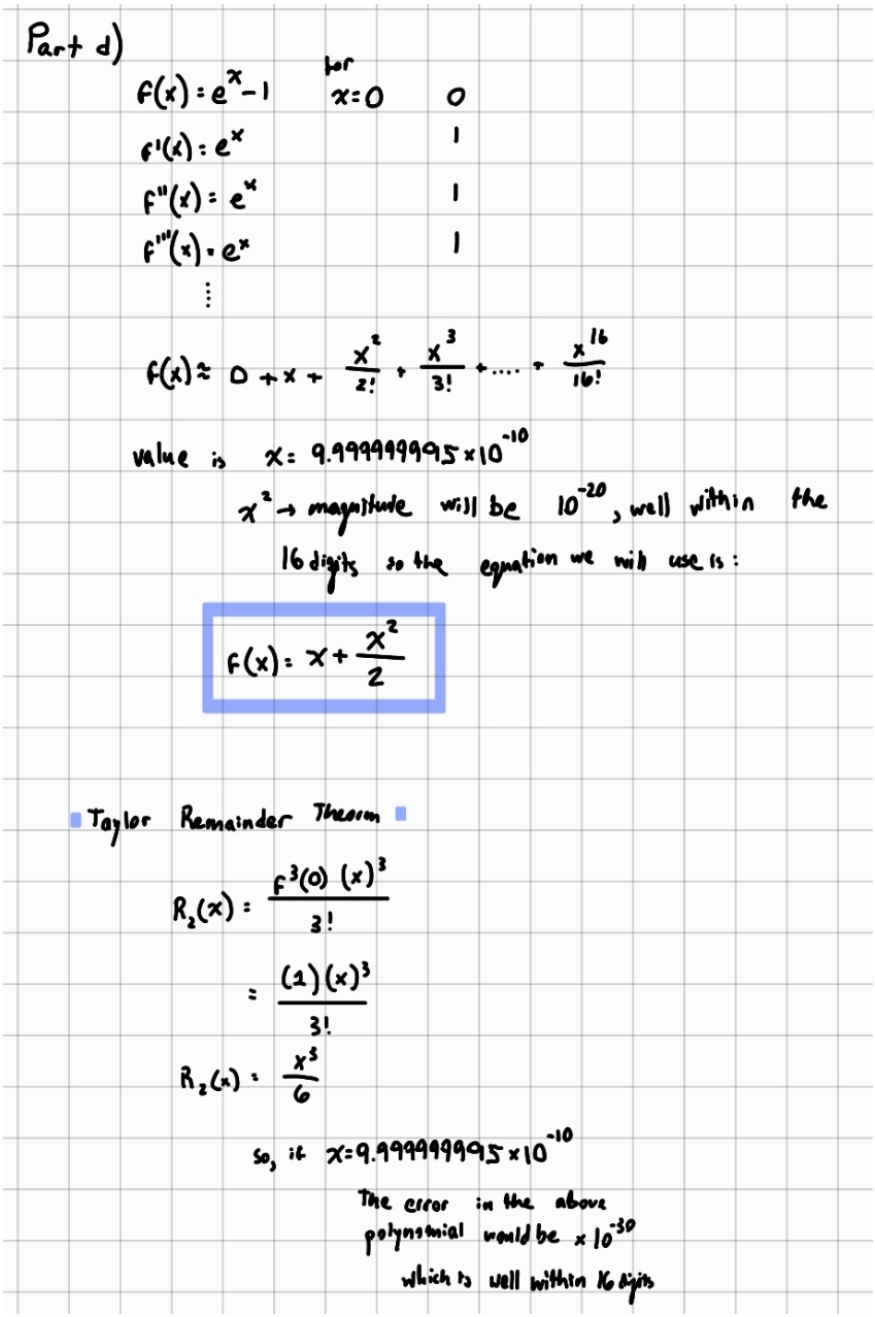

Figure 6: Problem 3.d - Taylor Polynomial Approximation Calculation

- Discussion: Determining to cut off the Taylor polynomial at the x^2 term was done for conciseness and because we are assuming double precision for this problem and thus there was no point (for the desired x) in making this formulation more accurate

v. Confirm that your answer to part (d) is correct.

- Problem/Discussion: As can be seen in the code uploaded to the GitHub, several iterations of the Taylor polynomial were calculated and the one determined above was the simplest while maintaining 16 digits of accuracy.

4. Practicing Python

i. Using a vector $t = [0 : \frac{\pi}{30} : \pi]$. Then create a vector $y = \cos t$. Write a code the evaluates the sum listed below and prints the statement "The sum is: S."

$$S = \sum_{k=1}^{N} t(k)y(k)$$

- Assumptions: For this code we will assume that the intent of the question is to include the t points of both 0 and 30 and thus, N will have to be 31, as both t and y will have 31 terms.

- Problem: The algorithm used in this problem was fairly simple as it made use of the *sum()* function within python. The code for this problem is listed below:

Ultimately, the final sum printed was **-20.686852364346837**

- Discussion: Using a graphing calculator one could check and confirm that the sum printed by the code (-20.68685...) was correct given the assumptions above. Another way this code could have been calculated without using the *sum()* function would have been to use a for loop, where each prior entry is added to the last.

ii. In one figure, plot the parametric curves listed below with $0 \leq \theta \leq 2\pi$ and $R = 1.2, \delta r = 0.1, f = 15$ and $p = 0$. (Make sure to adjust the axis). In a second figure, plot 10 curves using a *for* and let $R = i, \delta r = 0.05, f = 2 + i$ for the i^{th} curve. Let the value of p be a uniformly distributed random number between 0 and 2.

$$x(\theta) = R(1 + \delta r \sin(f\theta + p)) \cos(\theta) \; x(\theta) = R(1 + \delta r \sin(f\theta + p)) \sin(\theta)$$

- Assumptions: We will assume that 100 terms is enough to create accurate *Wavy circles*, and the random function given in the Homework document generates uniformly distributed random numbers.

- Problem: Using the equations above, and the values given in the problem statement, the following basic parametric curve was generated:

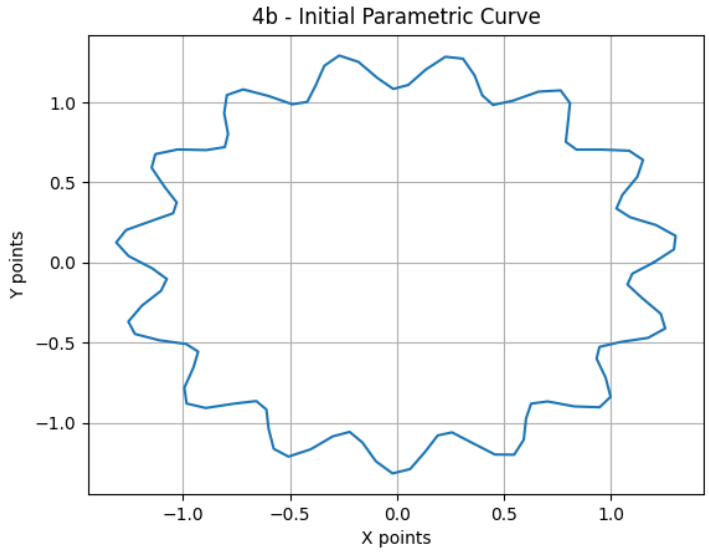

Figure 7: Problem 4.b.1 - Basic Parametric Curve with fixed values

The plot of 10 curves, using a for loop as described, is listed below, with each parametric curve plotted, expanding outward:

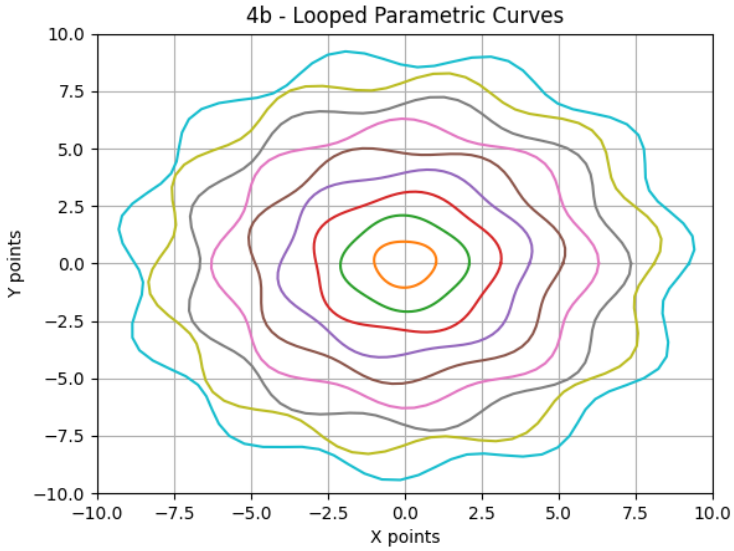

Figure 8: Problem 4.b.2 - Looped Parametric Curves

- Discussion: These curves above are in the shape we would expect (i.e. circles with oscillating edges that grow in size with increasing iterations of the for loop). What is interesting about each of these is that, due to the random phasing and the placement of the ridges on each of the "circles", you can see how much or how little each iteration is out of phase with the previous by how well their peaks and valleys line up with each other.

II. Appendix

1. Code

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

#---------------------------------------#
#Problem Functions

#Problem 1

#Problem 2
def Problem2():
    #Part b
    A = (1.0/2.0) * np.array([[1.0,1.0],
                              [1+10**(-10),1-10**(-10)]])
    b = np.array([[1.0],
                  [1.0]])
    x = np.array([[1.0],
                  [1.0]])
    A_inv = np.array([[1-10**(10),10**10],
                      [1+10**(10),-10**(10)]])

    _,svds,_ = np.linalg.svd(A,full_matrices=False)
    #print(svds)

    sigma_max = max(svds)
    sigma_min = min(svds)
    Condition_Number = sigma_max/sigma_min

    print("The Condition Number for A is ", Condition_Number)

    #Part c
    delta_b = np.array([[10^(-5)],[3*10^(-5)]])
    #delta_b = np.array([[10^(-5)],[10^(-5)]])
    #delta_b = np.array([[10^(-2)],[3*10^(-2)]])
    b_pert = b+delta_b
    x_pert = A_inv@b_pert

    err_abs = np.linalg.norm(x-x_pert)
    err_rel = np.linalg.norm(x-x_pert)/np.linalg.norm(x)

    err_rel_in = np.linalg.norm(delta_b)/np.linalg.norm(b)
```

```
    print("The absolute error resultant from the perturbation ", err_abs)
    print("The relative error resultant from the perturbation ", err_rel)
    print("The relative error in the inputs is ", err_rel_in)
    print("The Condition number times thr relative error in the inputs is: ", err_rel_in

    '''kappa_denom = (np.sqrt(b_pert[0]**2+b_pert[1]**2)/np.sqrt(b[0]**2+b[1]**2))
    kappa_num = ((np.sqrt(err_abs[0]**2+err_abs[1]**2)/np.sqrt(x[0]**2+x[1]**2)))
    kappa = kappa_num/kappa_denom'''

#Problem 3
def Problem3():
    #Justifying part b)
    x_test = np.linspace(-1*10**(-14),1*10**(-14),100)
    y_test = np.e**x_test
    #print((y_test - 1))

    #part c)
    x = 9.999999995000000*(10**(-10))
    y = np.e**x
    print(y-1)

    #part e)
    y_taylor_1 = x + (x**2)/(1*2) #+ (x**3)/(1*2*3) + (x**4)/(1*2*3*4) + (x**5)/(1*2*3*4

    y_taylor = y_taylor_1
    print(y_taylor)




#Problem 4
def Problem4a():
    t = np.arange(0,(31*np.pi/30),(np.pi/30))
    #print(t)
    y = np.cos(t)
    s = t * y
    S = sum(s)
    print("the sum is: ", S)

def Problem4b():
    #Part 1
    theta = np.linspace(0,2*np.pi, 100)
    R = 1.2
    dr = 0.1
```

```
    f = 15
    p = 0
    x = R * (1 + dr * np.sin(f*theta + p)) * np.cos(theta)
    y = R * (1 + dr * np.sin(f*theta + p)) * np.sin(theta)

    #Plotting Initial Figure
    plt.figure()
    plt.plot(x,y)
    plt.title("4b - Initial Parametric Curve")
    plt.xlabel("X points")
    plt.ylabel("Y points")
    plt.grid(True)
    plt.show()

    #Part 2
    plt.figure()
    for i in range(10):
        R = i
        dr = 0.05
        f = 2+i
        p = random.uniform(0,2)
        x = R * (1 + dr * np.sin(f*theta + p)) * np.cos(theta)
        y = R * (1 + dr * np.sin(f*theta + p)) * np.sin(theta)
        plt.plot(x,y) #Calling plot within the loop
    plt.title("4b - Looped Parametric Curves")
    plt.xlabel("X points")
    plt.ylabel("Y points")
    plt.xlim([-10,10])
    plt.ylim([-10,10])
    plt.grid(True)
    plt.show()



#---------------------------------------#
# Main Script

#Calling Problem 2
Problem2()

#Calling Problem 3
Problem3()

#Calling Problem 4
Problem4a()
```

```
Problem4b()
```