Assignment 1
FIT3139

Daniel Nguyen
32471033

Table of Contents

`

Preface and Assumptions

All referenced code is contained in the labelled appendix.

Part 3:
I am assuming that both sources of error are evaluated together at the same time. I have taken this information from comments made by the teaching team on the edstem forum (In particular post #74)

Part 1 – Model Error

1.1

A Taylor Series of a function centred at point x = 0 is infact a Maclaurin Series.

Due to the nature of the Maclaurin Series of cos(x), every odd term after the first (3rd, 5th etc.) is equal to 0, therefore the number of terms must be increased by 2 to see a difference in the Maclaurin Series of cos(x)

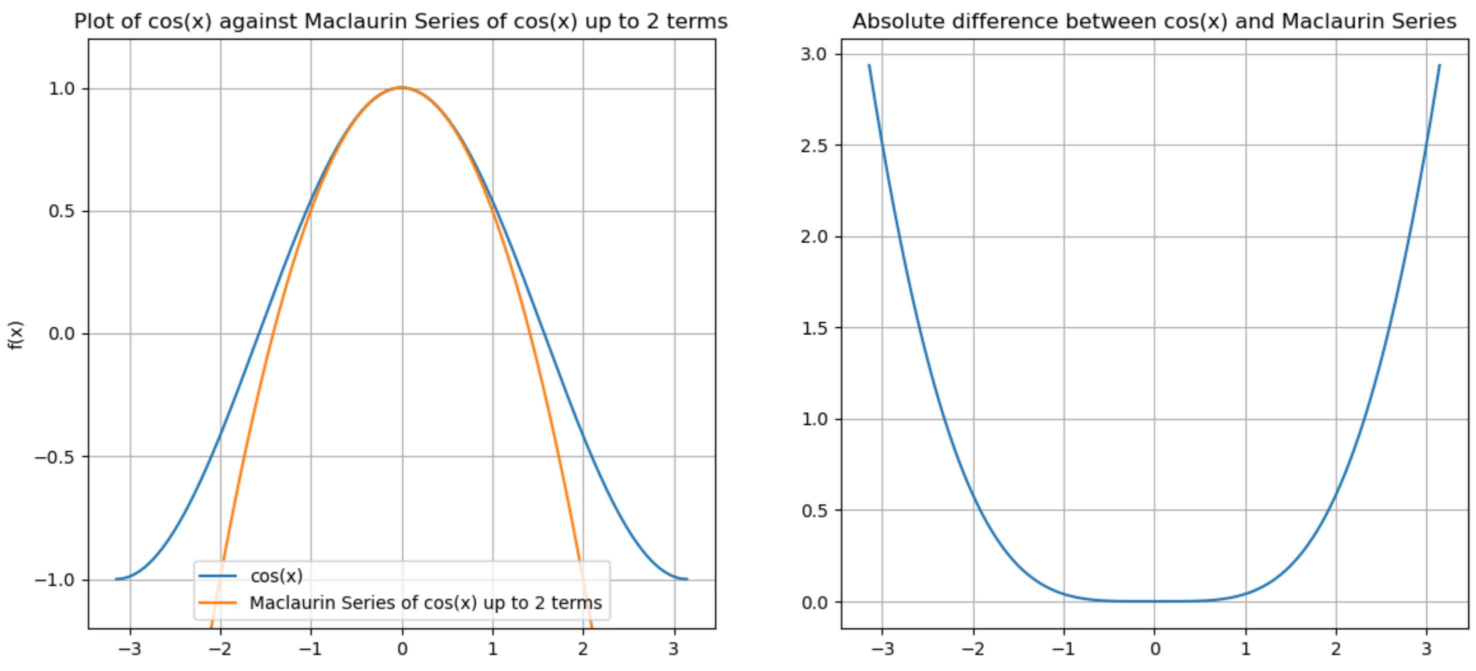

Fig. 1

With a Maclaurin series of 2 terms, we can see that the error is 0 at the centring point, and this error increases as we move away from the centre.

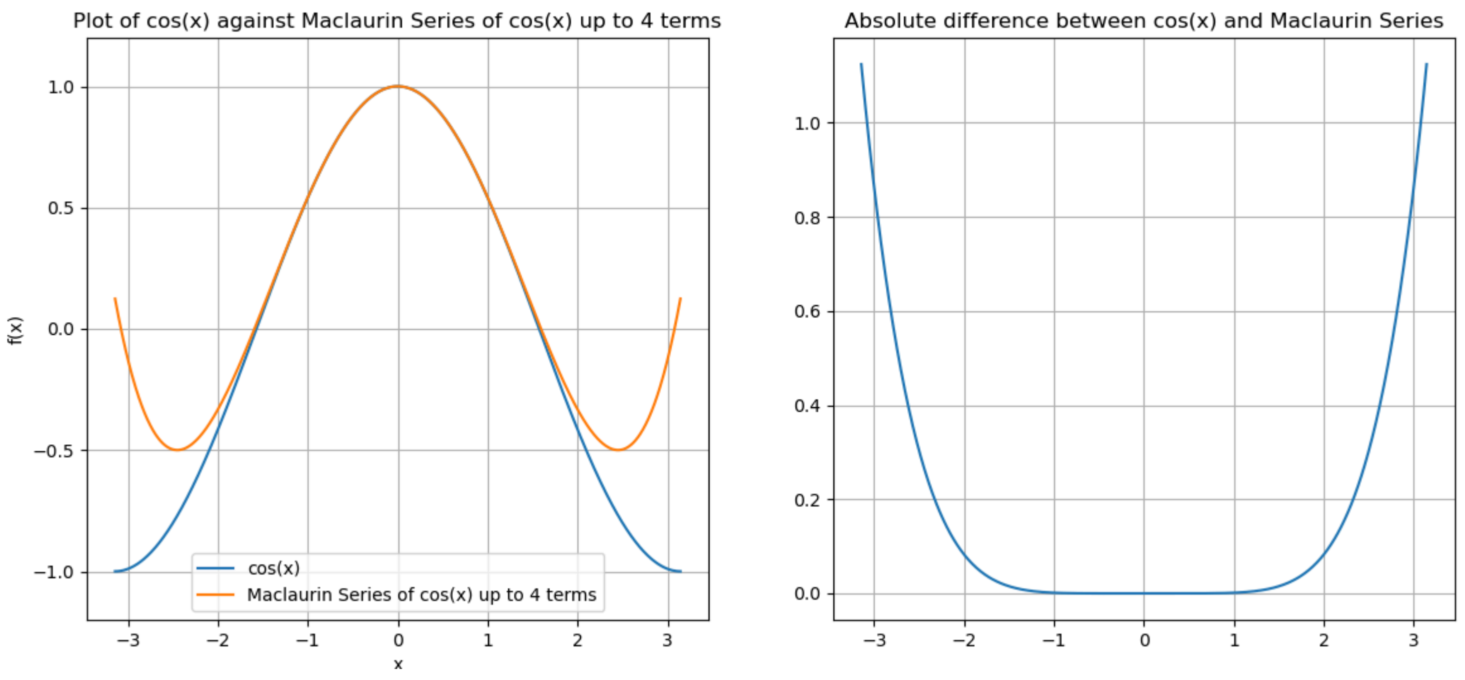

Fig 2.

With the calculation of additional terms in the Maclaurin Series, we can see that the error is smaller as we move away from the centring point, however, our approximation model still begins to fail the further away we are.

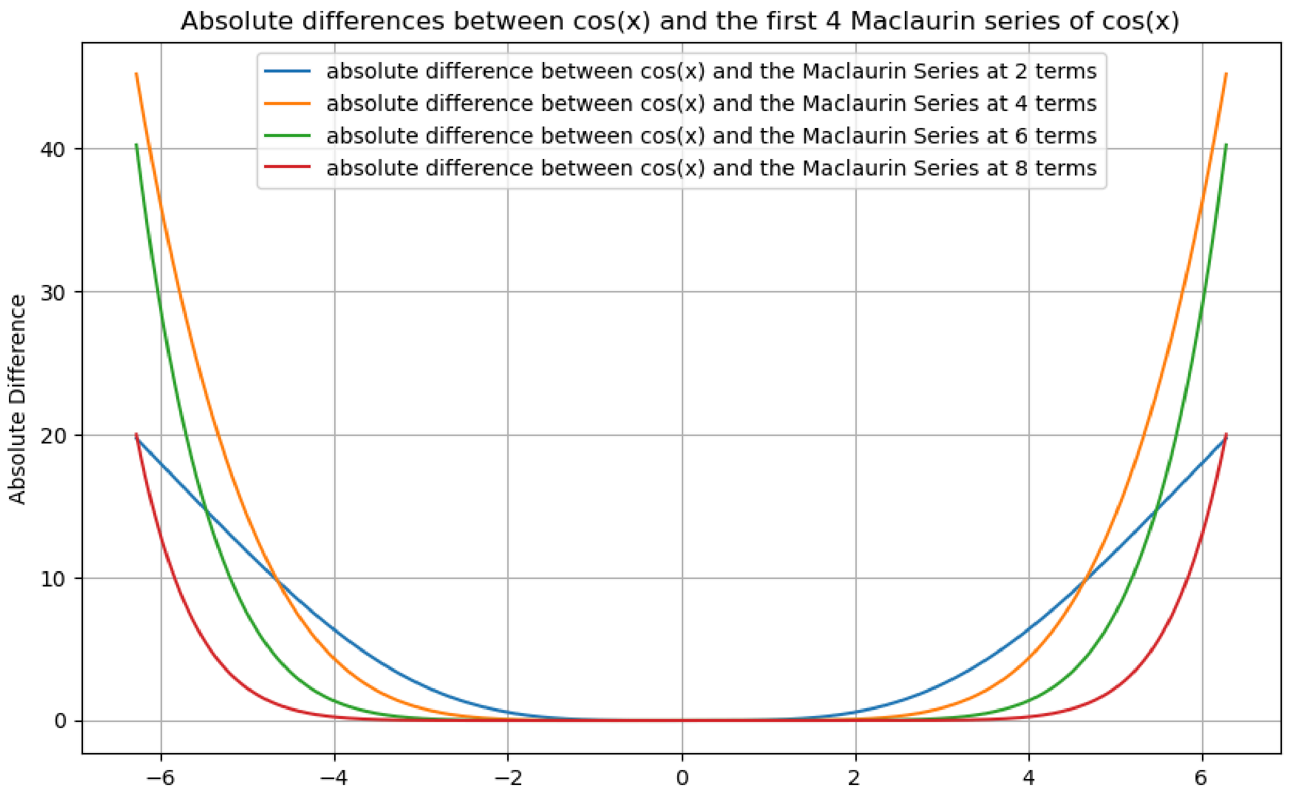

Fig 3.

Looking at the absolute differences for various increasingly accurate Maclaurin Series, we can see that for each increase in number of terms, the model gets increasingly more accurate than the previous iteration. Each further term added to the Maclaurin series allows a higher level of accuracy in approximating cos(x) further away from the centring point, until the approximation fails and the absolute difference endlessly increases.

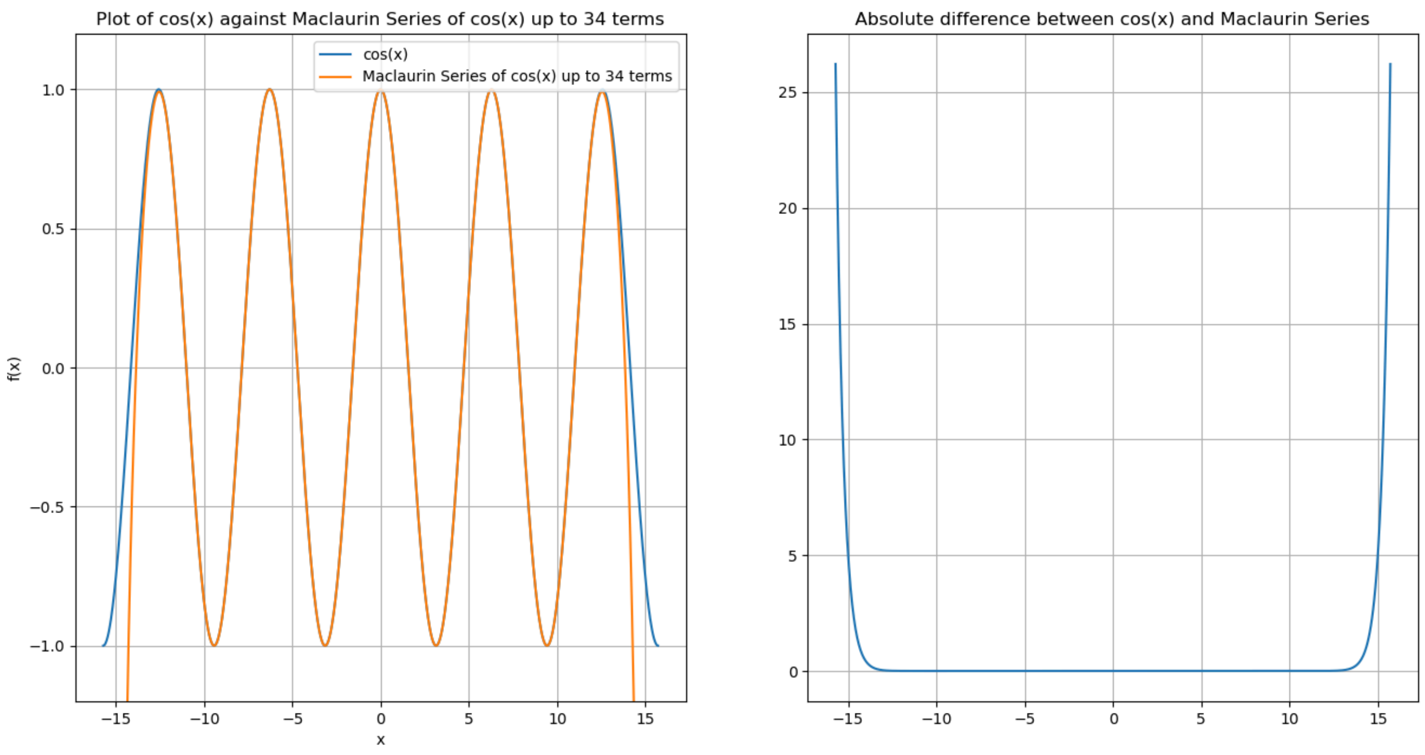

Fig 4.

As we continue to refine the accuracy of our Maclaurin Series, we see that the approximation follows the original function even further away from the centring point. In figure 4, we see that the approximation is accurate for multiple periods to the negative and positive x directions.

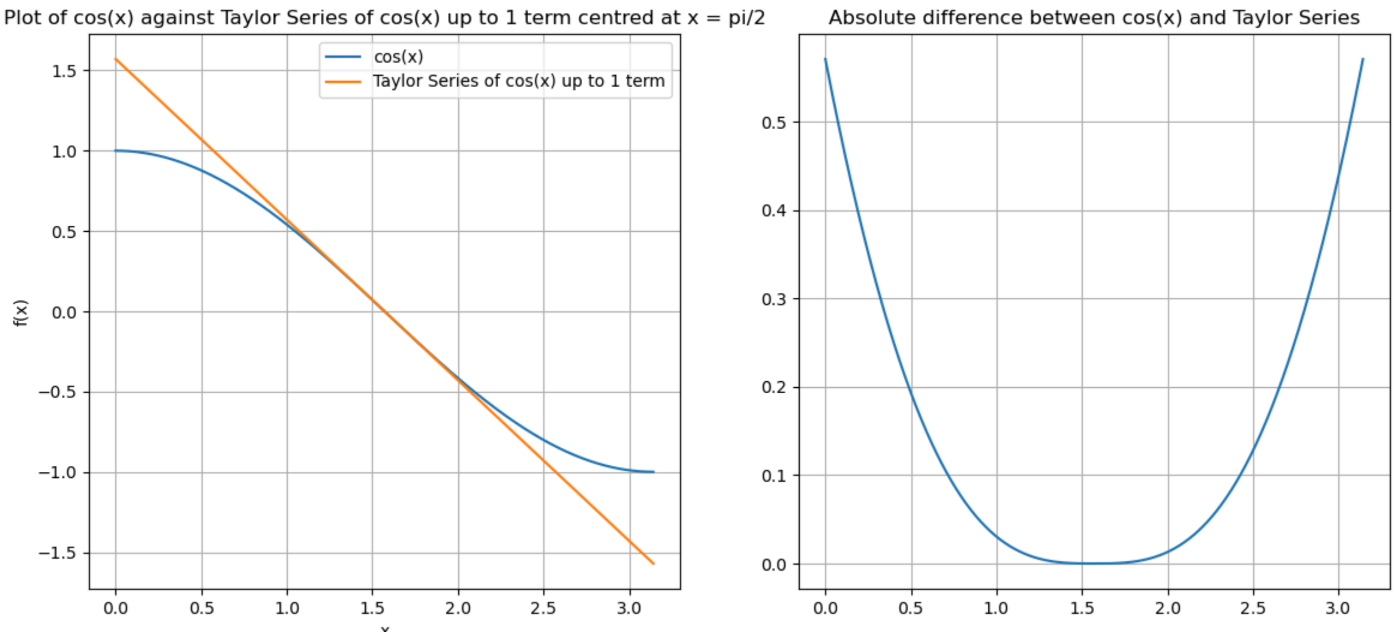

Fig. 5

When plotting the Taylor series of cos(x), centred at x = pi/2, we see that the first usable Taylor Series changes compared to when centred at x = 0. Now, every even term is usable, instead of every odd term. As before, the approximation is accurate around the centring point, but falls apart the further away we travel.

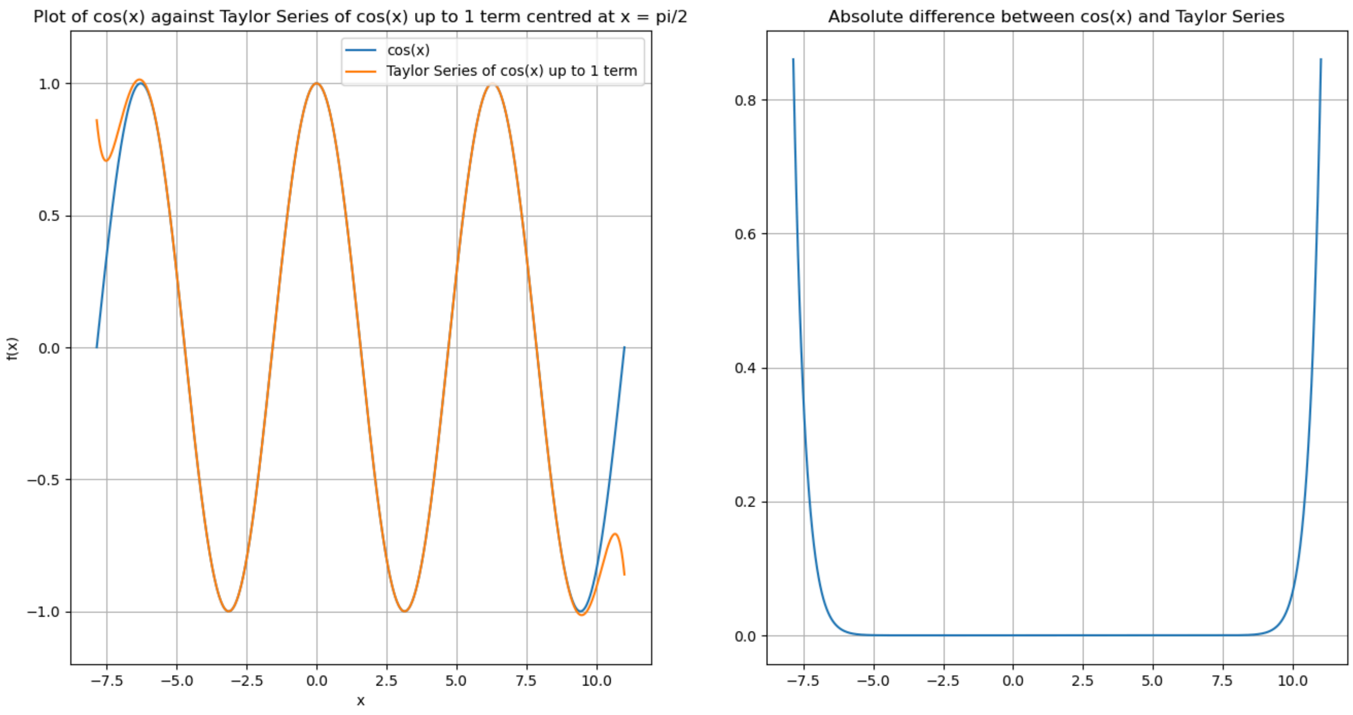

Fig 6.

We see, once again, that further refinements to our Taylor Series yields not only increased accuracy nearer to the centring point, but also an extension of where the approximation is accurate further outwards, until eventually the error rises indefinitely where our model fails, as it does not have enough terms to continue producing turning points in its plot.

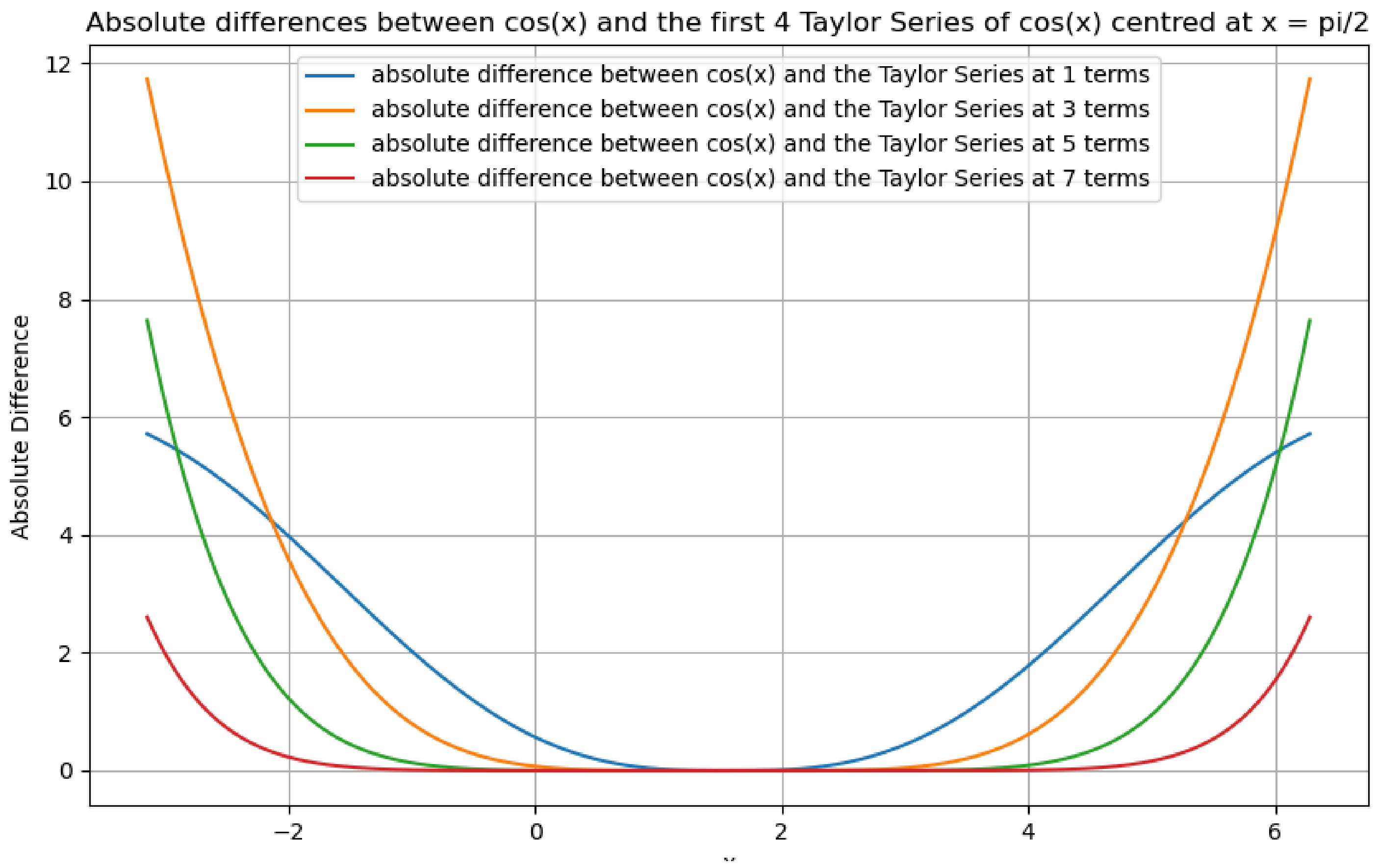

Fig 7.

We can see again that as we calculate more terms for the Taylor Series, the accuracy closer to the centring point is increased, in addition to increased accuracy further away from the centring point until the approximation fails, where the absolute difference between the 2 functions steeply rises.

1.2

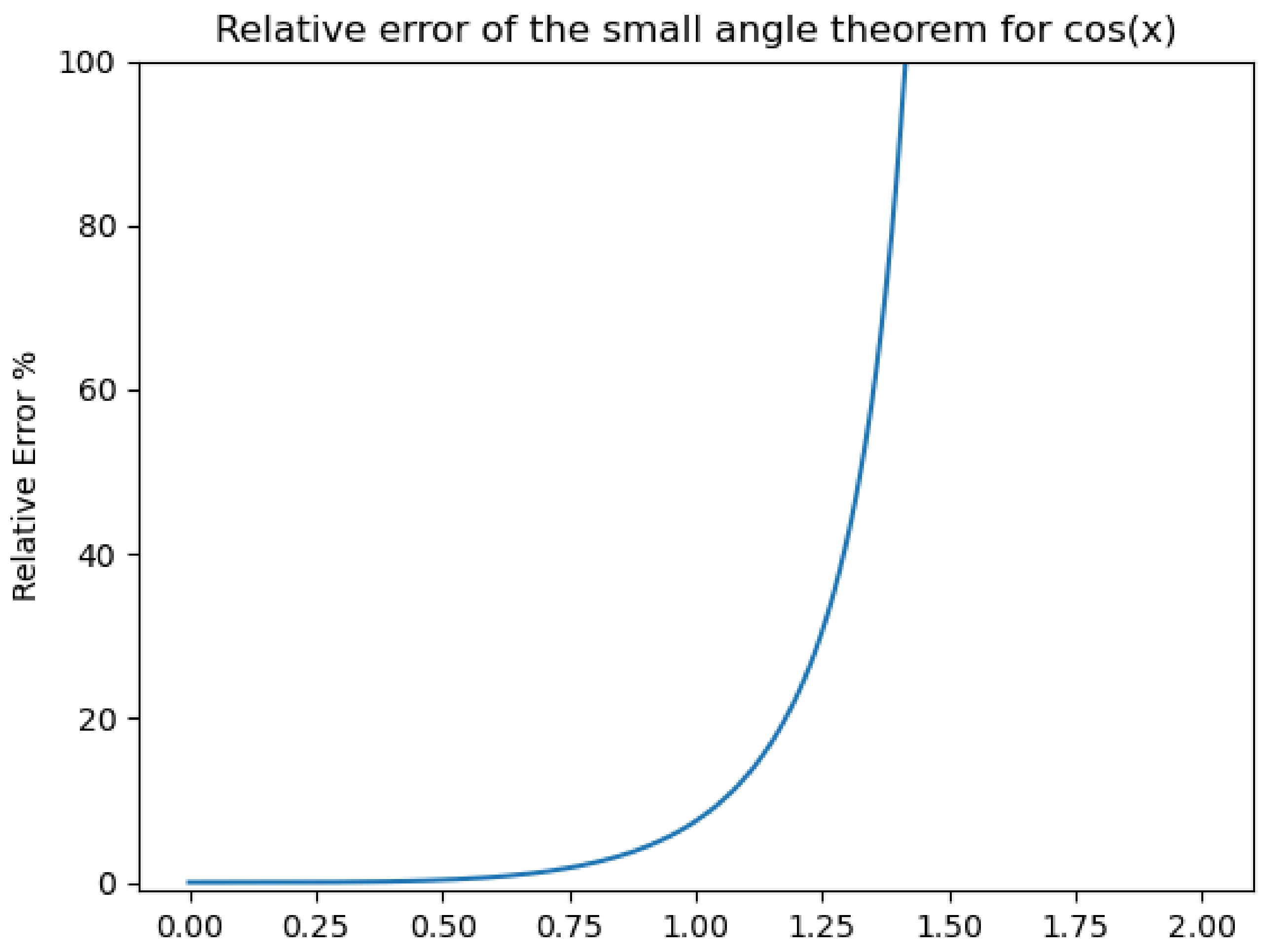

Fig 8.

The relative error percentage is calculated as:

((Actual Value - Calculated Value) / Actual Value) * 100

We can see that as we move away from 0, the relative error rapidly increases, both due to our model only working at values close to 0, and the fact that as cos(x) approaches 0, the denominator in our relative error calculations also approaches 0, increasing the magnitude of any error.

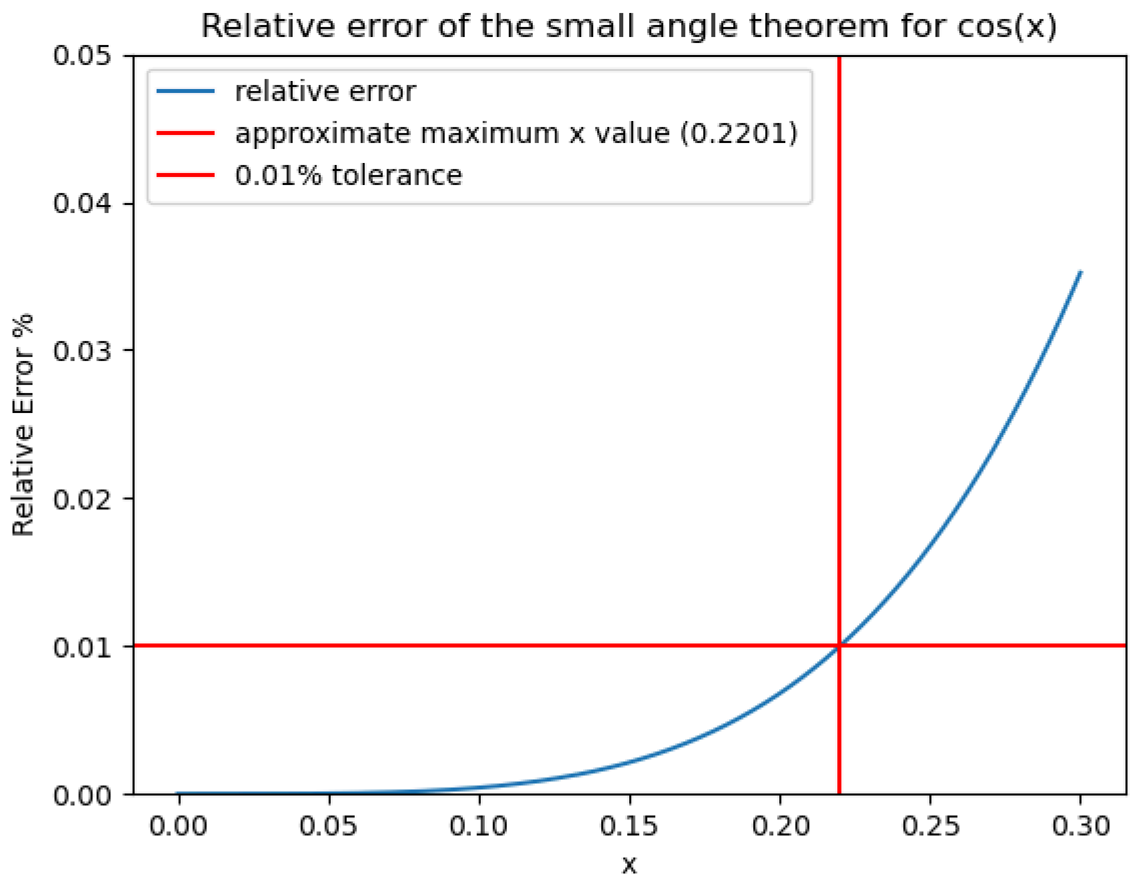

Fig 9.

In python, we cannot find the exact value where the relative error is less than 0.01%, however we can find a good approximation of this value. By creating a function to scan many x values (see smallAngleBoundary in appendix) until we find one where the error is larger than 0.01%, we can see that once x passes 0.2201 approaching 0, the error drops below 0.01%.

1.3

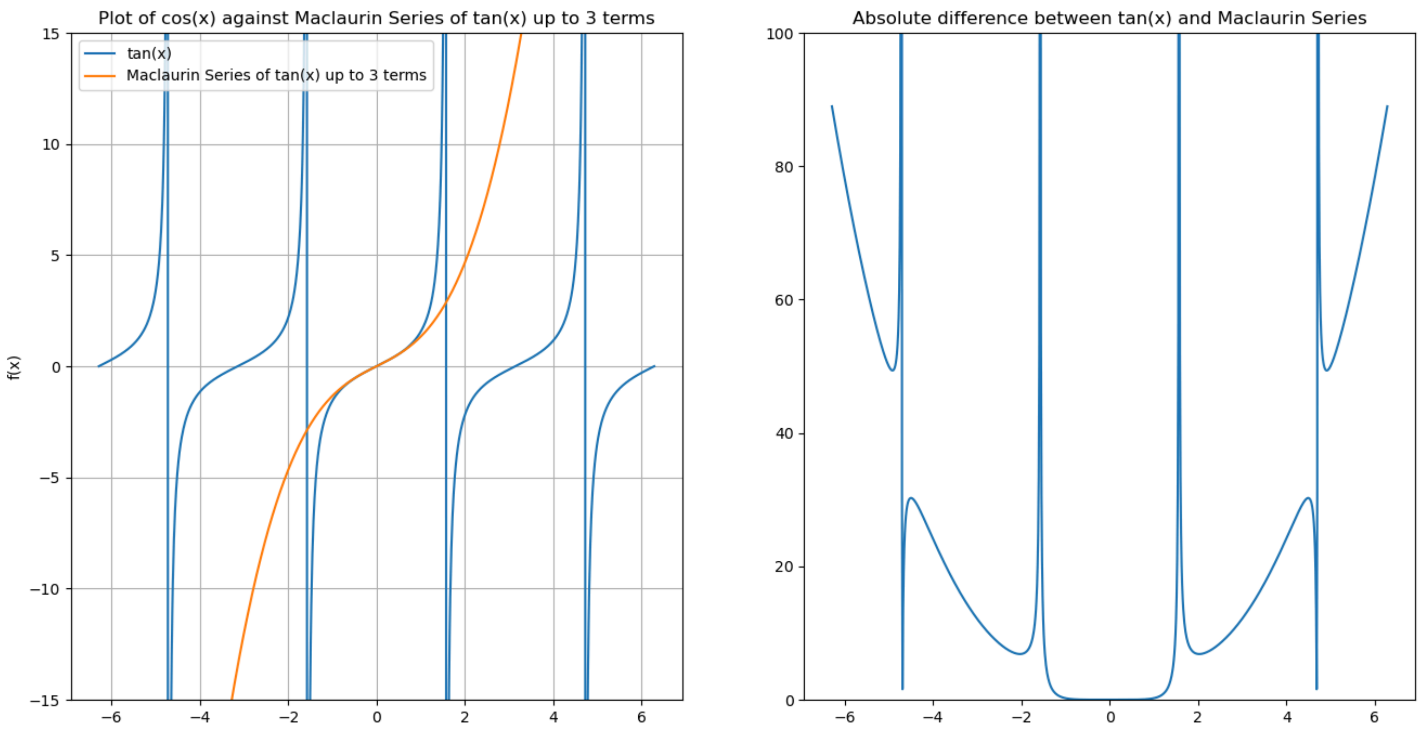

Fig 10.

With a Maclaurin series centered at x = 0, we can see that our approximation follows the original curve close to the centring point, but fails further away, especially after the central period in either x direction. The plot of the absolute difference between each curve is interesting, as it looking at it in isolation seems to imply that there are points where our approximation is accurate again further away from x = 0, but looking at both plots as a whole, we can see that this is not true.

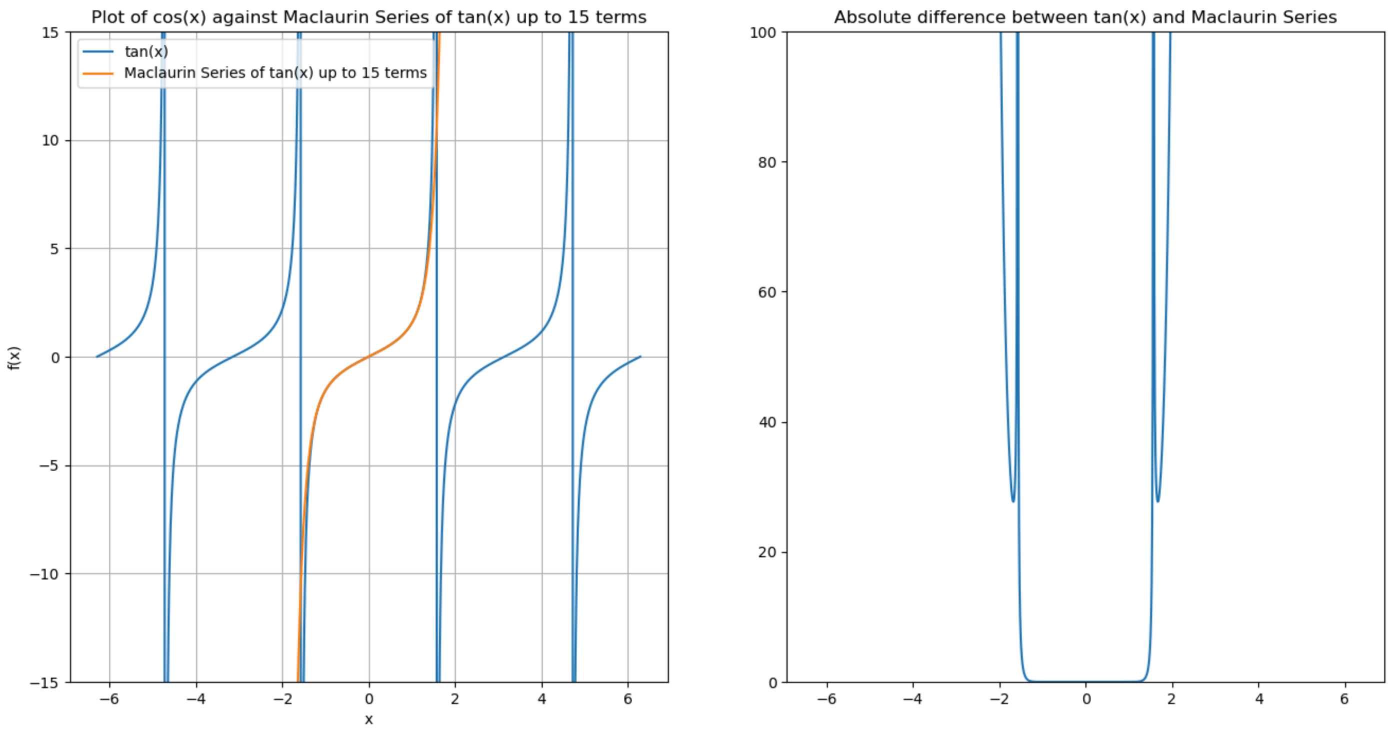

Fig 11.

When we add more terms to the Maclaurin series, we gain increased accuracy for the central period, however, unlike our cos(x) approximations, this accuracy does not contine past this. We have greatly increased accuracy near the centring point, but our approximation does not work past the asymptotes of tan(x)

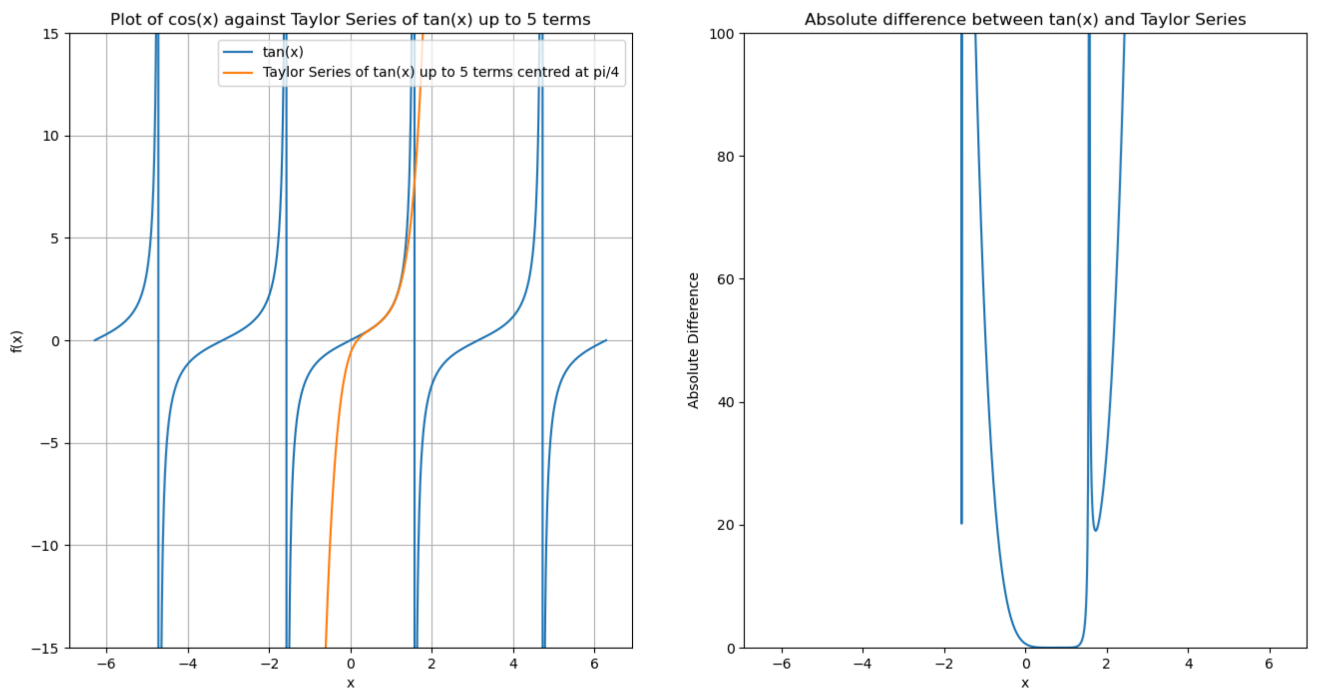

Fig 12

When centring the taylor series at another point, such as pi/4 in figure 12, we see that again, near the centring point, the approximation is accurate, however, as we move away, the error climbs rapidly as our approximation fails.

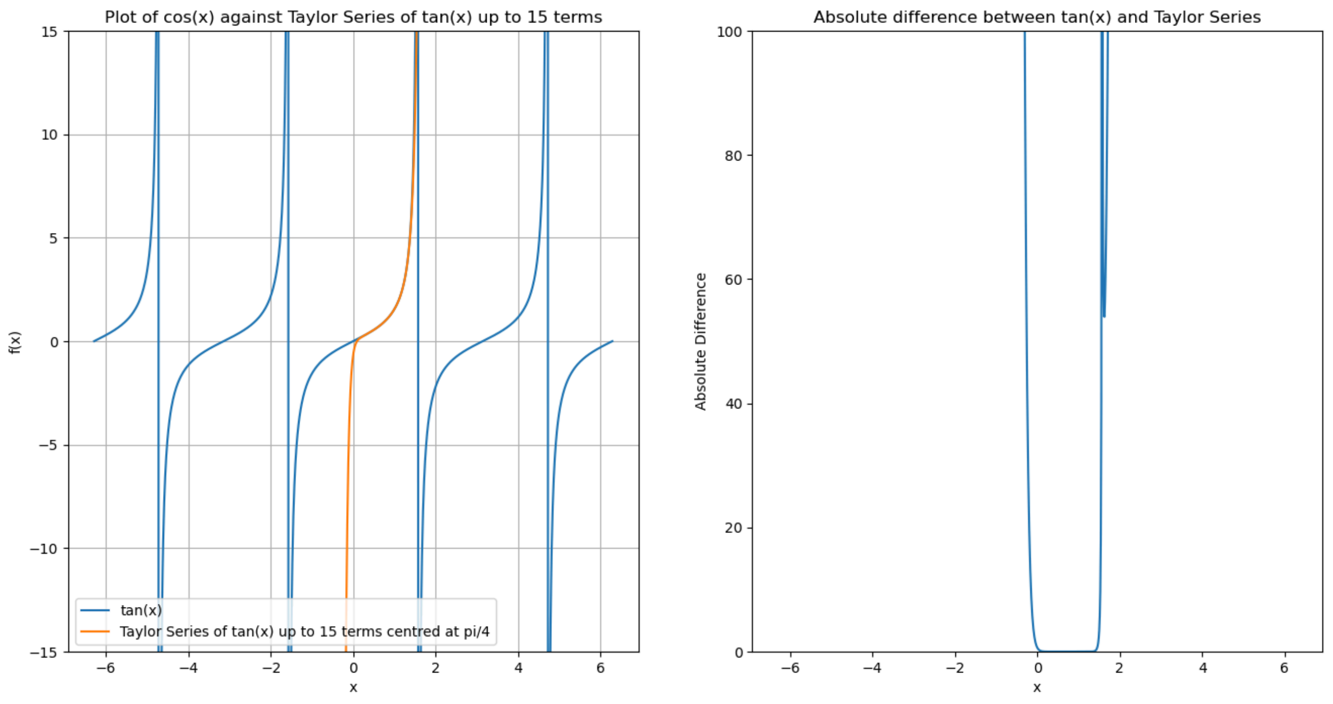

Fig 13.

When plotting the Taylor Series with more terms against tan(x), surprisingly, there is also a great loss of accuracy after x = 0 in the negative x direction, however accuracy is increased towards the first asymptote at x = pi/2.

Part 2 – Data Error

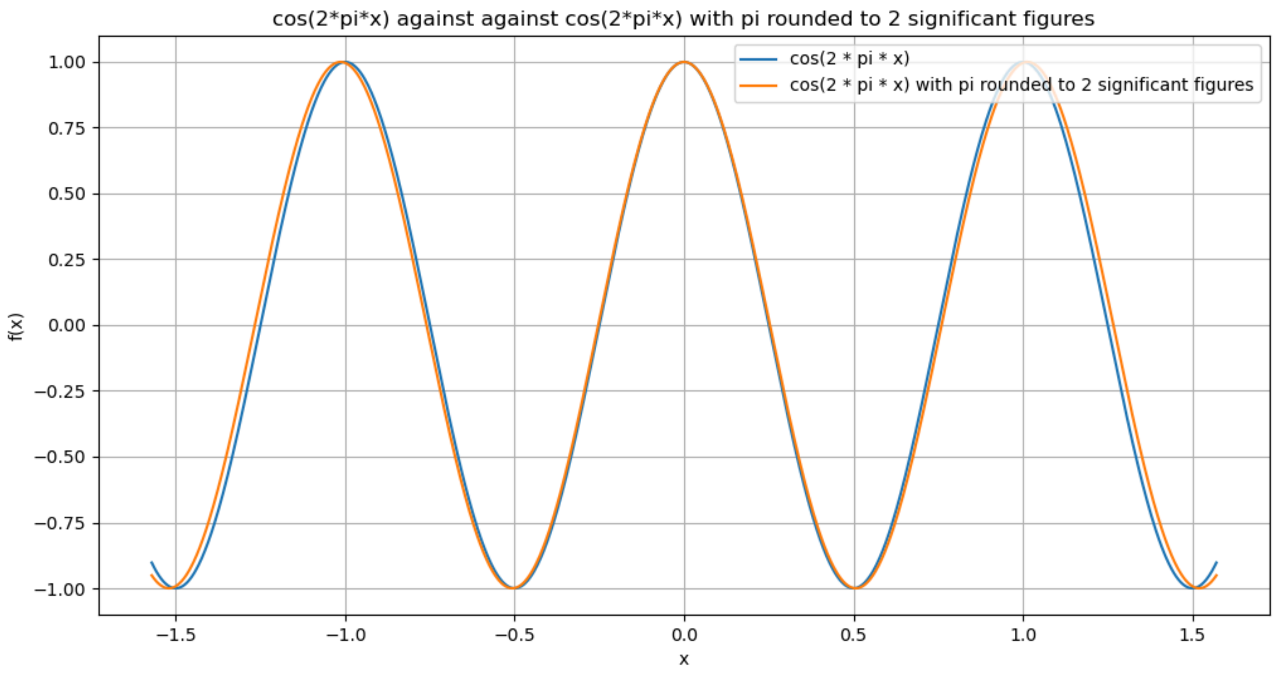

Fig 14.

Comparing the cos function against the same with pi rounded to 2 significant figures, we observe that the curves become further out of sync as we move further away from x = 0. This makes sense as rounding pi is equivalent to applying a coefficient to a perfectly accurate pi, which would affect the period of the curve. The period of our rounded pi curve is slightly longer, as pi rounded to 2 significant figures is less than pi.

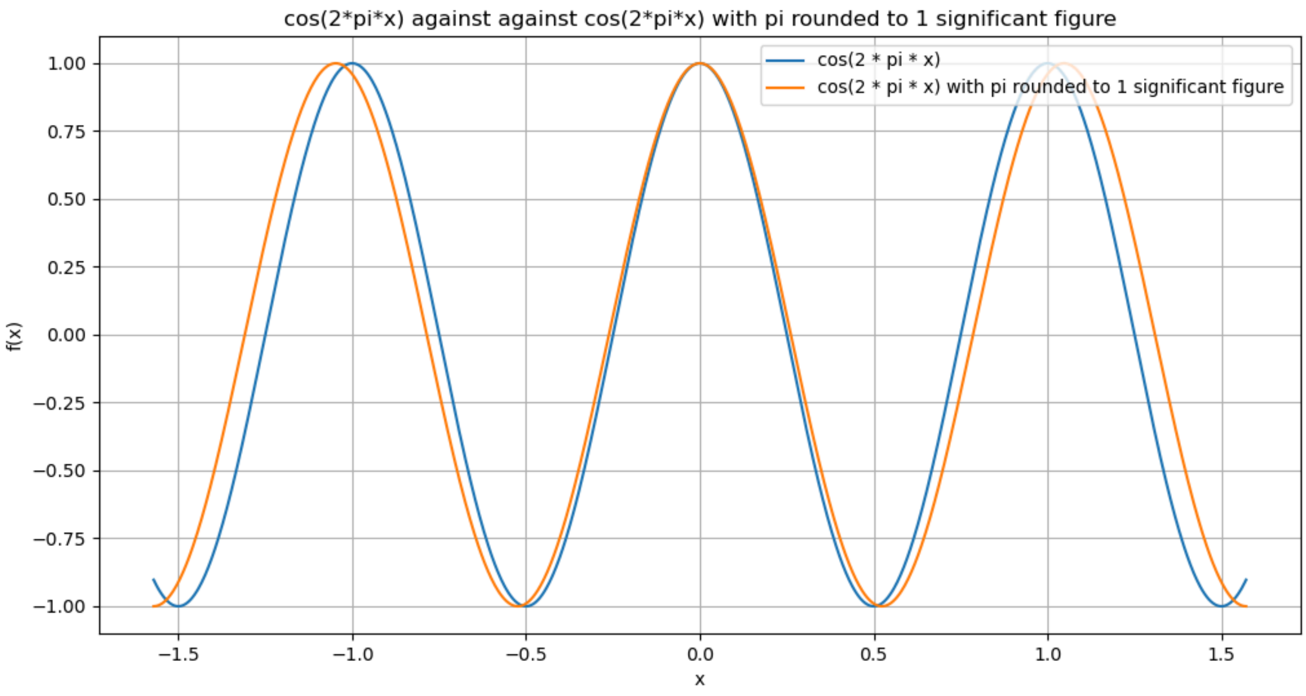

Fig 15.

Looking at a more extreme example, we can see that when pi is rounded to 1 significant figure, there is a much clearer desync when compared to that in figure 14.

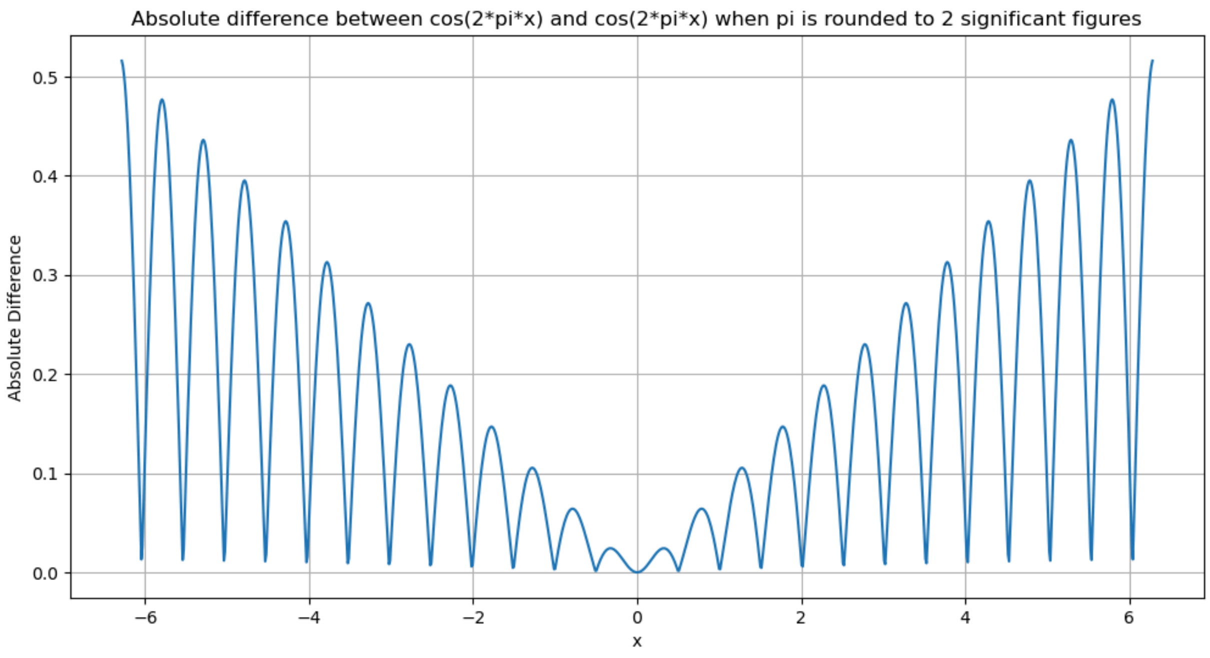

Fig 16.

When plotting the absolute difference between the 2 curves (shown in figure 14), we can see that the error climbs the further away from x = 0 we travel. As the 2 curves intersect, we can see the difference falling and then climbing again until the next intersection, however the peaks continue to climbe the further away we get.
It seems apparent, that eventually the curves will sync up again due to the nature of their differing periods. We can see this in the following figure 17:

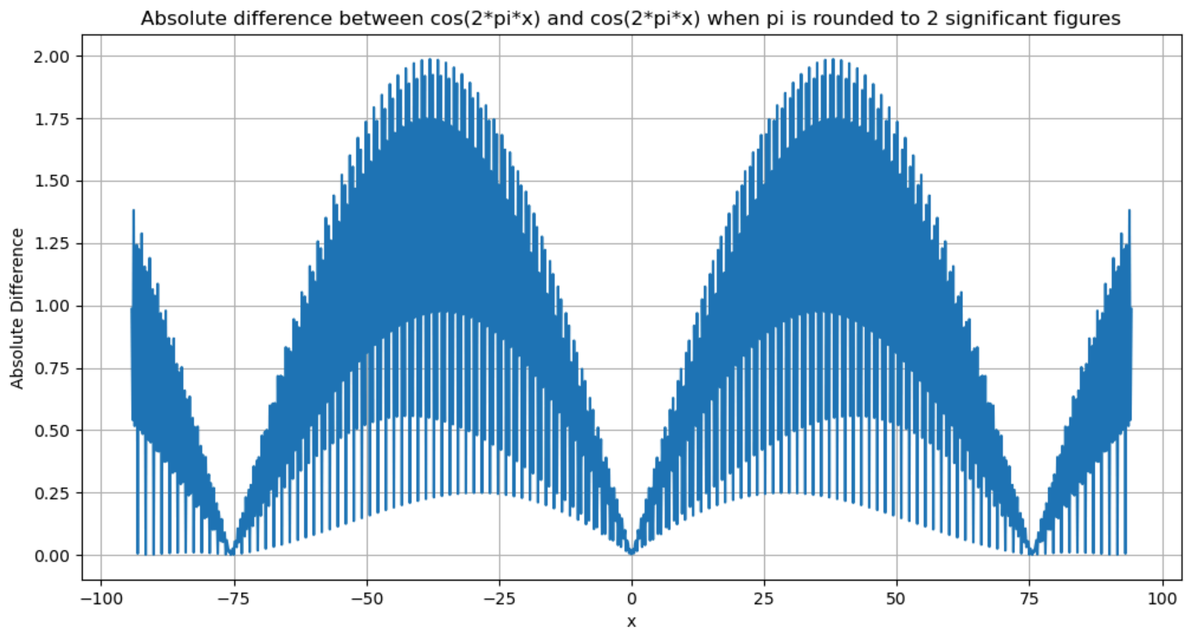

Fig 17.

We can see that eventually the plot of the absolute difference repeats itself eventually, as well as the rising errors in the negative and positive x directions from each intersection of periods. The low resolution of the graph is due to the stepsize in our plotting function and the large range of x values shown. Decreasing the stepsize may yield a higher resolution graph, however eventually this becomes extremely computationally expensive.

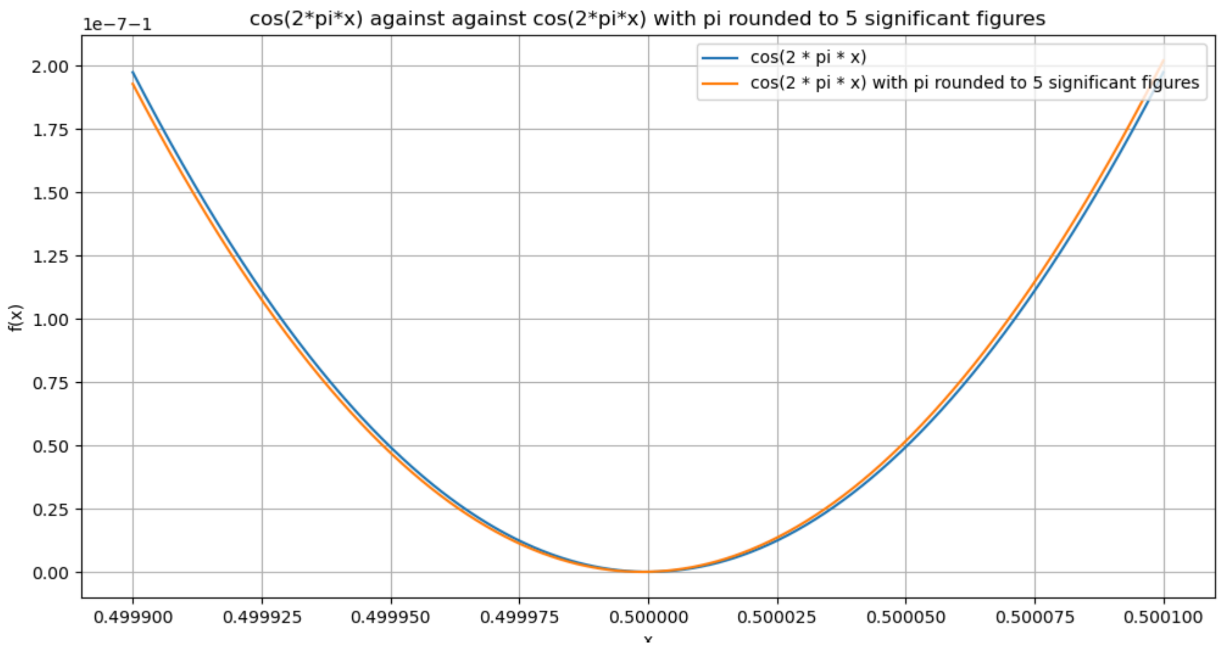

Fig 18.

When rounding pi to more significant digits, we gain greatly increased accuracy. We can see in Figure 18 that the difference in the curves is extremely small. When rounded to 5 digits, the resulting number is larger than pi, so we can see the slightly shorter period in our approximated curve.

Part 3 – Sensitivity and condition number

When evaluating the sensitivity with the 2 sources of error (a truncated Taylor Series centred at x = 0 after the first non zero term with inputs rounded to 3 significant figures), we arrive at an approximation function that is equal to:

f(x) = 1

As rounding inputs for f(x) = 1 to 3 significant figures is redundant (the result does not change for any rounding of x, or any value of x for that matter), we can simply analyse the sensitivity of the function without rounding.

Calculating the relative forward error is simple, as we just return:

| Delta y / y |

Which is the difference real y and approximated y based on our approximation function, divided by the real y value.

Calculating the relative backwards error is a bit more involved, as we are required to return:

| Delta x / x |

Delta x refers to the difference required in input for our real function to receive the same difference in output between the real and approximation functions. However, as our approximation function is simply a straight line, Delta x is equal to:

| x rounded to the nearest integer - x |

This is because our real function is always equal to 1 for integer values of x (i.e. the period of our real function is 1)

Using this, we can construct a plot to see the ratio of the relative forward and backwards errors, also known as the condition number:

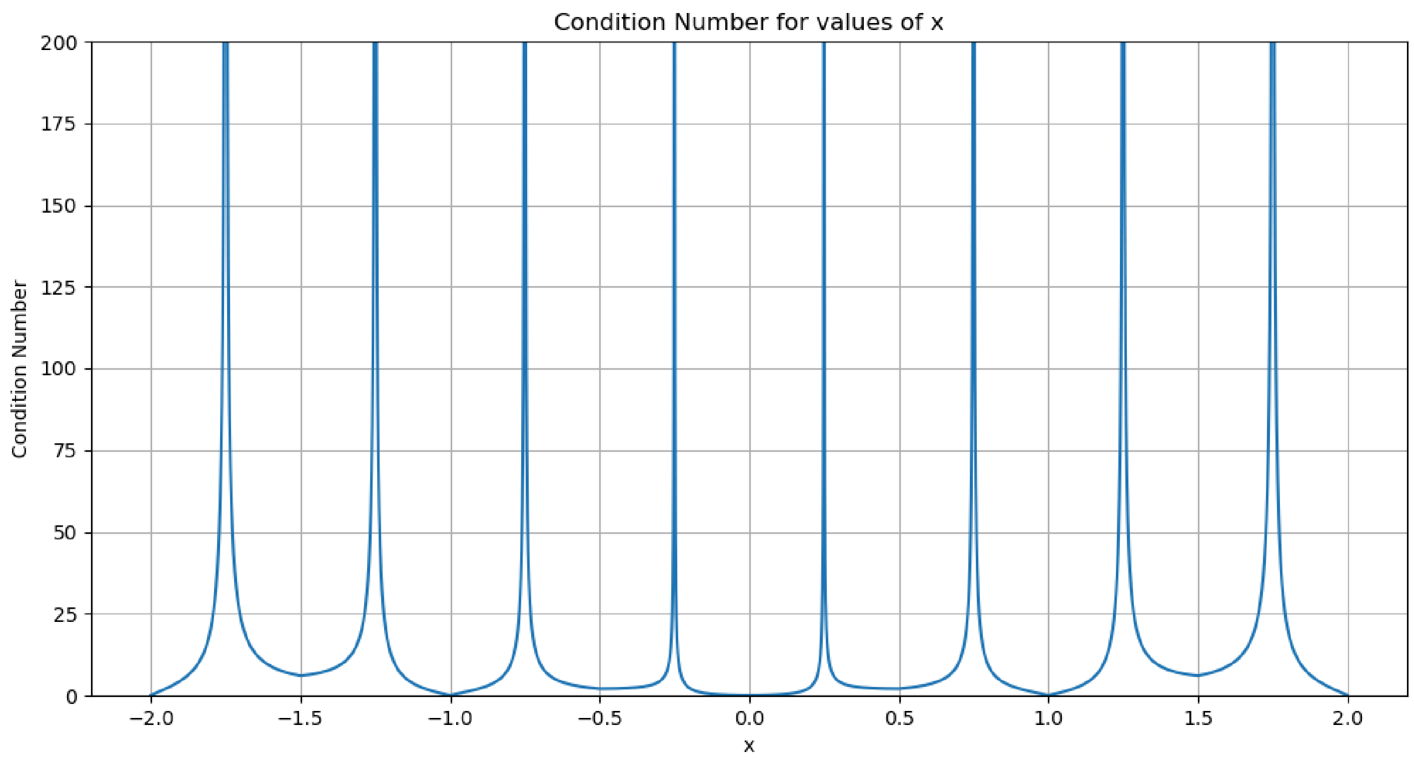

Fig 19.

Here we see the computed ratio of relative forward error over relative backward error for x values in the range [-2, 2]. We can see that the Condition Number increases greatly, indicating a very high sensitivity of the function around the zeros of our real function cos(2 * pi * x). As the function approaches the x-axis, any change in input x will produce massive changes in f(x) relative to the real value of f(x), as the denominator of our Condition Number is very close to 0.

As the function approaches our approximation every period, the condition number approaches 0, indicating a low sensitivity to error, as the output of our approximation and the output of the real function are relatively close.

Let us create an approximation for the Condition Number using the equation:

$$\text{Condition Number} \approx \left| \left(x f'(x) \right) / f(x) \right|$$

Which is an equation for approximating the condition number without having to calculate the relative forward and backwards errors. For our function, the plot of this equation is:

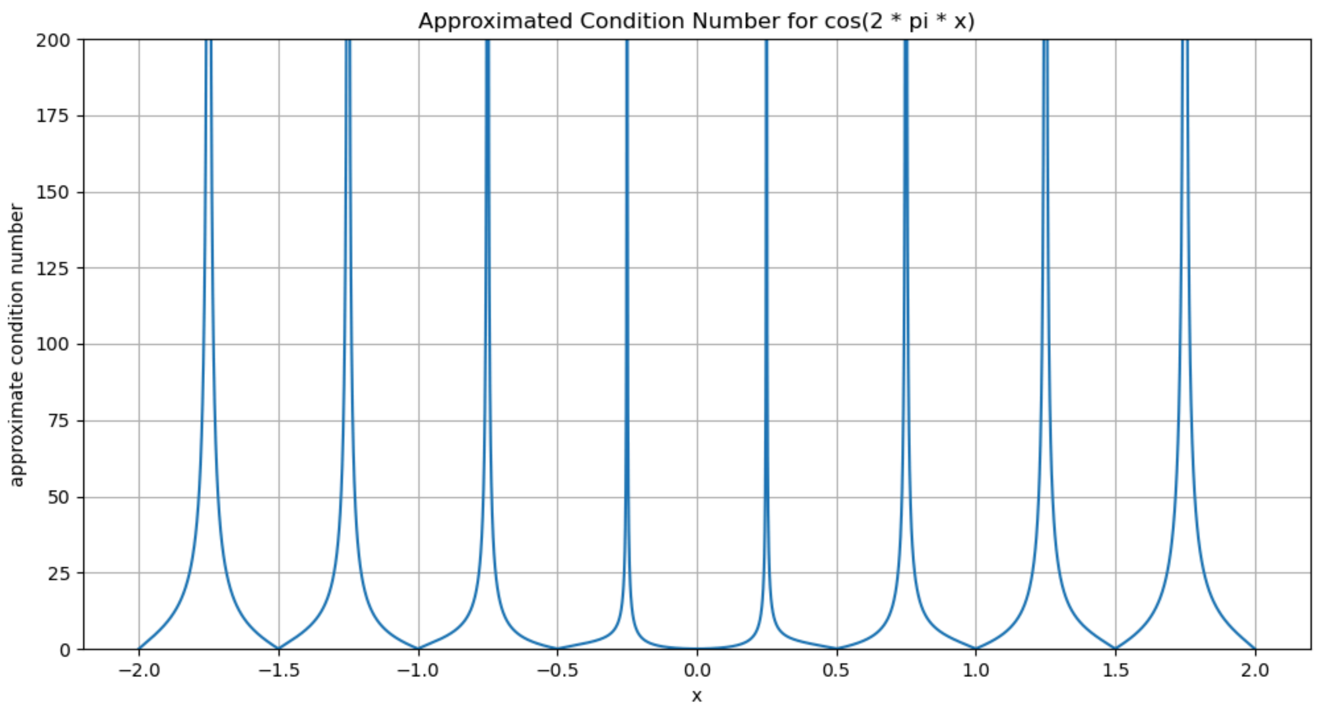

Fig 20.

Again, by using an approximation of the Condition Number, we can still see a remarkably similar shape of the graph. We can see that the sensitivity of the function rises extremely high as f(x) approaches 0. We also note that in between these zeros in the function are the same points in which the Condition Number is the lowest, signifying a decreased sensitivity at these points.

Overall, we can note that the function is very sensitive, as only rarely is the Condition Number, both calculated and approximated is near 1, and the vast majority of the function exhibits an extremely high sensitivity, where the Condition Number is greatly above 1, indicating that the function is sensitive.

Appendix

Libraries Used

NumPy, SymPy, MatPlotLib were all used in the creation of code for this assignment.

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

taylorSeries Function

```
def taylorSeries(f, c, n):
    """_summary_
        Generate all taylor series terms for a given function (f), with
a degree (n terms),
        centred at point x = c

    Args:
        f (_SymPy function_): original function
        c (_int / float_): centring point
        n (_int_): number of terms to compute

    Returns:
        taylorSeriesTerms (_arr_): array containing all terms of the
taylor series computed
    """

    x = sp.Symbol('x')

    taylorSeriesPolynomial = 0

    '''
    Creation of each individual term in the taylor series, summing them
as they are created
    '''
    for i in range(n + 1):
        factorial = sp.factorial(i)
        term = (f.subs(x, c) / factorial) * ((x - c) ** i)
        f = f.diff()
        taylorSeriesPolynomial += term

    return taylorSeriesPolynomial
```

smallAngleRelError Function

```
def smallAngleRelError(xVals):
    """_summary_
        Calculate relative error for the small angle theorem for cos(x)

    Args:
        xVals (_NumPy Linspace_): range of x values to be subbed into
our equation

    Returns:
        _y values_: the relative error at each x value, expressed as a
percentage
    """
    x = sp.Symbol('x')
    cos = sp.cos(x)
    approx = 1 - ((x ** 2) / 2)

    relError = abs((cos - approx) / cos) * 100
    relErrorLambdify = sp.lambdify(x, relError, 'numpy')

    return relErrorLambdify(xVals)
```

smallAngleBoundary Function

```
def smallAngleBoundary(tolerance, xVals):
    """_summary_
        Finds the boundary for which the relative error for the small
angle theorem for cos(x) is below the tolerance threshold
    Args:
        tolerance (_float/int_): tolerance for accuracy
        xVals (_NumPy Linspace_): _description_

    Returns:
        _float_: first x value in the linspace that is over the
tolerance (i.e. every value under this in our Linspace satisifies our
tolerance)
    """
    for x in xVals:
        error = smallAngleRelError(x)
        if error > tolerance:
            return x
```

roundPi Function

```python
def roundPi(sigFigs):
    """_summary_
        rounds pi to needed significant figures
    Args:
        sigFigs (_int_): number of significant figures to be rounded to

    Returns:
        _type_: pi rounded to sigFigs significant figures
    """
    return round(np.pi, sigFigs - 1)
```

relForwardError1Term Function

```python
def relForwardError1Term(x):
    """_summary_
        calculated relative forward error for Part 3
    Args:
        x (_float_): x value

    Returns:
        _type_: forward error of our function at that x value
    """
    cosval = np.cos(2 * np.pi * x)
    approxval = 1

    return abs((approxval - cosval) / cosval)
```

relBackwardsError1Term Function

```
def relBackwardError1Term(xVals):
    """_summary_
        calculated relative backwards error for Part 3
    Args:
        xVals (_NumPy Linspace_): range of x values to be processed

    Returns:
        _type_: array of backwards error values for each index in the
linspace input
    """
    backwardErrorVals = []
    for x in xVals:
        backwardErrorVals.append(abs((round(x, 0) - x) / x))
    return backwardErrorVals
```

conditionNumber1Term Function

```
def conditionNumber1Term(x):
    """_summary_
        calculates condition number for Part 3
    Args:
        x (_type_): x value

    Returns:
        _type_: condition number at x value
    """
    return relForwardError1Term(x) / relBackwardError1Term(x)
```

Plotting Functions

```
'''
Figure 1
'''
def plotFig1():
    x = sp.Symbol('x')

    cos = sp.cos(x)

    cosMaclaurinSeries = taylorSeries(cos, 0, 2)
    xVals = np.linspace(-1 * np.pi, np.pi, 1000)

    cosLambdify = sp.lambdify(x, cos, 'numpy')
    maclaurinLambdify = sp.lambdify(x, cosMaclaurinSeries, 'numpy')

    cosVals = cosLambdify(xVals)
    cosMacluarinVals = maclaurinLambdify(xVals)

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (14, 6))
    ax1.plot(xVals, cosVals, label = 'cos(x)')
    ax1.plot(xVals, cosMacluarinVals, label = 'Maclaurin Series of
cos(x) up to 2 terms')
    ax1.set_title('Plot of cos(x) against Maclaurin Series of cos(x) up
to 2 terms')
    ax1.set_xlabel('x')
    ax1.set_ylabel('f(x)')
    ax1.legend()
    ax1.grid(True)
    ax1.set_ylim(-1.2, 1.2)

    ax2.plot(xVals, abs(cosVals - cosMacluarinVals))
    ax2.set_title('Absolute difference between cos(x) and Maclaurin
Series')
    ax2.grid(True)

    plt.show()

'''
Figure 2
'''
def plotFig2():
    x = sp.Symbol('x')

    cos = sp.cos(x)

    cosMaclaurinSeries = taylorSeries(cos, 0, 4)
    xVals = np.linspace(-1 * np.pi, np.pi, 1000)

    cosLambdify = sp.lambdify(x, cos, 'numpy')
    maclaurinLambdify = sp.lambdify(x, cosMaclaurinSeries, 'numpy')

    cosVals = cosLambdify(xVals)
    cosMacluarinVals = maclaurinLambdify(xVals)

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (14, 6))
    ax1.plot(xVals, cosVals, label = 'cos(x)')
    ax1.plot(xVals, cosMacluarinVals, label = 'Maclaurin Series of 
cos(x) up to 4 terms')
    ax1.set_title('Plot of cos(x) against Maclaurin Series of cos(x) up 
to 4 terms')
    ax1.set_xlabel('x')
    ax1.set_ylabel('f(x)')
    ax1.legend()
    ax1.grid(True)
    ax1.set_ylim(-1.2, 1.2)

    ax2.plot(xVals, abs(cosVals - cosMacluarinVals))
    ax2.set_title('Absolute difference between cos(x) and Maclaurin 
Series')
    ax2.grid(True)

    plt.show()

'''
Figure 3
'''
def plotFig3():
    x = sp.Symbol('x')

    cos = sp.cos(x)

    plt.figure(figsize = (10, 6))

    for i in range(2, 9, 2):
        cosMaclaurinSeries = taylorSeries(cos, 0, i)
        xVals = np.linspace(-2 * np.pi, 2 * np.pi, 1000)

        cosLambdify = sp.lambdify(x, cos, 'numpy')
        maclaurinLambdify = sp.lambdify(x, cosMaclaurinSeries, 'numpy')

        cosVals = cosLambdify(xVals)
        cosMacluarinVals = maclaurinLambdify(xVals)

        plt.plot(xVals, abs(cosVals - cosMacluarinVals), label =
f"absolute difference between cos(x) and the Maclaurin Series at {i}
terms")

    plt.title("Absolute differences between cos(x) and the first 4
Maclaurin series of cos(x)")
    plt.xlabel('x')
    plt.ylabel('Absolute Difference')
    plt.legend()
    plt.grid()

    plt.show()

'''
Figure 4
'''
def plotFig4():
    x = sp.Symbol('x')

    cos = sp.cos(x)

    cosMaclaurinSeries = taylorSeries(cos, 0, 34)
    xVals = np.linspace(-5 * np.pi, 5 * np.pi, 1000)

    cosLambdify = sp.lambdify(x, cos, 'numpy')
    maclaurinLambdify = sp.lambdify(x, cosMaclaurinSeries, 'numpy')

    cosVals = cosLambdify(xVals)
    cosMacluarinVals = maclaurinLambdify(xVals)

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (16, 8))
    ax1.plot(xVals, cosVals, label = 'cos(x)')
    ax1.plot(xVals, cosMacluarinVals, label = 'Maclaurin Series of
cos(x) up to 34 terms')
```

```
    ax1.set_title('Plot of cos(x) against Maclaurin Series of cos(x) up
to 34 terms')
    ax1.set_xlabel('x')
    ax1.set_ylabel('f(x)')
    ax1.legend()
    ax1.grid(True)
    ax1.set_ylim(-1.2, 1.2)

    ax2.plot(xVals, abs(cosVals - cosMacluarinVals))
    ax2.set_title('Absolute difference between cos(x) and Maclaurin
Series')
    ax2.grid(True)

    plt.show()

'''
Figure 5
'''
def plotFig5():
    x = sp.Symbol('x')

    cos = sp.cos(x)

    cosTaylorSeries = taylorSeries(cos, np.pi/2, 1)
    xVals = np.linspace(0, np.pi, 1000)

    cosLambdify = sp.lambdify(x, cos, 'numpy')
    taylorLambdify = sp.lambdify(x, cosTaylorSeries, 'numpy')

    cosVals = cosLambdify(xVals)
    cosTaylorVals = taylorLambdify(xVals)

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (14, 6))
    ax1.plot(xVals, cosVals, label = 'cos(x)')
    ax1.plot(xVals,  cosTaylorVals, label = 'Taylor Series of cos(x) up
to 1 term')
    ax1.set_title('Plot of cos(x) against Taylor Series of cos(x) up to
1 term centred at x = pi/2')
    ax1.set_xlabel('x')
    ax1.set_ylabel('f(x)')
    ax1.legend()
    ax1.grid(True)
    ax2.plot(xVals, abs(cosVals - cosTaylorVals))
    ax2.set_title('Absolute difference between cos(x) and Taylor
Series')
    ax2.grid(True)

    plt.show()

'''
Figure 6
'''
def plotFig6():
    x = sp.Symbol('x')

    cos = sp.cos(x)

    cosTaylorSeries = taylorSeries(cos, np.pi/2, 22)
    xVals = np.linspace(-2.5 * np.pi, 3.5 * np.pi, 1000)

    cosLambdify = sp.lambdify(x, cos, 'numpy')
    taylorLambdify = sp.lambdify(x, cosTaylorSeries, 'numpy')

    cosVals = cosLambdify(xVals)
    cosTaylorVals = taylorLambdify(xVals)

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (16, 8))
    ax1.plot(xVals, cosVals, label = 'cos(x)')
    ax1.plot(xVals,  cosTaylorVals, label = 'Taylor Series of cos(x) up
to 1 term')
    ax1.set_title('Plot of cos(x) against Taylor Series of cos(x) up to
1 term centred at x = pi/2')
    ax1.set_xlabel('x')
    ax1.set_ylabel('f(x)')
    ax1.legend()
    ax1.grid(True)
    ax1.set_ylim(-1.2, 1.2)

    ax2.plot(xVals, abs(cosVals - cosTaylorVals))
    ax2.set_title('Absolute difference between cos(x) and Taylor
Series')
    ax2.grid(True)

    plt.show()
```

```
'''
Figure 7
'''
def plotFig7():
    x = sp.Symbol('x')

    cos = sp.cos(x)

    plt.figure(figsize = (10, 6))

    for i in range(1, 8, 2):
        cosMaclaurinSeries = taylorSeries(cos, np.pi/2, i)
        xVals = np.linspace(-1 * np.pi, 2 * np.pi, 1000)

        cosLambdify = sp.lambdify(x, cos, 'numpy')
        maclaurinLambdify = sp.lambdify(x, cosMaclaurinSeries, 'numpy')

        cosVals = cosLambdify(xVals)
        cosMacluarinVals = maclaurinLambdify(xVals)

        plt.plot(xVals, abs(cosVals - cosMacluarinVals), label =
f"absolute difference between cos(x) and the Taylor Series at {i}
terms")

    plt.title("Absolute differences between cos(x) and the first 4
Taylor Series of cos(x) centred at x = pi/2")
    plt.xlabel('x')
    plt.ylabel('Absolute Difference')
    plt.legend()
    plt.grid()

    plt.show()

'''
Figure 8
'''
def plotFig8():
    x = sp.Symbol('x')
    xVals = np.linspace(0, 0.3, 1000)

    plt.plot(xVals, smallAngleRelError(xVals))
    plt.ylim(0, 0.1)
    plt.title("Relative error of the small angle theorem for cos(x)")
    plt.xlabel('x')
    plt.ylabel('Relative Error %')
    plt.show()

'''
Figure 9
'''
def plotFig9():
    xVals = np.linspace(0, 0.3, 1000)
    maxXVal = smallAngleBoundary(0.01, xVals)

    plt.plot(xVals, smallAngleRelError(xVals), label = 'relative
error')
    plt.axvline(x = maxXVal, label = f'approximate maximum x value
({round(maxXVal, 4)})', color = 'red')
    plt.axhline(y = 0.01, label = '0.01% tolerance', color = 'red')
    plt.ylim(0, 0.05)
    plt.title("Relative error of the small angle theorem for cos(x)")
    plt.xlabel('x')
    plt.ylabel('Relative Error %')
    plt.legend()
    plt.show()

    plt.show()

'''
Figure 10
'''
def plotFig10():
    x = sp.Symbol('x')

    tan = sp.tan(x)

    tanMaclaurinSeries = taylorSeries(tan, 0, 3)
    xVals = np.linspace(-2 * np.pi, 2 * np.pi, 1000)

    tanLambdify = sp.lambdify(x, tan, 'numpy')
    maclaurinLambdify = sp.lambdify(x, tanMaclaurinSeries, 'numpy')

    tanVals = tanLambdify(xVals)
    tanMacluarinVals = maclaurinLambdify(xVals)

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (16, 8))
    ax1.plot(xVals, tanVals, label = 'tan(x)')
    ax1.plot(xVals, tanMacluarinVals, label = 'Maclaurin Series of 
tan(x) up to 3 terms')
    ax1.set_title('Plot of cos(x) against Maclaurin Series of tan(x) up 
to 3 terms')
    ax1.set_xlabel('x')
    ax1.set_ylabel('f(x)')
    ax1.legend()
    ax1.grid(True)
    ax1.set_ylim(-15, 15)

    ax2.plot(xVals, abs(tanVals - tanMacluarinVals))
    ax2.set_title('Absolute difference between tan(x) and Maclaurin 
Series')
    ax2.set_ylim(0, 100)

    plt.show()

'''
Figure 11
'''
def plotFig11():
    x = sp.Symbol('x')

    tan = sp.tan(x)

    tanMaclaurinSeries = taylorSeries(tan, 0, 15)
    xVals = np.linspace(-2 * np.pi, 2 * np.pi, 1000)

    tanLambdify = sp.lambdify(x, tan, 'numpy')
    maclaurinLambdify = sp.lambdify(x, tanMaclaurinSeries, 'numpy')

    tanVals = tanLambdify(xVals)
    tanMacluarinVals = maclaurinLambdify(xVals)

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (16, 8))
    ax1.plot(xVals, tanVals, label = 'tan(x)')
    ax1.plot(xVals, tanMacluarinVals, label = 'Maclaurin Series of 
tan(x) up to 15 terms')
    ax1.set_title('Plot of cos(x) against Maclaurin Series of tan(x) up 
to 15 terms')
    ax1.set_xlabel('x')
    ax1.set_ylabel('f(x)')
    ax1.legend()
    ax1.grid(True)
    ax1.set_ylim(-15, 15)

    ax2.plot(xVals, abs(tanVals - tanMacluarinVals))
    ax2.set_title('Absolute difference between tan(x) and Maclaurin
Series')
    ax2.set_ylim(0, 100)

    plt.show()

'''
Figure 12
'''
def plotFig12():
    x = sp.Symbol('x')

    tan = sp.tan(x)

    tanMaclaurinSeries = taylorSeries(tan, np.pi / 4, 5)
    xVals = np.linspace(-2 * np.pi, 2 * np.pi, 1000)

    tanLambdify = sp.lambdify(x, tan, 'numpy')
    maclaurinLambdify = sp.lambdify(x, tanMaclaurinSeries, 'numpy')

    tanVals = tanLambdify(xVals)
    tanMacluarinVals = maclaurinLambdify(xVals)

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (16, 8))
    ax1.plot(xVals, tanVals, label = 'tan(x)')
    ax1.plot(xVals, tanMacluarinVals, label = 'Taylor Series of tan(x)
up to 5 terms centred at pi/4')
    ax1.set_title('Plot of cos(x) against Taylor Series of tan(x) up to
5 terms')
    ax1.set_xlabel('x')
    ax1.set_ylabel('f(x)')
    ax1.legend()
    ax1.grid(True)
    ax1.set_ylim(-15, 15)

    ax2.plot(xVals, abs(tanVals - tanMacluarinVals))
    ax2.set_title('Absolute difference between tan(x) and Taylor
Series')
```

```
    ax2.set_ylim(0, 100)
    ax2.set_xlabel('x')
    ax2.set_ylabel('Absolute Difference')

    plt.show()

'''
Figure 13
'''
def plotFig13():
    x = sp.Symbol('x')

    tan = sp.tan(x)

    tanMaclaurinSeries = taylorSeries(tan, np.pi / 4, 15)
    xVals = np.linspace(-2 * np.pi, 2 * np.pi, 1000)

    tanLambdify = sp.lambdify(x, tan, 'numpy')
    maclaurinLambdify = sp.lambdify(x, tanMaclaurinSeries, 'numpy')

    tanVals = tanLambdify(xVals)
    tanMacluarinVals = maclaurinLambdify(xVals)

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (16, 8))
    ax1.plot(xVals, tanVals, label = 'tan(x)')
    ax1.plot(xVals, tanMacluarinVals, label = 'Taylor Series of tan(x)
up to 15 terms centred at pi/4')
    ax1.set_title('Plot of cos(x) against Taylor Series of tan(x) up to
15 terms')
    ax1.set_xlabel('x')
    ax1.set_ylabel('f(x)')
    ax1.legend()
    ax1.grid(True)
    ax1.set_ylim(-15, 15)

    ax2.plot(xVals, abs(tanVals - tanMacluarinVals))
    ax2.set_title('Absolute difference between tan(x) and Taylor
Series')
    ax2.set_ylim(0, 100)
    ax2.set_xlabel('x')
    ax2.set_ylabel('Absolute Difference')

    plt.show()
```

```
'''
Figure 14
'''
def plotFig14():
    xVals = np.linspace(-1/2 * np.pi, 1/2 * np.pi, 1000)

    x = sp.Symbol('x')

    func = sp.cos(2 * np.pi * x)
    roundedFunc = sp.cos(2 * roundPi(2) * x)

    funcLamdify = sp.lambdify(x, func, 'numpy')
    roundedFuncLamdify = sp.lambdify(x, roundedFunc, 'numpy')

    plt.figure(figsize = (12, 6))
    plt.plot(xVals, funcLamdify(xVals), label = 'cos(2 * pi * x)')
    plt.plot(xVals, roundedFuncLamdify(xVals), label = 'cos(2 * pi * x)
with pi rounded to 2 significant figures')
    plt.title("cos(2*pi*x) against against cos(2*pi*x) with pi rounded
to 2 significant figures")
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.legend(loc = 1)
    plt.grid()
    plt.show()

'''
Figure 15
'''
def plotFig15():
    xVals = np.linspace(-1/2 * np.pi, 1/2 * np.pi, 1000)

    x = sp.Symbol('x')

    func = sp.cos(2 * np.pi * x)
    roundedFunc = sp.cos(2 * roundPi(1) * x)

    funcLamdify = sp.lambdify(x, func, 'numpy')
    roundedFuncLamdify = sp.lambdify(x, roundedFunc, 'numpy')

    plt.figure(figsize = (12, 6))
    plt.plot(xVals, funcLamdify(xVals), label = 'cos(2 * pi * x)')
    plt.plot(xVals, roundedFuncLamdify(xVals), label = 'cos(2 * pi * x)
with pi rounded to 1 significant figure')
    plt.title("cos(2*pi*x) against against cos(2*pi*x) with pi rounded
to 1 significant figure")
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.legend(loc = 1)
    plt.grid()
    plt.show()

'''
Figure 16
'''
def plotFig16():
    xVals = np.linspace(-2 * np.pi, 2 * np.pi, 1000)

    x = sp.Symbol('x')

    func = sp.cos(2 * np.pi * x)
    roundedFunc = sp.cos(2 * roundPi(2) * x)

    absError = func - roundedFunc

    absErrorLambd = sp.lambdify(x, absError, 'numpy')

    plt.figure(figsize = (12, 6))
    plt.plot(xVals, abs(absErrorLambd(xVals)))
    plt.title("Absolute difference between cos(2*pi*x) and cos(2*pi*x)
when pi is rounded to 2 significant figures")
    plt.xlabel('x')
    plt.ylabel('Absolute Difference')
    plt.grid()
    plt.show()

'''
Figure 17
'''
def plotFig17():
    xVals = np.linspace(-30 * np.pi, 30 * np.pi, 1000)

    x = sp.Symbol('x')

    func = sp.cos(2 * np.pi * x)
    roundedFunc = sp.cos(2 * roundPi(2) * x)

    absError = func - roundedFunc

    absErrorLambd = sp.lambdify(x, absError, 'numpy')

    plt.figure(figsize = (12, 6))
    plt.plot(xVals, abs(absErrorLambd(xVals)))
    plt.title("Absolute difference between cos(2*pi*x) and cos(2*pi*x)
when pi is rounded to 2 significant figures")
    plt.xlabel('x')
    plt.ylabel('Absolute Difference')
    plt.grid()
    plt.show()

'''
Figure 18
'''
def plotFig18():
    #xVals = np.linspace(-1/2 * np.pi, 1/2 * np.pi, 1000)
    xVals = np.linspace(0.4999, 0.5001, 1000)

    x = sp.Symbol('x')

    func = sp.cos(2 * np.pi * x)
    roundedFunc = sp.cos(2 * roundPi(5) * x)

    funcLamdify = sp.lambdify(x, func, 'numpy')
    roundedFuncLamdify = sp.lambdify(x, roundedFunc, 'numpy')

    plt.figure(figsize = (12, 6))
    plt.plot(xVals, funcLamdify(xVals), label = 'cos(2 * pi * x)')
    plt.plot(xVals, roundedFuncLamdify(xVals), label = 'cos(2 * pi * x)
with pi rounded to 5 significant figures')
    plt.title("cos(2*pi*x) against against cos(2*pi*x) with pi rounded
to 5 significant figures")
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.legend(loc = 1)
    plt.grid()
    plt.show()

'''
```

```python
Figure 19
'''
def plotFig19():
    xVals = np.linspace(-2, 2, 5000)

    x = sp.Symbol('x')

    plt.figure(figsize = (12, 6))
    plt.plot(xVals, conditionNumber1Term(xVals))
    plt.title("Condition Number for values of x")
    plt.xlabel('x')
    plt.ylabel('Condition Number')
    plt.grid()
    plt.ylim(0, 200)
    plt.show()

'''
Figure 20
'''
def plotFig20():
    xVals = np.linspace(-2, 2, 5000)

    x = sp.Symbol('x')

    f = sp.cos(2 * np.pi * x)
    fprime = f.diff(x)
    condNumber = (x * fprime) / f

    condNumberLambdify = sp.lambdify(x, condNumber, 'numpy')

    plt.figure(figsize = (12, 6))
    plt.plot(xVals, abs(condNumberLambdify(xVals)))
    plt.title("Approximated Condition Number for cos(2 * pi * x)")
    plt.xlabel('x')
    plt.ylabel('approximate condition number')
    plt.grid()
    plt.ylim(0, 200)
    plt.show()
```