I. Problems

1. Consider the polynomial:

$$p(x) = (x-9)^2 = x^9 - 18x^8 + 144x^7 - 673x^6 + 2016x^5 - 4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512$$

i. Given the p(x) above, evaluate p via its coefficients for x = 1.920, 1.921, 1.922, ... , 2.080 (i.e. x = [1.920 : 0.001 : 2.080];) .

- Assumptions: We will be assuming for this problem that the numbers used for the following plot(s) are in double precision and that the finite distribution of points listed above shows to sufficient detail the issues with the problem.

- Problem: In order to solve this problem it was necessary to preform a simple Python script (see the appendix) and generate the plot using the expanded coefficient form of the expression given in the problem statement. The plot generated is shown below:

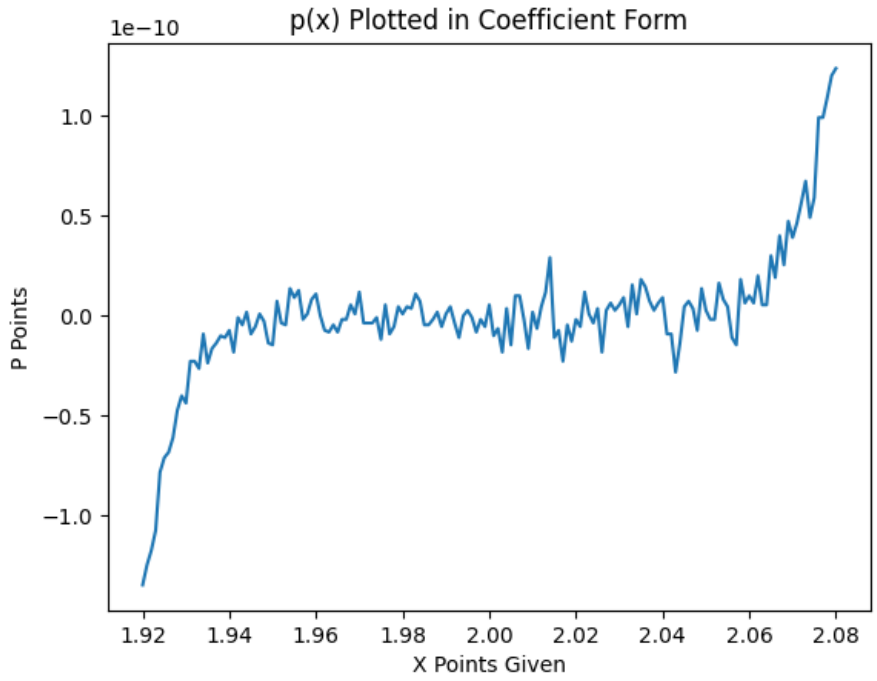

Figure 1: Problem 1.i - Plot Generated in Coefficient form

- Discussion: (Discussion for this problem will be left to section iii)

ii. Now evaluate p via the simplified expression $(x-9)^2$ and graph it.

- Assumptions: (same as above)

- Problem: In order to solve this problem it was necessary to preform a simple Python script (see the appendix) and generate the plot using the simplified binomial form of the expression given in the problem statement. The plot generated is shown below:

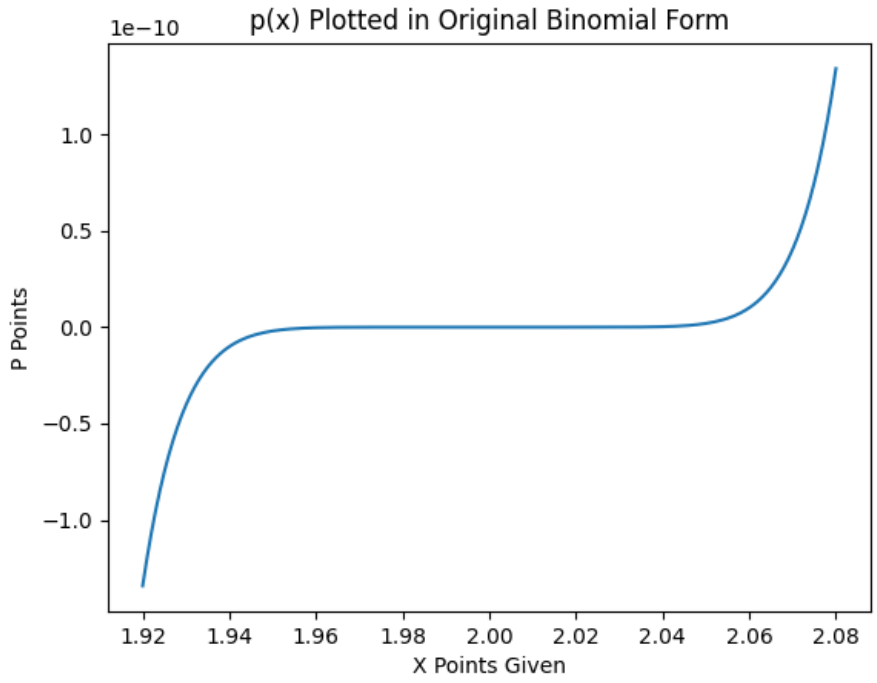

Figure 2: Problem 1.i - Plot Generated in Coefficient form

- Discussion: (Discussion for this problem will be left to section iii)

iii. What is the difference between the two data sets (plots) generated above and why do you think this may be occuring?

- Discussion: The difference between these two plots is that the first plot contains a number of inaccuracies in the data, shown by the irregularities in the plot, while the second plot presents a much cleaner distribution of the $p(x)$ function. What is causing this discrepancy is the amount of potential error points in the distributed version on the equation as each subtraction (of which there are 5) presents a potential source for error, while the simplified version of the equation only has one potential source for error and that is points incredibly (within double precision) close to 2.

2. Given the nature of catastrophic cancellation, determine a way to calculate the following expressions to avoid this. Justify your answers through discussion

- Assumptions: (The following assumptions apply to all 3 sections of this problem) We will assume that the subtraction operator, as demonstrated in class, is the main source for error in these equations and thus the best way to remove potential error would be to eliminate those operators.

i. Evaluate $\sqrt{x+1}-1$ for $x \approx 0$

- Problem: Using the conjugate, the above expression can be simplified to the following to avoid error at $x \approx 0$:

$$\sqrt{x+1}-1$$

$$(\sqrt{x+1}-1) * \frac{\sqrt{x+1}+1}{\sqrt{x+1}+1}$$

$$\frac{x+1-1}{\sqrt{x+1}+1}$$

$$\boxed{\frac{x}{\sqrt{x+1}+1}}$$

- Justification/Discussion: The above expression is the same mathematical expression as the original given, however though this process of multiplication with the conjugate, all subtraction operations have been removed and thus any chance of catastrophic cancellation around 0 has been removed in kind.

ii. Evaluate $\sin x - \sin y$ for $x \approx y$

- Problem: In order to simplify this expression and remove the cancellation we will have to prevent a subtraction of sines.

- Assumptions: As instructed in lecture, we will make the extra assumption for this problem that we know $x - y$ to full precision.

$$\sin x - \sin y$$

Trigonometric Identities:

$$\sin A + B = \sin A \cos B + \cos A \sin B$$
$$\sin A - B = \sin A \cos B - \cos A \sin B$$

$$A = \frac{x+y}{2}, B = \frac{x-y}{2}$$

$$x = A + B, y = A - B$$

Using the Trig identity above and the prior variable definitions, we get:

$$\sin(A+B) - \sin(A-B) = 2\cos(A)\sin(B)$$

$$\sin(x) - \sin(y) = 2\cos(\tfrac{x+y}{2})\sin(\tfrac{x-y}{2})$$

$$\boxed{2\cos(\tfrac{x+y}{2})\sin(\tfrac{x-y}{2})}$$

- Justification/Discussion: This provides a partial solution to the cancellation because a subtraction within the sine function will still retain the amount of accuracy the built-in sine function can withstand, where as the original form would remove accuracy after the sine of each value was calculated and thus result in a less accurate final answer. With our assumption that we know $x-y$ to full precision however, we can state that the above answer is satisfactory in removing any catastrophic cancellation.

iii. Evaluate $\frac{1-\cos x}{\sin x}$ for $x \approx 0$

- Problem: In this problem, we need to remove the $1-\cos(x)$ from the numerator, and this can be done through the use of simple Trig identities.

$$\frac{1-\cos x}{\sin x}$$

$$\frac{1-\cos x}{\sin x} * \frac{1+\cos x}{1+\cos x}$$

$$\frac{1-\cos^2 x}{\sin x(1+\cos x)}$$

$$\frac{1-1+\sin^2 x}{\sin x(1+\cos x)}$$

$$\frac{\sin^2 x}{\sin x(1+\cos x)}$$

$$\boxed{\frac{\sin x}{(1+\cos x)}}$$

- Justification/Discussion: Much like the first problem in this set, this is a fairly simple rewrite. Through the use of a conjugate-like multiplication as well as the Pythagorean identity ($\sin^2 x + \cos^2 x = 1$), one can achieve this relatively eloquent simplification of the original formula without any subtractions which would result in catastrophic cancellation.

3. Determine the second degree Taylor polynomial $P_2(x)$ for $f(x) = (1 + x + x^3)\cos x)$ about $x_0 = 0$

- Assumptions: We are assuming that a Taylor Polynomial about 0 is an accurate formula for estimating the given function around $x = 0.5$. Further, we are assuming that the "error formula" alluded to in the question is the following: (Note, these assumptions will be used throughout this question)

$$R_n(x) = \frac{M}{(n+1)!}(x-a)^{n+1}$$

$$\text{where } M \geq |f^{(n+1)}(z)| \text{ for all } z \in [a, x]$$

- Problem:

$$f(x) = (1 + x + x^3)\cos x$$

$$P_2(x) = f(0) + \frac{\frac{d}{dx}[f(x)](0)}{1!}x + \frac{\frac{d^2}{dx^2}[f(x)](0)}{2!}x^2$$

$$f(0) = 1$$

$$\frac{d}{dx}[f(x)] = (1 + 3x^2)\cos(x) - (1 + x + x^3)\sin(x)$$

$$\frac{d}{dx}[f(x)](0) = 1$$

$$\frac{d^2}{dx^2}[f(x)] = 6x\cos(x) - (1 + 3x^2)\sin(x) - (1 + 3x^2)\sin(x) - (1 + x + x^3)\cos(x)$$

$$\frac{d^2}{dx^2}[f(x)](0) = -1$$

$$\frac{d^3}{dx^3}[f(x)] = 6\cos(x) - 6x\sin(x) - 6x\sin(x) - (1 + 3x^2)\cos(x) - 6x\sin(x) - (1 + 3x^2)\cos(x) + (1 + x + x^3)\sin(x) - (1 + 3x^2)\cos(x)$$

$$\text{Simplified: } \frac{d^3}{dx^3}[f(x)] = x^3\sin(x) - 9x^2\cos(x) - 17x\sin(x) + 3\cos(x) + \sin(x)$$

So, the second degree Taylor polynomial is:

$$\boxed{P_2(x) = 1 + x - \frac{1}{2}x^2}$$

And the error formula is:

$$\boxed{R_2(x) = \frac{z^3\sin(z) - 9z^2\cos(z) - 17z\sin(z) + 3\cos(z) + \sin(z)}{6}(x)^3}$$

Where z is the the value of x that provides the maximum value for the 3rd derivative

i. Using $P_2(0.5)$, find an approximation for $f(0.5)$ and determine an upper bound for the error $|f(0.5) - P_2(0.5)|$ using the Taylor error formula (listed above) and compare it to the actual error.

$$f(0.5) = 1.426$$

$$P_2(0.5) = 1.375$$

Using the error formula (calculated above) we need to determine z:

$$\frac{d^3}{dx^3}[f(x)] = x^3\sin(x) - 9x^2\cos(x) - 17x\sin(x) + 3\cos(x) + \sin(x)$$

Test the ends of the interval $[0, 0.5]$

$$\frac{d^3}{dx^3}[f(0)] = 3$$

$$\frac{d^3}{dx^3}[f(0.5)] = -2.8776$$

Use 0 for z

$$R_2(0.5) = \frac{3}{6}(0.5)^3$$

$$R_2(0.5) = 0.0625$$

Actual error

$$|f(0.5) - P_2(0.5)| = 1.426 - 1.375 = 0.05107$$

- Discussion: The upper bound for the error 0.0625 is in the same order of magnitude, but slightly larger, than the actual error calculated for the given point, which was 0.05107.

ii. Use $P_2(x)$ is used to approximate $f(x)$ and determine a general bound for the error $|f(x) - P_2(x)|$.

A bound for the error that we could use for when $P_2(x)$ is used to approximate $f(x)$ would be the error formula, where we assume z to be x. This function of x will be:

$$\boxed{R_2(x) = \frac{x^3\sin(x) - 9x^2\cos(x) - 17x\sin(x) + 3\cos(x) + \sin(x)}{6}(x)^3}$$

- Discussion: While this may not be the exact upper bound for the error, the x^3 present in the original equation means that the farther a point is from zero, the more likely it would be that the x we are evaluating at would give the largest value for the third derivative of $f(x)$.

iii. Use $\int_0^1 P_2(x)dx$ to approximate $\int_0^1 f(x)dx$.

$$\int_0^1 P_2(x)dx = \int_0^1 1 + x - \frac{1}{2}x^2 dx$$

$$[x + \frac{1}{2}x^2 - \frac{1}{6}x^3]_0^1$$

$$1 + 1/2 - 1/6 = \boxed{\frac{4}{3}}$$

iv. Estimate the error in the integral.

We can estimate the error of the integral by evaluating:

$$\int_0^1 |R_2(x)|dx$$

However, on this interval, we need to choose a z for our error function. For the sake of simplicity, and the fact that the question does not ask for the upper bound on the error, we will choose z=0 as it is within the domain evaluated on this integral ($[0, 1]$) and thus the formula simplifies to:

$$\int_0^1 \frac{1}{2}x^3 dx$$

We can evaluate this to get the following estimate on the error:

$$[\frac{1}{8}x^4]_0^1$$

$$1/8 = \boxed{0.125}$$

- Discussion: If we were to use code to evaluate the actual integral of $f(x)$, we would find that the error is approximately 0.06165 which is roughly half of our error estimate listed above. This is not surprising however, as the formula used for $R_2(x)$ was intended to arrive at a maximum bound on the error and thus it being in the same order of magnitude but larger is expected.

4. Consider the quadratic equation $ax^2 + bx + c$ with $a = 1, b = -56, c = 1$.

- Assumptions: We are assuming, for this problem, that Python's precision when calculating square roots is much higher than 3 decimal places and thus it can be used to compare values. Further, we can assume that the phrase "3 decimals" refers to points after the decimal place in standard base 10 and not in scientific notation.

i. Making the assumption that we can calculate the square root to 3 correct decimals (e.g. $\sqrt{2} \approx 1.414 \pm \frac{1}{2}10^{-3}$ and find the relative errors for the two roots to the quadratic when computed using the standard formula:

$$r_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Problem: The work for this problem was done by hand and was compared and check with the code listed in the appendix. The by hand work is listed below.

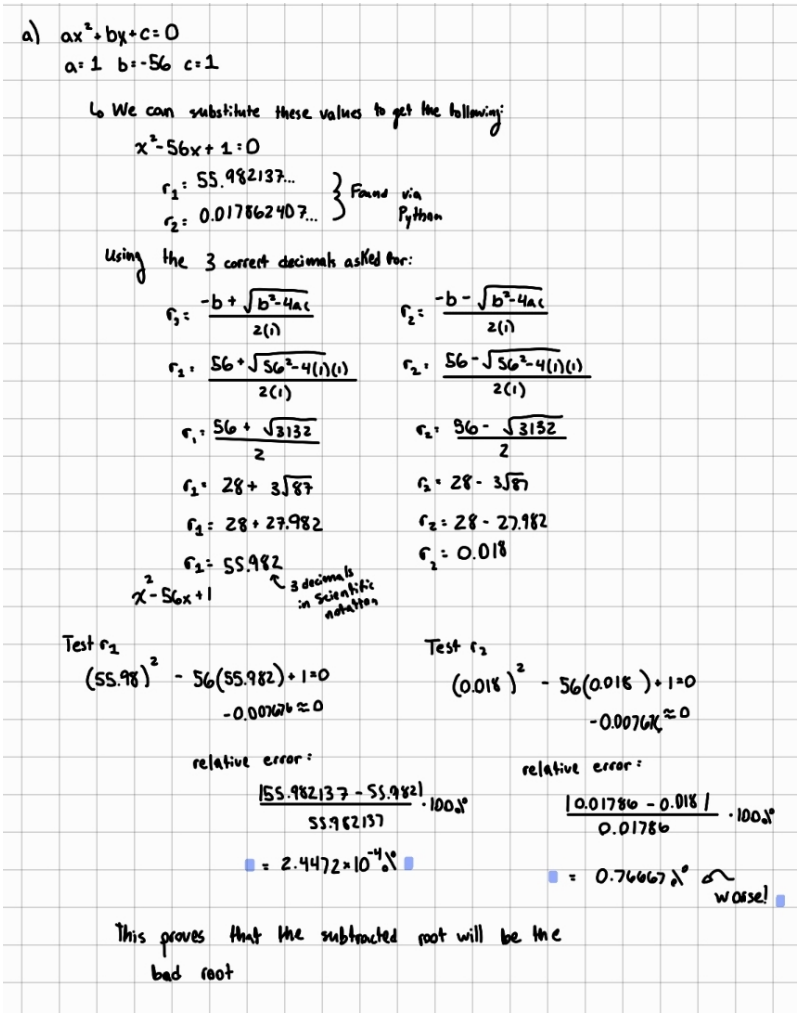

Figure 3: Problem 4.a - Determining Relative Errors

- Discussion: This problem had the result we would expect, where the root which was derived through a subtraction operation resulted in more error accumulation than the one containing an addition operation. As mentioned in the assumptions, this error was determined through the use of Python's results using the same formula, and thus that would also have some error, albeit much smaller than determined above.

ii. Find better approximation for the "bad" root by manipulating $(x - r_1)(x - r_2) = 0$ and relating it to a, b, c. Find the two relations and see if either can be used to find a more accurate estimation of the "bad" root.

- Problem: The work for this problem was done by hand and is listed below.

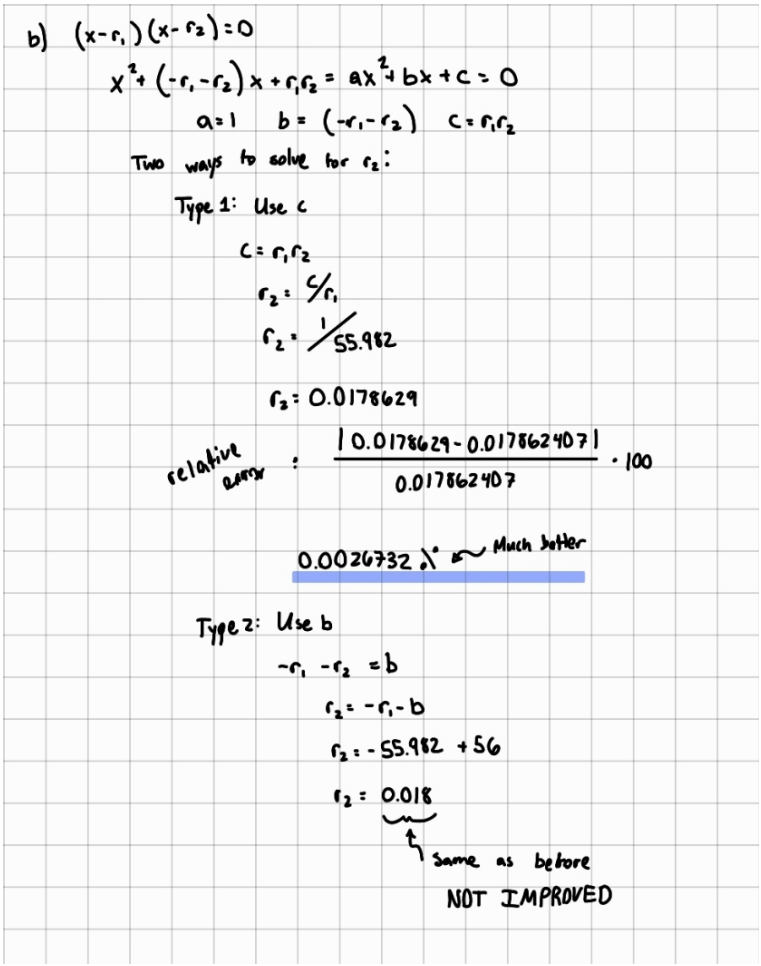

Figure 4: Problem 4.b - Using b and c to approximate r_2

- Discussion: Looking at the two methods used here, it should come as no surprise that the method I attempted first (the one using c) was more accurate as it had no subtraction and thus no chance for possible error to accumulate. Whereas the other method (the one using b) used the exact same subtraction seen in part a.) and thus would have the same relative error.

5. If one were to compute $y = x_1 - x_2$ with $\tilde{x}_1 = x_1 + \Delta x_1$ and $\tilde{x}_2 = x_2 + \Delta x_2$ being approximations to the exact values. If the operation $x_1 - x_2$ is carried out we have $\tilde{y} = y + (\Delta x_1 - \Delta x_2)$ where $(\Delta x_1 - \Delta x_2) = \Delta y$

- Assumptions: For this problem we will assume that the values chosen for

i. Determine the upper bounds on the absolute error $|\Delta y|$ and the relative $|\Delta y|/|y|$, when is the relative error large?

- Problem: The following problem was calculated/determined theoretically using the properties of the absolute error formula and the relative error formula. The work for this was done by hand and is listed below.

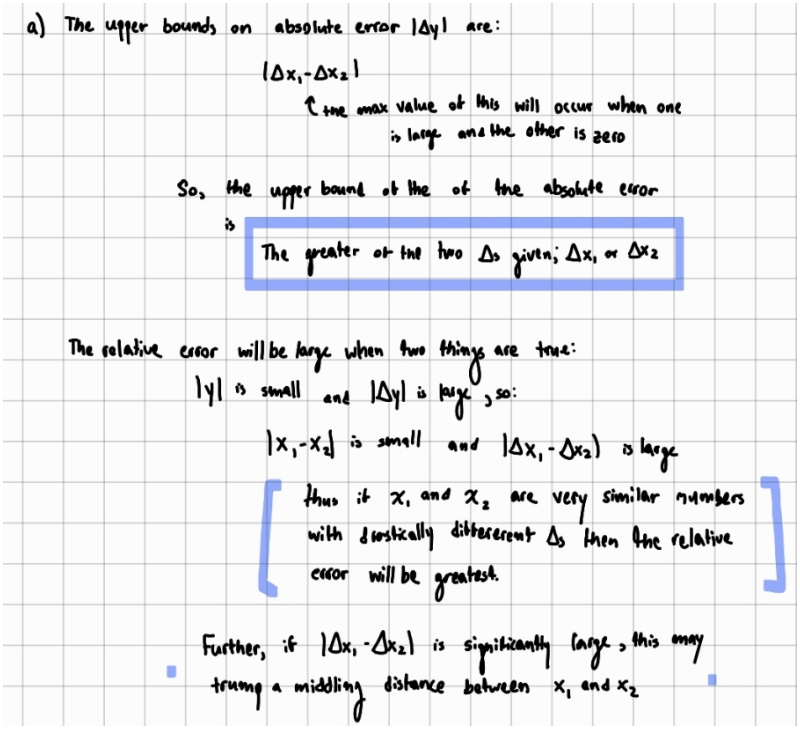

Figure 5: Problem 5.a - Determining Bounds on Absolute and Relative Error

- Discussion: This problem, as the x and Δx values are not strictly defined, left me to solve for the bounds on the two errors theoretically, deriving a lot of my reasoning from the properties of fractions. It is impossible, without given the x and Δx to determine true numerical bounds for these errors, however I believe my explanation gives a good reasoning and explanation for why I choose the bounds I did.

ii. Manipulate $\cos(x+\delta)-cos(x)$ into an expression without subtraction using trigonometric identities. Pick two values of x and plot the difference between your expression and $\cos(x+\delta)-\cos(x)$ for $\delta = 10^{-16}, 10^{-15}, ..., 10^{-2}, 10^{-1}, 10^{0}$.

- Problem: The work for the problem consisted of two steps. The first was to manipulate the given expression to remove the subtraction and the second was to pick two values of x (blank and bigger blank) and tabulate them. The work for the first part was done by hand and is listed below and the work for the second part was coded in python, the final figures of which is also listed below. The two xs chosen were $x_1 = \frac{3\pi}{4}$ and $x_2 = 10^7$.

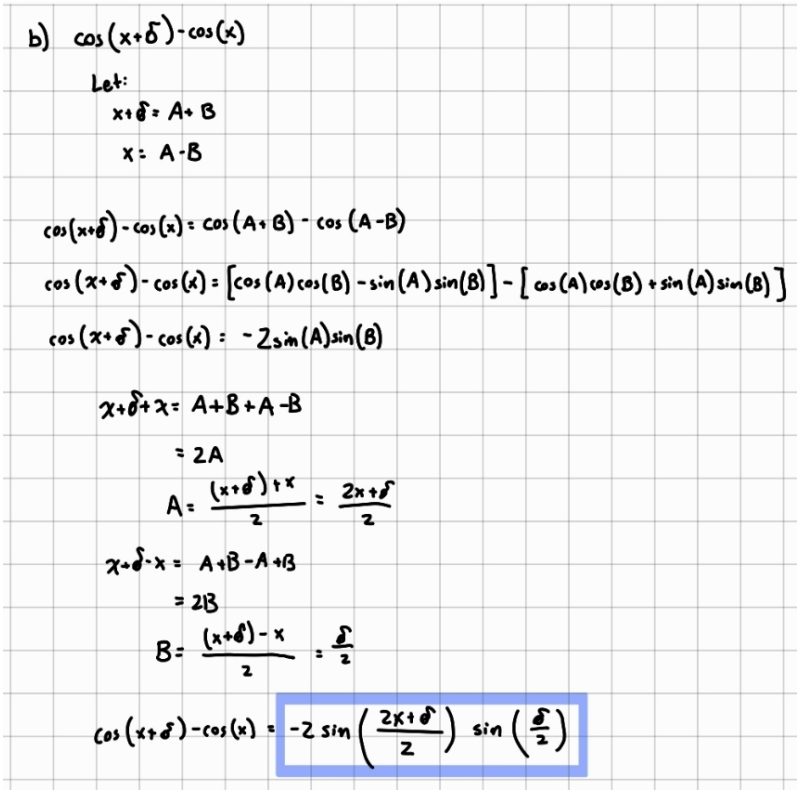

Figure 6: Problem 5.b - Simplified Expression

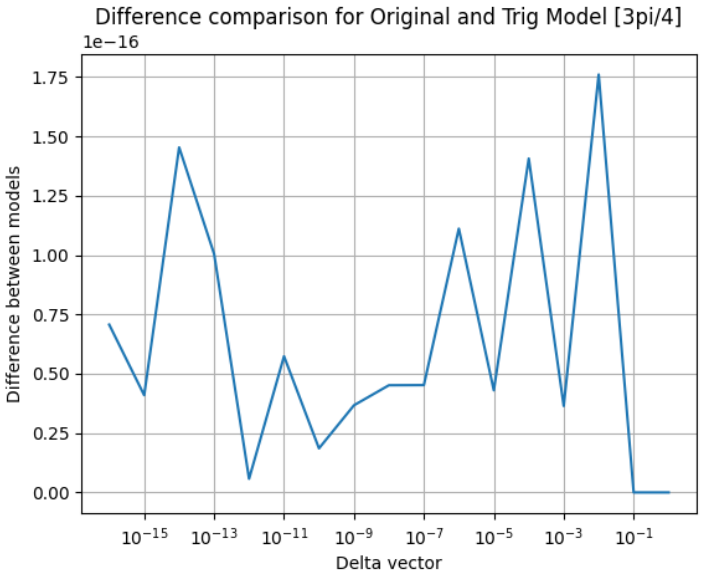

Figure 7: Problem 5.b - Differences between models for $x_1 = \frac{3\pi}{4}$

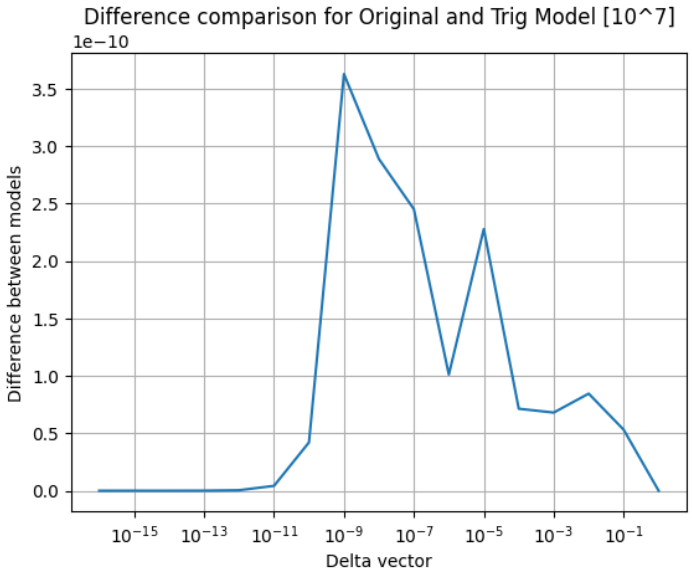

Figure 8: Problem 5.b - Differences between models for $x_2 = 10^7$

- Discussion: Looking at these figures, it is interesting to note that both have relatively consistent error. In particular, the smaller x_1 has a fluctuating error around 10^{-16} that peaks with a delta of 10^{-1}. Meanwhile, the much larger x_2 has a fluctuating error around 10^{-10} that peaks with a delta of 10^{-9}. This discrepancy is likely caused by two factors: 1) the double precision of python, where (for x_2) the smallest values of delta have no effect on the final value as python simply cannot process them, whereas the smaller x_1 would not have this issue, and 2) the error for x_2 is 6 orders of magnitude larger as the values of $x + \delta$ and x are (in Scientific Notation) much closer to each other than their delta equivalents for x_1.

iii. Taylor expansion yields $f(x+\delta) - f(x) = \delta f'(x) + \frac{\delta^2}{2!} f''(\xi), \xi \in [x, x+\delta]$. Use this expression to make an algorithm to approximate $\cos(x+\delta) - \cos(x)$. Explain why you chose the algorithm. Next, compare the approximation from your algorithm with part (b) with the same values for x and δ.

- Problem: Developing an algorithm for this problem was quite a challenge as the ξ was not defined to minimize error. Thus, an iterative method was needed to test different values between x and $x+\delta$. This was done by creating a vector of fractions (between 0 and 1) to multiply with delta, using the term $x + frac * \delta$, and substituting that in for ξ. The loop used for this algorithm is pictured below (with the rest of the code in the appendix).

```
# Iterating most accurate ksi for part (c)
# Test for x1, iterating through fractions of delta to determine a ksi value that
# minimizes error
fx1_c = np.zeros(17)
del_fracs = np.linspace(0,1,100000)
print(del_fracs)
for d in range(17):
    ksi1 = x1+(del_vec[d]*del_fracs)
    fx1_c_test = -(del_vec[d] * np.sin(x1)) - ((del_vec[d]**2/2) * np.cos(ksi1))
    err = abs(fx1_b[d] - fx1_c_test)
    #print(err)
    min_arg1 = np.argmin(err)
    #print(min_arg1)
    fx1_c[d] = -(del_vec[d] * np.sin(x1)) - ((del_vec[d]**2/2) * np.cos(ksi1[min_arg1]))

print(abs(fx1_c-fx1_b))
```

Figure 9: Problem 5.c - For Loop used for ξ Algorithm

These, as expected, varied with each δ but resulted in values similar to that in parts a) and b). The absolute error comparisons of the two methods is pictured below. As the errors were incredibly small, yet grew with delta, two figures are provided. One with all delta terms, and second where the first few are removed to show the small magnitude of the errors.

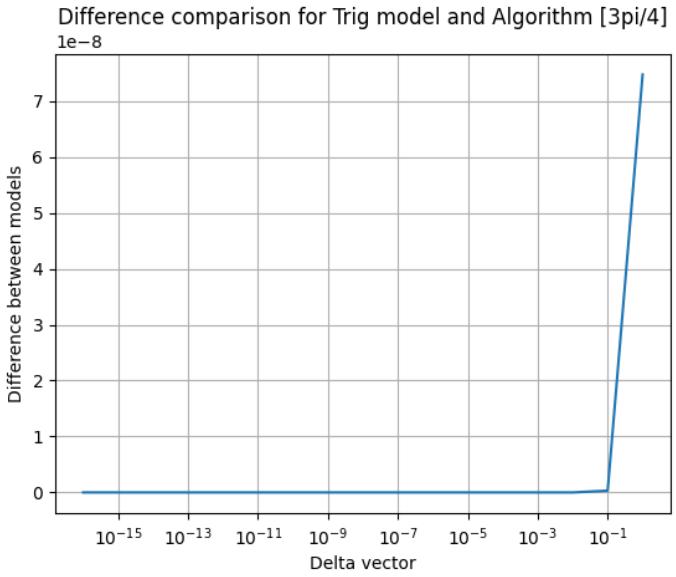

Figure 10: Problem 5.c - Full Error Plot of Algorithm for $x_1 = \frac{3\pi}{4}$

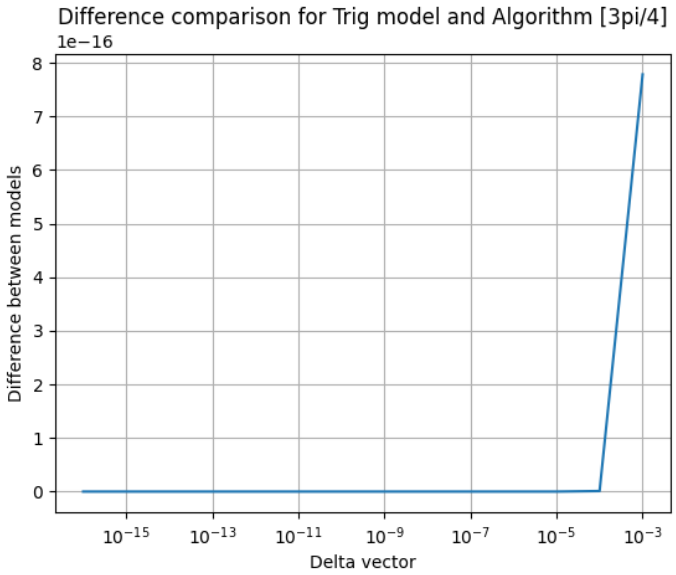

Figure 11: Problem 5.c - Condensed Error Plot of Algorithm for $x_1 = \frac{3\pi}{4}$

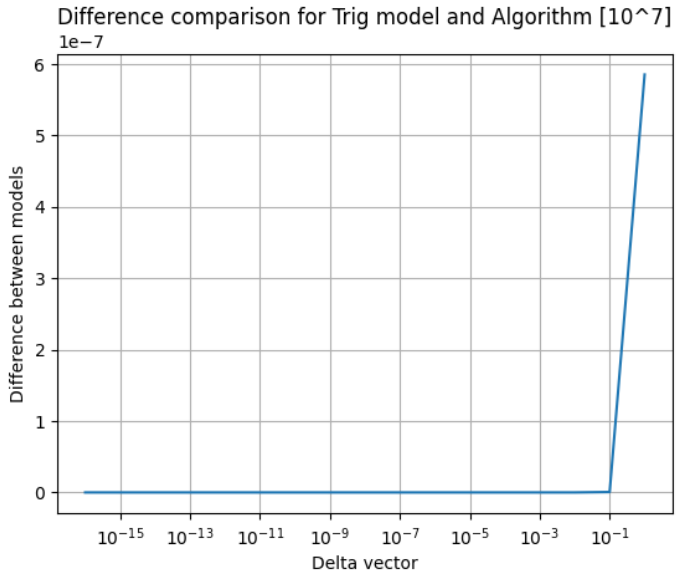

Figure 12: Problem 5.c - Full Error Plot of Algorithm for $x_2 = 10^7$

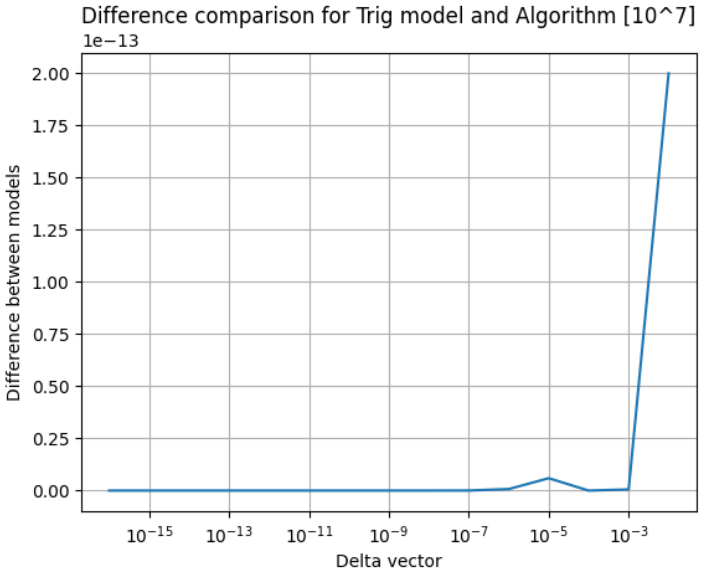

Figure 13: Problem 5.c - Condensed Error Plot of Algorithm for $x_2 = 10^7$

- Discussion: These error plots, given the crude nature of the algorithm, were honestly quite surprising. The Taylor expansion formula we were given contains a δ^2 term which caused the error to increase as the deltas grew, however (for both x_1 and x_2) while errors were not within the same order of magnitude, the were relatively small (staying below 10^{-7})) which means that the algorithm chose ξs that maximized accuracy.

II. Appendix

1. Code

```
import numpy as np
import matplotlib.pyplot as plt
#The following code is done written within this file and then
#run in the terminal


# PROBLEM 1

x = np.arange(1.920, 2.080,0.001)
p1 = x**9 - 18*(x**8) + 144*(x**7) - 672*(x**6) + 2016*(x**5)...
- 4032*(x**4) + 5376*(x**3) - 4608*(x**2) + 2304*x - 512
plt.figure(1)
plt.plot(x,p1)
plt.xlabel("X Points Given")
plt.ylabel("P Points")
plt.title("p(x) Plotted in Coefficient Form")
plt.show()

p2 = (x-2)**9
plt.figure(2)
plt.plot(x,p2)
plt.xlabel("X Points Given")
plt.ylabel("P Points")
plt.title("p(x) Plotted in Original Binomial Form")
plt.show()

# PROBLEM 2
#Failed attempt to manually determine the error
'''x = np.linspace(1.99999999,2.00000001,1000000)
y = np.linspace(0.5,2.5,1000000)
diff = np.abs(x-y)
z1_bad = np.sin(x) - np.sin(y)
z1_good = 2 * np.cos((x+y)/2) * np.sin((x-y)/2)
plt.figure(3)
plt.plot(diff,z1_bad)
plt.plot(diff,z1_good)
plt.title("Error Determination Attempt - Failed")
plt.show()'''

# PROBLEM 4
# Using python  to find roots to a precision higher than 3 decimal places
```

```
# x^2-56x+1=0
a = 1
b = -56
c = 1

descrim = b**2 - 4*a*c
r1 = (-b + np.sqrt(b**2 - 4*a*c)) / (2 * a)
r2 = (-b - np.sqrt(b**2 - 4*a*c)) / (2 * a)

print("r1 is ", r1)
print("r2 is ", r2)


# PROBLEM 5
# Creating data to be turned into a table for part b) and c)
# where the error in the data will be compred between each set


x1 = (3* np.pi)/4
x2 = 10**7
pow_vec = np.arange(16,-1,-1)
del_vec = (1/10)**(pow_vec)

#print(pow_vec)
#print(del_vec)

# Original formula (a)
fx1_a = np.cos(x1+del_vec)-np.cos(x1)
fx2_a = np.cos(x2+del_vec)-np.cos(x2)

# Trig simplification (b)
fx1_b = -2 * np.sin(x1 + (del_vec / 2)) * np.sin(del_vec / 2)
fx2_b = -2 * np.sin(x2 + (del_vec / 2)) * np.sin(del_vec / 2)


# Plot comparing (a) and (b) for x1
plt.figure(4)
plt.grid(True)
plt.semilogx(del_vec,abs(fx1_a-fx1_b))
plt.xlabel("Delta vector")
plt.ylabel("Difference between models")
plt.title("Difference comparison for Original and Trig Model [3pi/4]")
plt.show()

# Plot comparing (a) and (b) for x2
```

```
plt.figure(4)
plt.grid(True)
plt.semilogx(del_vec,abs(fx2_a-fx2_b))
plt.xlabel("Delta vector")
plt.ylabel("Difference between models")
plt.title("Difference comparison for Original and Trig Model [10^7]")
plt.show()



# Iterating most accurate ksi for part (c)
# Test for x1, iterating through fractions of delta to determine a ksi value that
# minimizes error
fx1_c = np.zeros(17)
del_fracs = np.linspace(0,1,100000)
print(del_fracs)
for d in range(17):
    ksi1 = x1+(del_vec[d]*del_fracs)
    fx1_c_test = -(del_vec[d] * np.sin(x1)) - ((del_vec[d]**2/2) * np.cos(ksi1))
    err = abs(fx1_b[d] - fx1_c_test)
    #print(err)
    min_arg1 = np.argmin(err)
    #print(min_arg1)
    fx1_c[d] = -(del_vec[d] * np.sin(x1))-...
    ((del_vec[d]**2/2) * np.cos(ksi1[min_arg1]))

print(abs(fx1_c-fx1_b))

diffs1 = abs(fx1_c-fx1_b)
plt.figure(5)
plt.grid(True)
plt.semilogx(del_vec,diffs1)
plt.xlabel("Delta vector")
plt.ylabel("Difference between models")
plt.title("Difference comparison for Trig model and Algorithm [3pi/4]")
plt.show()

#Plot with first poor elements removed
plt.figure(6)
plt.grid(True)
plt.semilogx(del_vec[:-3],diffs1[:-3])
plt.xlabel("Delta vector")
plt.ylabel("Difference between models")
plt.title("Difference comparison for Trig model and Algorithm [3pi/4]")
plt.show()
```

```
fx2_c = np.zeros(17)
del_fracs = np.linspace(0,1,100000)
print(del_fracs)
for d in range(17):
    ksi2 = x2+(del_vec[d]*del_fracs)
    fx2_c_test = -(del_vec[d] * np.sin(x2)) - ((del_vec[d]**2/2) * np.cos(ksi2))
    err = abs(fx2_b[d] - fx2_c_test)
    #print(err)
    min_arg2 = np.argmin(err)
    #print(min_arg1)
    fx2_c[d] = -(del_vec[d] * np.sin(x2)) -...
    ((del_vec[d]**2/2) * np.cos(ksi2[min_arg2]))

print(abs(fx2_c-fx2_b))

diffs2 = abs(fx2_c-fx2_b)
plt.figure(7)
plt.grid(True)
plt.semilogx(del_vec,diffs2)
plt.xlabel("Delta vector")
plt.ylabel("Difference between models")
plt.title("Difference comparison for Trig model and Algorithm [10^7]")
plt.show()

#Plot with first poor elements removed
plt.figure(8)
plt.grid(True)
plt.semilogx(del_vec[:-2],diffs2[:-2])
plt.xlabel("Delta vector")
plt.ylabel("Difference between models")
plt.title("Difference comparison for Trig model and Algorithm [10^7]")
plt.show()
```