Homework 1

Blake Hamilton

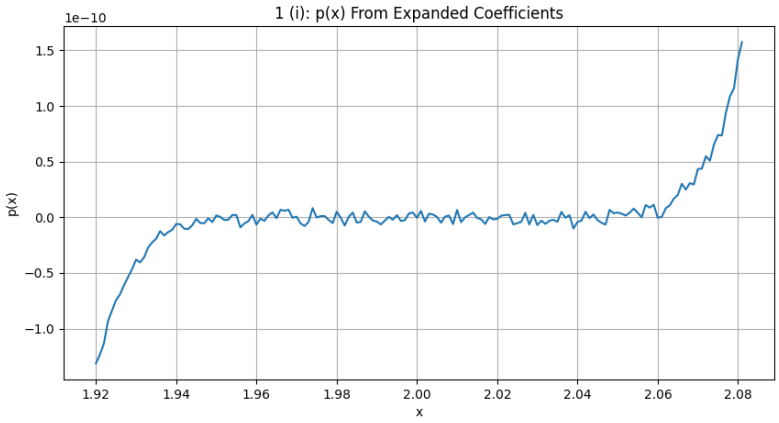
January 27, 2024

Question 1

(i)

```python
# Define the polynomial coefficients
coefficients = [1, -18, 144, -672, 2016, -4032, 5376,
    -4608, 2304, -512]

x_values = np.arange(1.920, 2.081, 0.001)
# Use numpy to create y values based on coeffs.
y_values = np.polyval(coefficients, x_values)

plt.figure(figsize=(10, 5))
plt.plot(x_values, y_values)
plt.title("1 (i): p(x) From Expanded Coefficients")
plt.xlabel("x")
plt.ylabel("p(x)")
plt.grid(True)
plt.savefig("plot_using_coefficients.png")
```

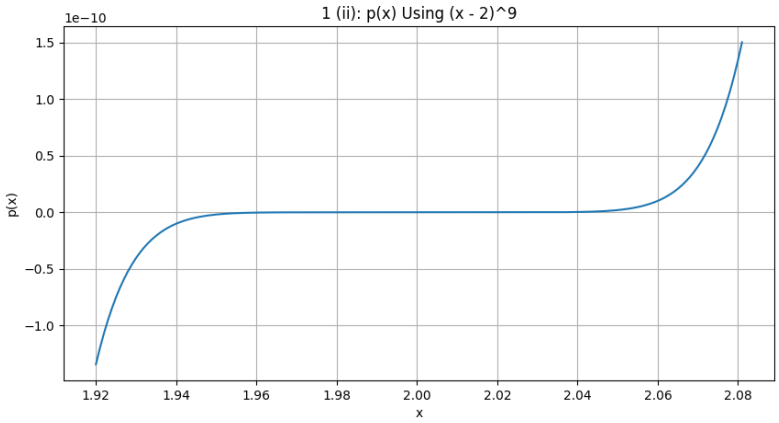
Listing 1: Polynomial Expression



(ii)

```
# Getting y vals directly from expression
y_values_alternative = (x_values - 2) ** 9

plt.figure(figsize=(10, 5))
plt.plot(x_values, y_values_alternative)
plt.title("1 (ii): p(x) Using (x - 2)^9")
plt.xlabel("x")
plt.ylabel("p(x)")
plt.grid(True)
plt.savefig("plot_using_expression.png")
```

Listing 2: $(x-2)^9$ Expression



(iii) The difference between the two graphs is that the first (using the expanded polynomial) has much more noise than the second (using $(x-2)^9$). As we have seen in class, subtraction can cause rounding errors, and since the expanded polynomial is almost entirely subtraction, this is causing rounding errors from floating point arithmetic. Since Python is using double precision values, there is maximum accuracy of 16 decimal digits, and we can visually see that with as much subtraction as is necessary, the polynomial method is vastly different from using $(x-2)^9$. Although neither plot is completely precise (as we do not have infinite precision on machines), for all intents and purposes the second plot is correct, as the round errors are negligible.

Question 2

(i) To evaluate $\sqrt{x+1}-1$ for $x \approx 0$ and avoid cancellation, I would multiply by the conjugate and then multiply. We see this gets rid of all subtraction

and is safe from cancellation issues.

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

(ii) We can evaluate $\sin(x)-\sin(y)$ for $x\approx y$ and avoid cancellation by using the trigonometric identity $\sin(a)-\sin(b)=2\sin(\frac{a-b}{2})\cos(\frac{a+b}{2})$. This way, we can use $\sin(x)-\sin(y)=2\sin(\frac{x-y}{2})\cos(\frac{x+y}{2})$. Although we still have subtraction inside of a sin term, if $x\approx y$ we know the term inside the sin is very small, and since $\sin(c)$ is stable and not susceptible to cancellation for an arbitrary $c\approx 0$, we can evaluate it without worry of cancellation.

(iii) To evaluate $\frac{1-\cos(x)}{\sin(x)}$ for $x\approx 0$ and avoid cancellation, we can multiply by the conjugate and use the identity $\sin^2(x)+\cos^2(x)=1$ to rid the expression of subtraction.

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

Question 3

We first find derivatives up to the second order of $f(x)$:

$$f(0)=1$$

$$f'(0)=(1+3x^2)\cos(x)-(1+x+x^3)\sin(x)\Big|_0=1$$

$$f''(0)=(-6x^2-2)\sin(x)+(-x^3+5x-1)\cos(x)\Big|_0=-1$$

We now find the second degree polynomial is $P_2(x)=1+x-\frac{x^2}{2}$.

(a) $P_2(0.5)=1+0.5-\frac{(0.5)^2}{2}=\frac{11}{8}=1.375$. We can find an upper bound on the error $|f(0.5)-P_2(0.5)|$ by finding the $(n+1)^{th}$ Taylor polynomial term.

$$f'''(0)=(x^3-17x+1)\sin(x)+(3-9x^2)\cos(x)\Big|_0=3$$

So, we know error $R_2(0.5)=|\frac{3\cdot(0.5)^3}{3!}|$ and the error is bounded by $R_2(0.5)=1/16=0.0625$. We then can find the actual error is $f(0.5)-P_2(0.5)\approx 1.4261-1.3750=0.0511<0.0625$, so we can confirm our actual error is within our expected error range.

(b) Following the same process as above for $x=0.5$, we can generalize for all x: the error is bounded by $|\frac{3\cdot x^3}{3!}|=|\frac{x^3}{2}|$.

(c) We can approximate $\int_0^1 f(x)\,dx$ using $\int_0^1 P_2(x)\,dx$ as follows:

$$\int_0^1 P_2(x)dx = \int_0^1 1 + x - \frac{x^2}{2}dx = x + \frac{x^2}{2} - \frac{x^3}{6}\Big|_0^1 = \frac{4}{3}$$

(d) We now estimate the error in the integral by finding $\int_0^1 R_2$. We know $\frac{x^3}{2}$ is strictly positive in $0 \leq x < 1$, we can integrate it without taking the absolute value.

$$\int_0^1 R_2 dx = \int_0^1 \frac{x^3}{2}dx = \frac{x^4}{8}\Big|_0^1 = \frac{1}{8}$$

Question 4

Considering the quadratic equation $ax^2+bx+c=0$ with $a=1$, $b=-56$, $c=1$:

(a)

$$r_{1,2} = \frac{56 \pm \sqrt{(-56)^2 - 4(1)(1)}}{2(1)} = \frac{56 \pm 55.964}{2}$$

$$r_1 = 55.982, r_2 = 0.018$$

Relative error calculation using $RE = \frac{|x-fl(x)|}{|x|}$:

$$\text{Relative error for } r_1 : \frac{|55.98213716 - 55.982|}{55.98213716} \approx 2.4 \cdot 10^{-6}$$

$$\text{Relative error for } r_2 : \frac{|0.01786284 - 0.018|}{0.01786284} \approx 7.7 \cdot 10^{-3}$$

We can see that we have one "good" root (r_1) and one "bad" root (r_2) based on the relative errors.

(b) The "bad" root is found when we subtract the root term from b. The two relations mentioned are Vieta's formulas. We can derive those formulas doing the following:

$$ax^2 + bx + c = 0 \Rightarrow x^2 + \frac{b}{a}x + \frac{c}{a} = 0$$

Since the x term comes from $r_1 + r_2$, we know $r_1 + r_2 = \frac{b}{a}$ and since the constant term comes from $r_1 r_2$, we know $r_1 r_2 = \frac{c}{a}$. First we will try the addition relation using our "good" r_1 to find an alternate to r_2, r_2':

$$r_1 + r_2 = \frac{b}{a} \Rightarrow 55.982 + r_2' = \frac{-56}{1} \Rightarrow r_2' = 0.018$$

We can see that this r_2' is the same as found in (a), so it is not a better result. This is because it requires subtraction that gives cancellation errors. We now try again with the multiplication relation:

$$r_1 r_2 = \frac{c}{a} \Rightarrow 55.982 r_2'' = \frac{1}{1} \Rightarrow r_2'' = 0.017863$$

We see that this gives a more precise answer, since we can retain the five decimal digits of precision from 55.982 in the division operation. We see below that this relative error is much smaller than part (a), meaning we found the root more accurately:

$$\text{Relative error for } r_2'' : \frac{|0.01786284 - 0.017863|}{0.01786284} \approx 9.0 \cdot 10^{-6}$$

Question 5

Playing with different values of x for $y = x_1 - x_2$ with $\tilde{x}_1 = x_1 + \Delta x_1$ and $\tilde{x}_2 = x_2 + \Delta x_2$:

```
# FUnction to compute the difference between two numbers,
    given their errors
def compute_difference(x1, x2, delta_x1, delta_x2):
    # Exact computation
    y_exact = x1 - x2

    # Approximations
    x1_hat = x1 + delta_x1
    x2_hat = x2 + delta_x2
    y_approx = x1_hat - x2_hat

    # Error from approximation
    error = (delta_x1 - delta_x2)

    return y_exact, y_approx, error

# playing with small and large values
x_small = 0.01
x_large = 1e6

delta_x1 = 0.001
delta_x2 = 0.001

print(compute_difference(x_small, x_small, delta_x1,
    delta_x2))
print(compute_difference(x_large, x_large, delta_x1,
    delta_x2))
```

Listing 3: Subtraction With Errors

(a) We find the absolute error below:

$$\begin{aligned}
|\Delta y| = |y - \widetilde{y}| &= |x_1 - x_2 - (\widetilde{x_1} - \widetilde{x_2})| && (1)\\
&= |x_1 - x_2 - x_1 - \Delta x_1 + x_2 + \Delta x_2| && (2)\\
&= |\Delta x_2 - \Delta x_1| && (3)\\
&\leq |\Delta x_1| + |\Delta x_2| \text{From Triangle Equality} && (4)
\end{aligned}$$

We now find relative error:

$$\frac{|\Delta y|}{|y|} = \frac{|y - \widetilde{y}|}{|y - \Delta y|} \tag{5}$$

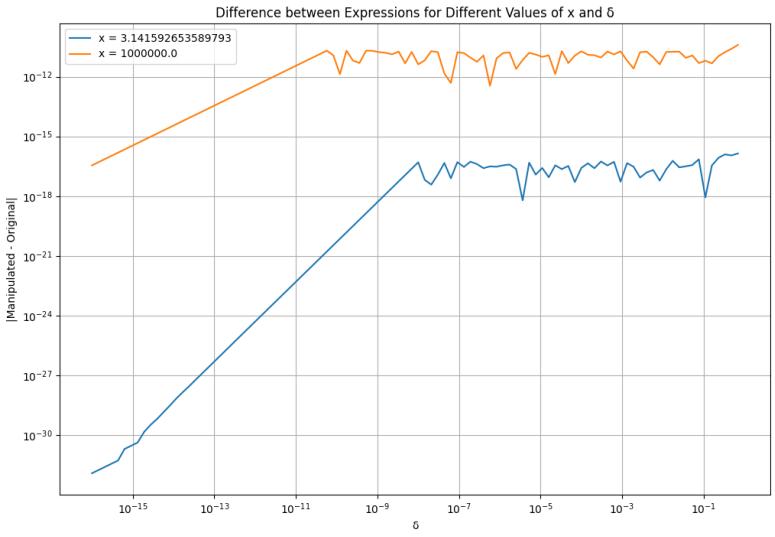
$$= \frac{|\Delta x_2 - \Delta x_1|}{|\widetilde{x_1} - \widetilde{x_2} - \Delta x_1 + \Delta x_2|} \tag{6}$$

$$\leq \frac{|\Delta x_1| + |\Delta x_2|}{|\widetilde{x_1} - \Delta x_1| + |\widetilde{x_2} + \Delta x_2|} \tag{7}$$

$$\tag{8}$$

The upper bound on the absolute error $|\Delta y$ is $|\Delta x_1| + |\Delta x_2|$. We then use this to find the upper bound on the relative error: $\frac{|\Delta x_1| + |\Delta x_2|}{|\widetilde{x_1} - \Delta x_1| + |\widetilde{x_2} + \Delta x_2|}$. The relative error is large when the absolute error is large or the difference between the x values is small. This makes sense, as an absolute error makes more of a difference when looking at smaller numbers than bigger numbers.

(b) Using $\cos(a) - \cos(b) = -2\sin(\frac{a+b}{2})\sin(\frac{a-b}{2})$, we can manipulate the expression to be $-2\sin(x + \frac{\delta}{2})\sin(\frac{\delta}{2})$.

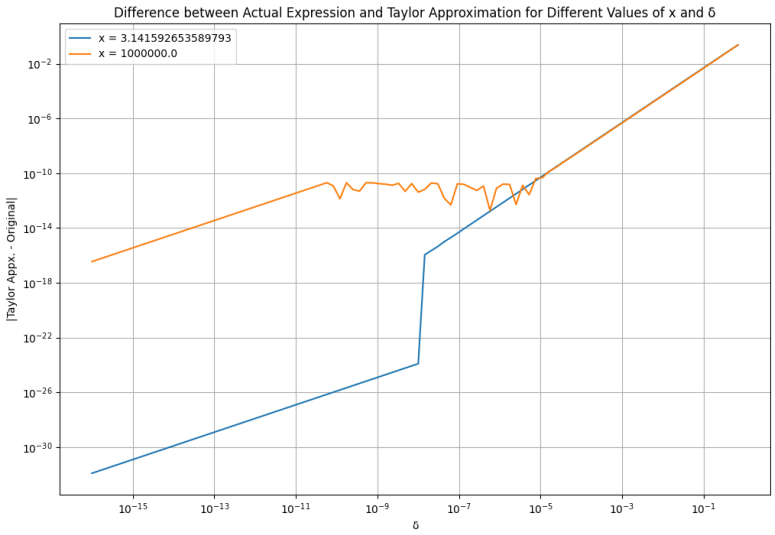


We can see in the graphic above that the difference between the two expression versions increases as we increases δ for both, but at some point tops out. As expected, the difference for the larger x value is greater than the difference for the smaller x value.

```
# Function to compute the manipulated expression:
# -2*sin(x + delta/2)*sin(delta/2)
def manipulated_expression(x, delta):
    return -2 * np.sin(x + delta/2) * np.sin(delta/2)

# Function to compute cos(x + delta) - cos(x)
def original_expression(x, delta):
    return np.cos(x + delta) - np.cos(x)

# Values of x
x_values = [np.pi, 1e6]

# delta between 10^{-16} to 1 using logspace
delta_values = np.logspace(-16, 1, endpoint=False,
    num=100)

# Plotting
plt.figure(figsize=(12, 8))

# Calculating the differences
for x in x_values:
    differences = np.abs(manipulated_expression(x,
        delta_values) - \
    original_expression(x, delta_values))
    plt.loglog(delta_values, differences, label=f'x =
        {x}')
```

Listing 4: Manipulated Expression Comparison

(c) Using the Taylor expansion provided, a new algorithm can be constructed (using $f(x) = \cos(x)$): $-\delta \cos(x)$. We also have an error term: $-\frac{\delta^2}{2} cos(x)$ with $\xi \in [x, x + \delta]$, but this should not be included in our approximation. This is chosen because we can substitute in our $f(x)$ to the Taylor approximation to see how well this works.



We can see that the errors are much larger using this approximation than the alternate form in part (b) does. This is expected, as the form in part (b) is analytically equivalent, but this is just an approximation.

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

# Function to compute the Taylor expansion approximation: delta*f'(x) + (delta)^2/(2!)f''(x)
def taylor_approximation(x, delta):
    # Since xi is somewhere in [x, x+delta], we will pick x+(delta/2) for the second derivative
    return delta * -np.sin(x) #+ (delta**2) / 2 * (-np.cos(x + delta/2))


# Function to compute cos(x + delta) - cos(x)
def original_expression(x, delta):
    return np.cos(x + delta) - np.cos(x)

# Values of x
x_values = [np.pi, 1e6]

# delta between 10^{-16} to 1 using logspace
delta_values = np.logspace(-16, 0, endpoint=False, num=100, base=10)

# Plotting
plt.figure(figsize=(12, 8))
```

```
22
23 # Calculating the differences
24 for x in x_values:
25     differences = np.abs(taylor_approximation(x,
           delta_values) - original_expression(x,
           delta_values))
26     plt.loglog(delta_values, differences, label=f'x =
           {x}')
```

Listing 5: Taylor Approximation for Problem 5