EE5011: Assignment 1

Polynomial Interpolation

J.Phani Jayanth - EE19B026

30 August 2022

Objective

- Utilizing *weave* to link a C implementation of *Polynomial Implementation* with Python.
- Performing the interpolation for several orders and sampling rates to draw analyses.
- Plotting and observing the effect of interpolation order and sampling rate on the interpolation accuracy (mean and maximum error) by studying the trend between the error in interpolation and the order of interpolation.

Polynomial Interpolation

We know that, the interpolating polynomial of degree $N-1$ through the N points $y_1 = f(x_1), y_2 = f(x_2), ..., y_N = f(x_N)$ is given by the Lagrange's classical formula:

$$P(x) = \frac{(x-x_2)(x-x_3)...(x-x_N)}{(x_1-x_2)(x_1-x_3)...(x_1-x_N)}y_1 + \frac{(x-x_1)(x-x_3)...(x-x_N)}{(x_2-x_1)(x_2-x_3)...(x_2-x_N)}y_2 + ... + \frac{(x-x_1)(x-x_2)...(x-x_{N-1})}{(x_N-x_1)(x_N-x_2)...(x_N-x_{N-1})}y_N$$

The straightforward implementation of Lagrange formula gives no error estimate. A much better algorithm (for constructing the same, unique, interpolating polynomial) is the *Neville's Algorithm*. Let P1 be the value at x of the unique polynomial of degree zero (i.e., a constant) passing through the point $(x1, y1)$; so $P1 = y1$. Likewise, define $P2, P3, ..., PN$. Now let $P12$ be the value at x of the unique polynomial of degree one passing through both $(x1, y1)$ and $(x2, y2)$. Likewise $P23, P34, ..., P(N1)N$. Similarly, for higher-order polynomials, up to $P123...N$, which is the value of the unique interpolating polynomial through all N points, i.e., the desired answer. For example, with $N = 4$:

$$\begin{array}{lllll} x_1: & y_1 = P_1 & & & \\ & & P_{12} & & \\ x_2: & y_2 = P_2 & & P_{123} & \\ & & P_{23} & & P_{1234} \\ x_3: & y_3 = P_3 & & P_{234} & \\ & & P_{34} & & \\ x_4: & y_4 = P_4 & & & \end{array}$$

Neville's algorithm is a recursive way of filling in the numbers in the tableau a column at a time, from left to right. It is based on the relationship between a "daughter" P and its two "parents,"

$$P_{i(i+1)...(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)...(i+m-1)} + (x_i - x)P_{(i+1)(i+2)...(i+m)}}{x_i - x_{i+m}}$$

This recurrence works because the two parents already agree at points $x_{i+1}...x_{i+m1}$. An improvement on the recurrence is to keep track of the small *differences* between parents and daughters, namely to define (for $m = 1, 2, ..., N - 1$),

$$C_{m,i} \equiv P_{i...(i+m)} - P_{i...(i+m-1)}$$

$$D_{m,i} \equiv P_{i...(i+m)} - P_{(i+1)...(i+m)}$$

From this, we can derive the relations:

$$D_{m+1,i} = \frac{(x_{i+m+1}-x)(C_{m,i+1}-D_{m,i})}{x_i - x_{i+m+1}}$$

$$C_{m+1,i} = \frac{(x_i-x)(C_{m,i+1}-D_{m,i})}{x_i - x_{i+m+1}}$$

At each level m, the C's and D's are the corrections that make the interpolation one order higher. The final answer $P_{1\ldots N}$ is equal to the sum of any y_i plus a set of C's and/or D's that form a path through the family tree to the rightmost daughter.

Question 1

We sample the function $f(x) = sin(x + x^2)$ from 0 to 1 at 5 points. We then perform a 4^{th} order interpolation of $f(x)$. We obtain the *polynomial interpolation* as follows, where the red dots represent the interpolated values.

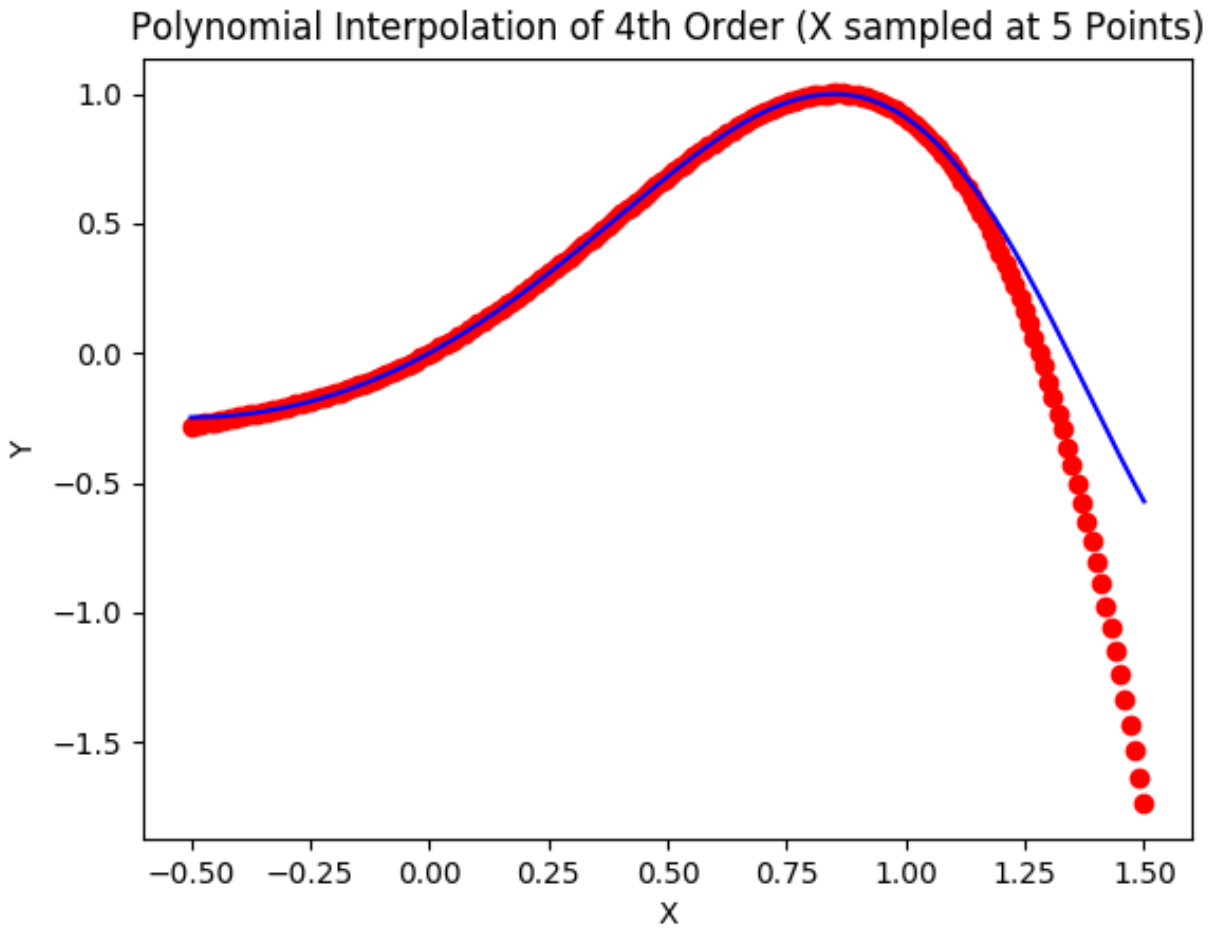


Figure 1: 4^{th} order Interpolation of $f(x) = sin(x + x^2)$ sampled at 5 points

We also plot the *average error* in the interpolation to assess its accuracy. We plot is as follows, with a *semilog-Y* axis.

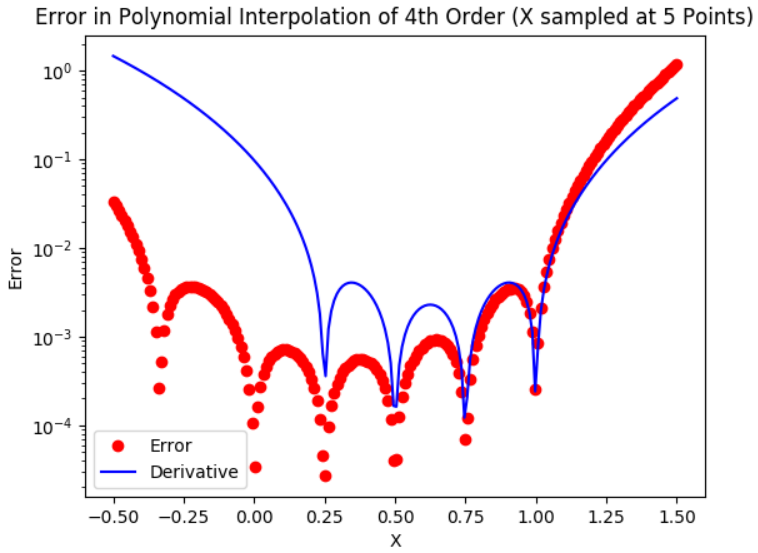


Figure 2: Error in 4^{th} order Interpolation of $f(x) = sin(x + x^2)$ sampled at 5 points

Question 2

The same function, i.e., $f(x) = sin(x + x^2)$ is now sampled from 0 to 1 at 30 points. We again perform a 4^{th} order interpolation as shown in the figure.

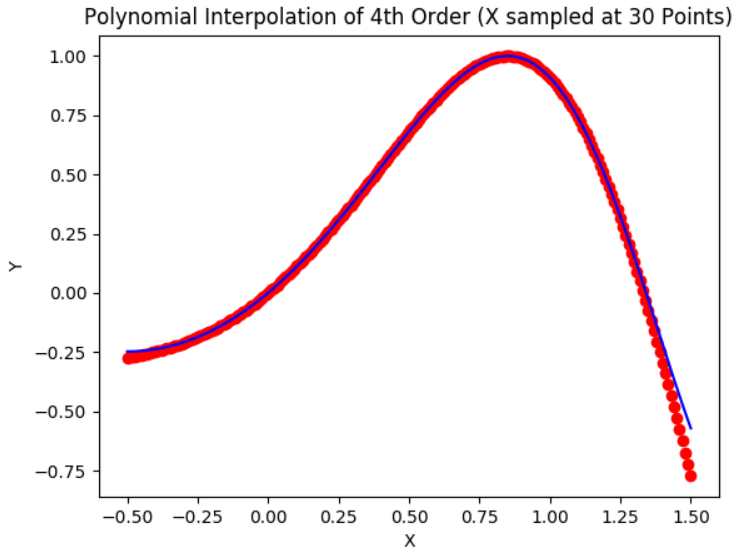


Figure 3: 4^{th} order Interpolation of $f(x) = sin(x + x^2)$ sampled at 30 points

We also plot the *average error* in incurred in the interpolation on a *semilog-Y* axis, as follows:

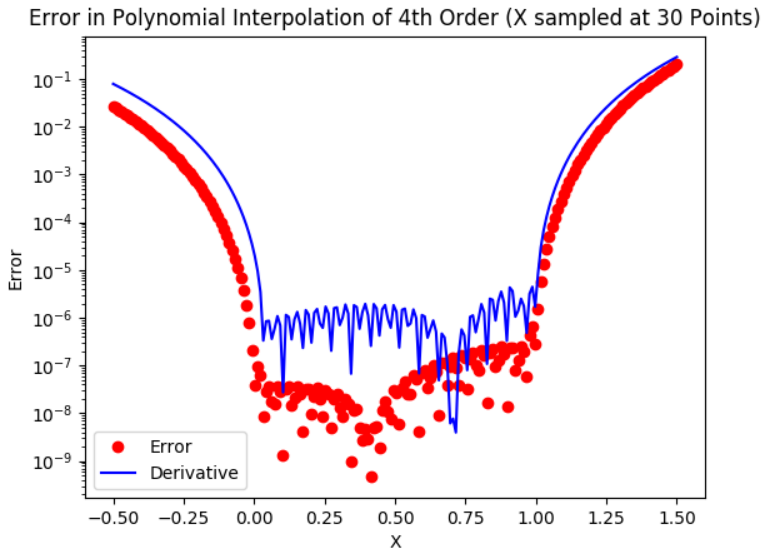


Figure 4: Error in 4^{th} order Interpolation of $f(x) = sin(x + x^2)$ sampled at 30 points

Question 3 & Question 4

With the same table of values, we vary the order of interpolation and observe how the associated error changes. We obtain the following trend in *Error* vs *Order of Polynomial Interpolation*.

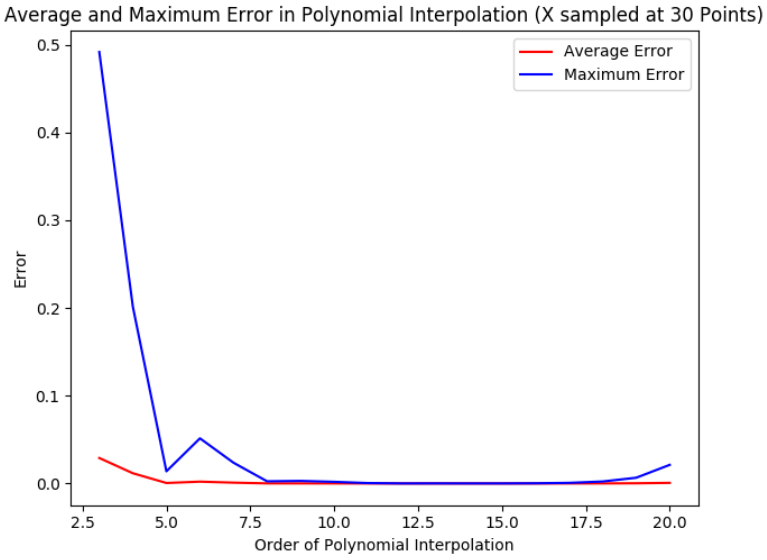
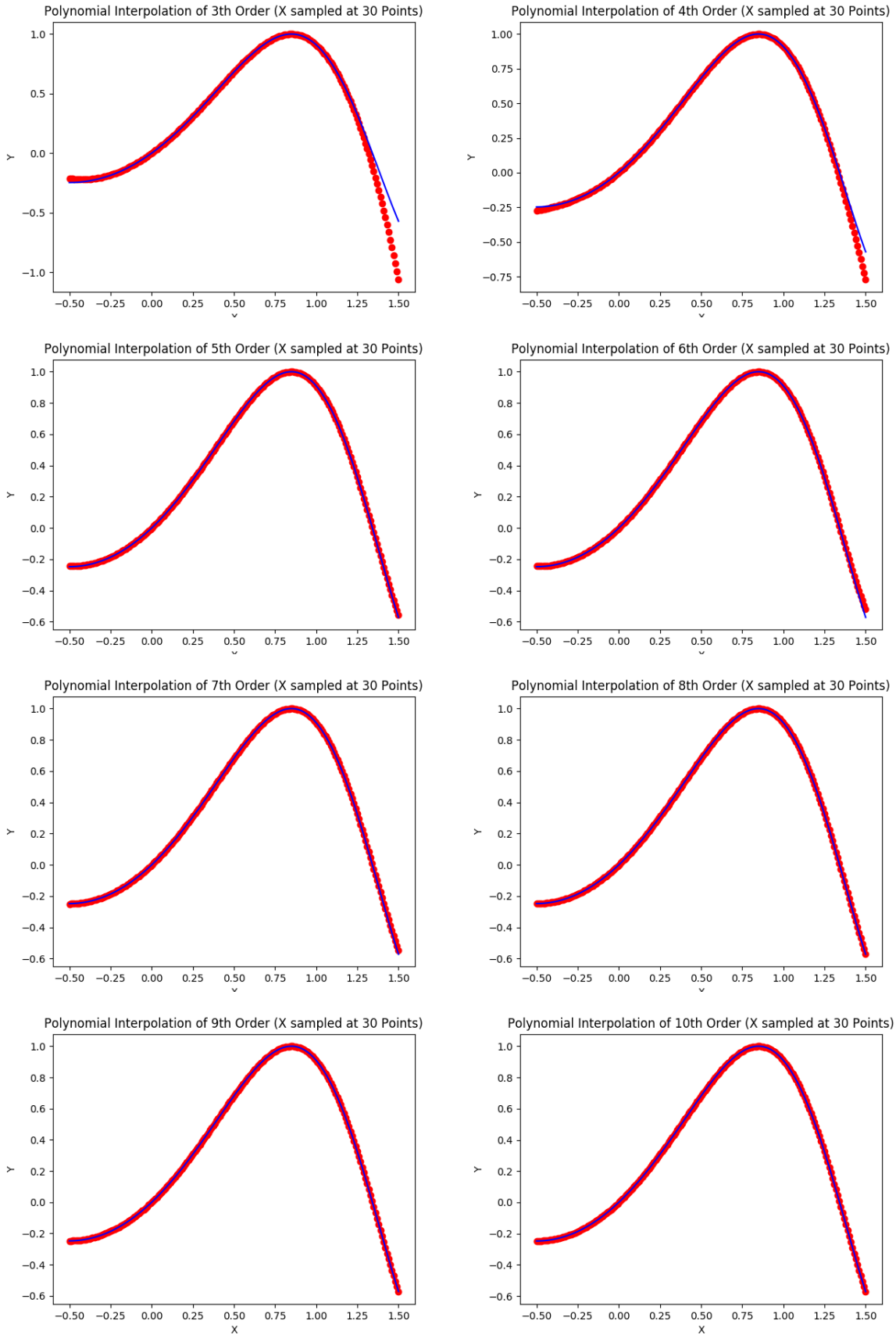
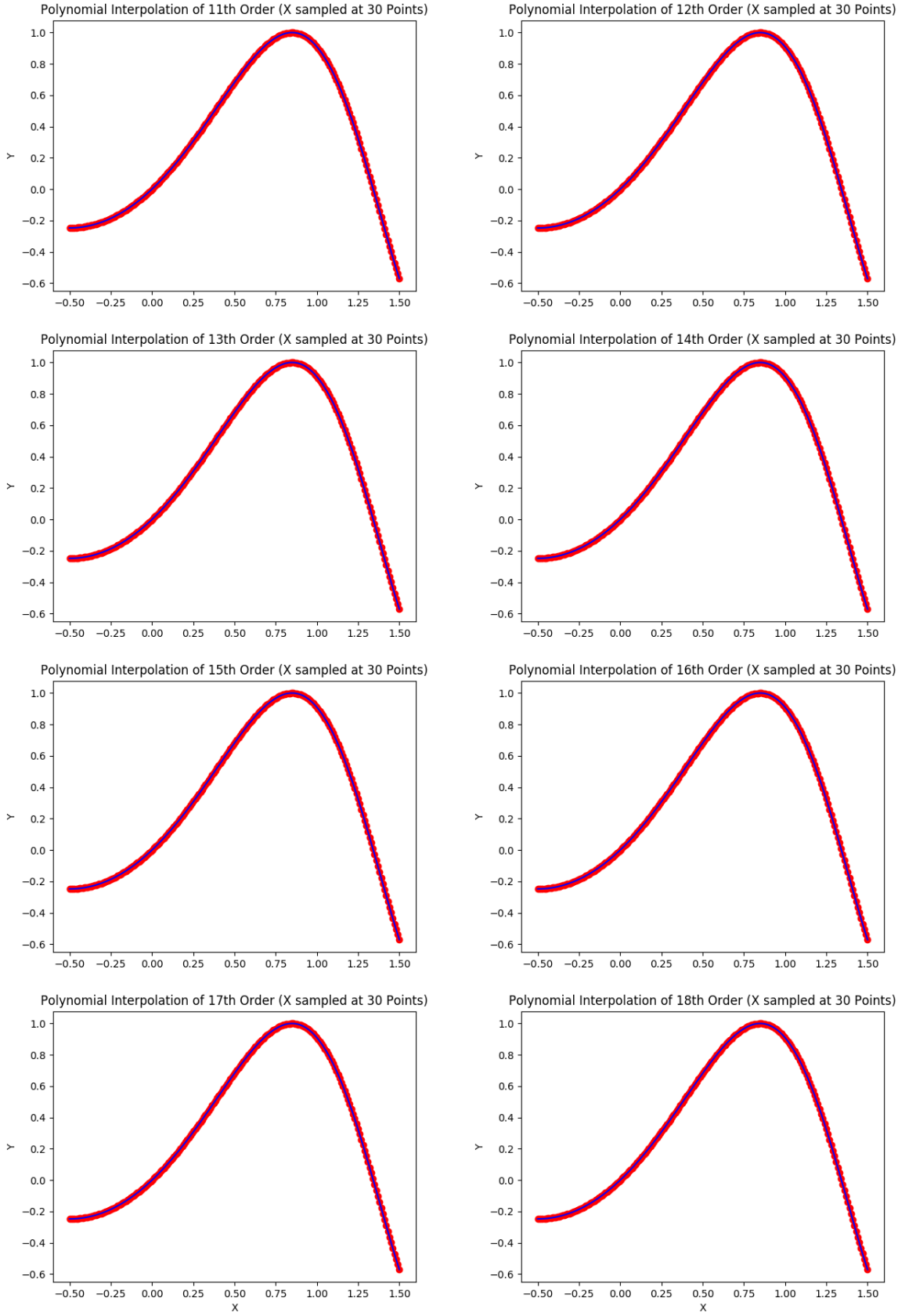


Figure 5: Error vs the order of Polynomial Interpolation

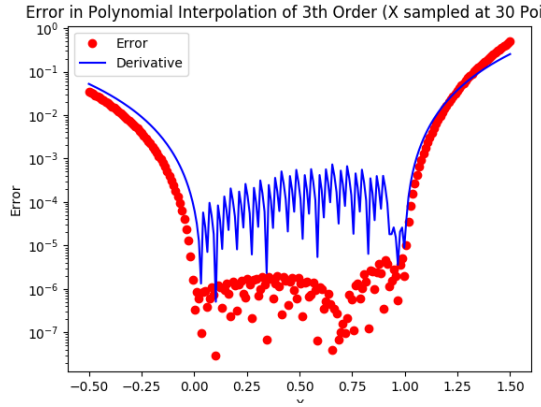
Clearly, we observe that both the *average* and the *maximum* error decrease with increasing the order of *polynomial interpolation*.

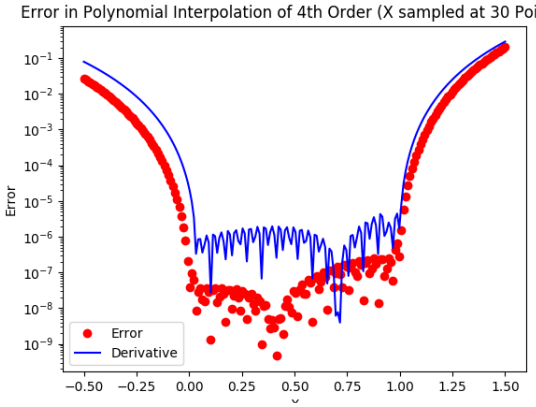
The obtained interpolated values are plotted against the actual function for various orders (3,4,5,...,18) of interpolation, as shown in the following figures.

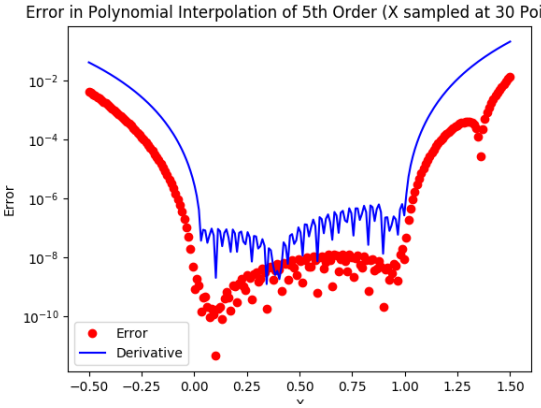


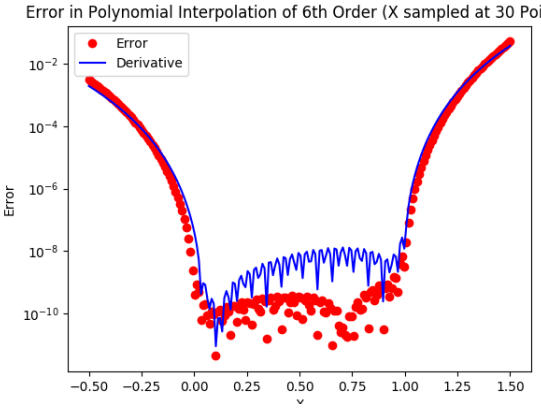


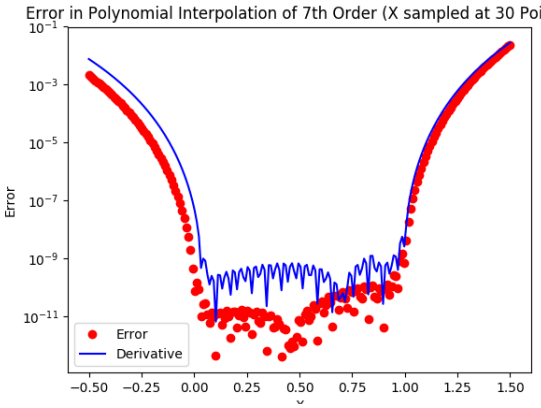
The errors in interpolation across various orders (3,4,5,...,18) are plotted against the actual function, as shown in the following figures.

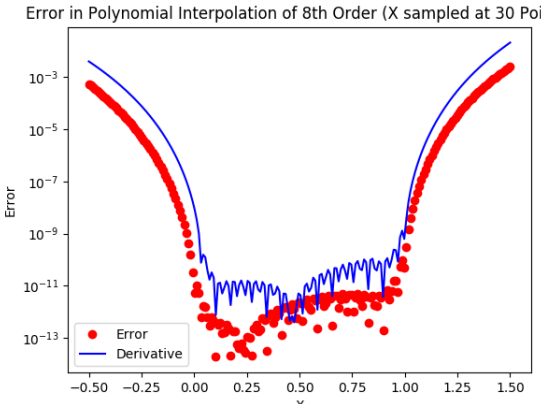

Error in Polynomial Interpolation of 3th Order (X sampled at 30 Points)
Error
Derivative
Error

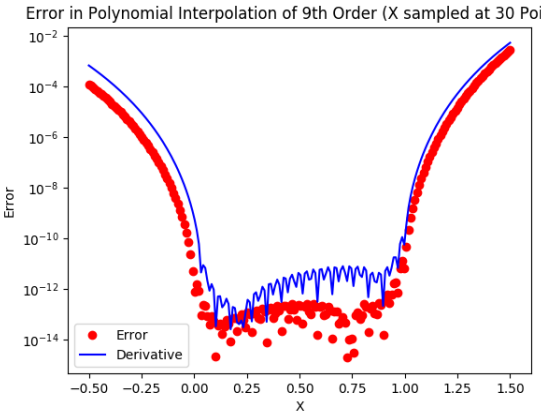

Error in Polynomial Interpolation of 4th Order (X sampled at 30 Points)
Error
Derivative
Error

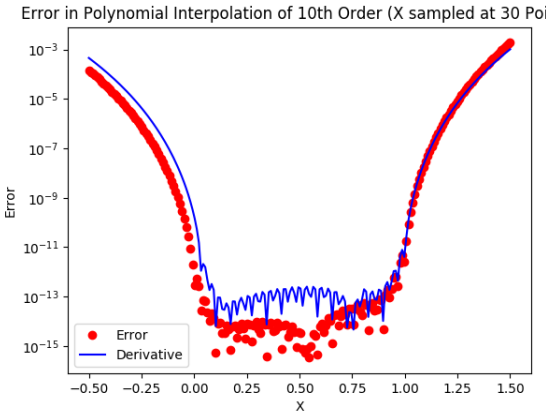

Error in Polynomial Interpolation of 5th Order (X sampled at 30 Points)
Error
Derivative
Error

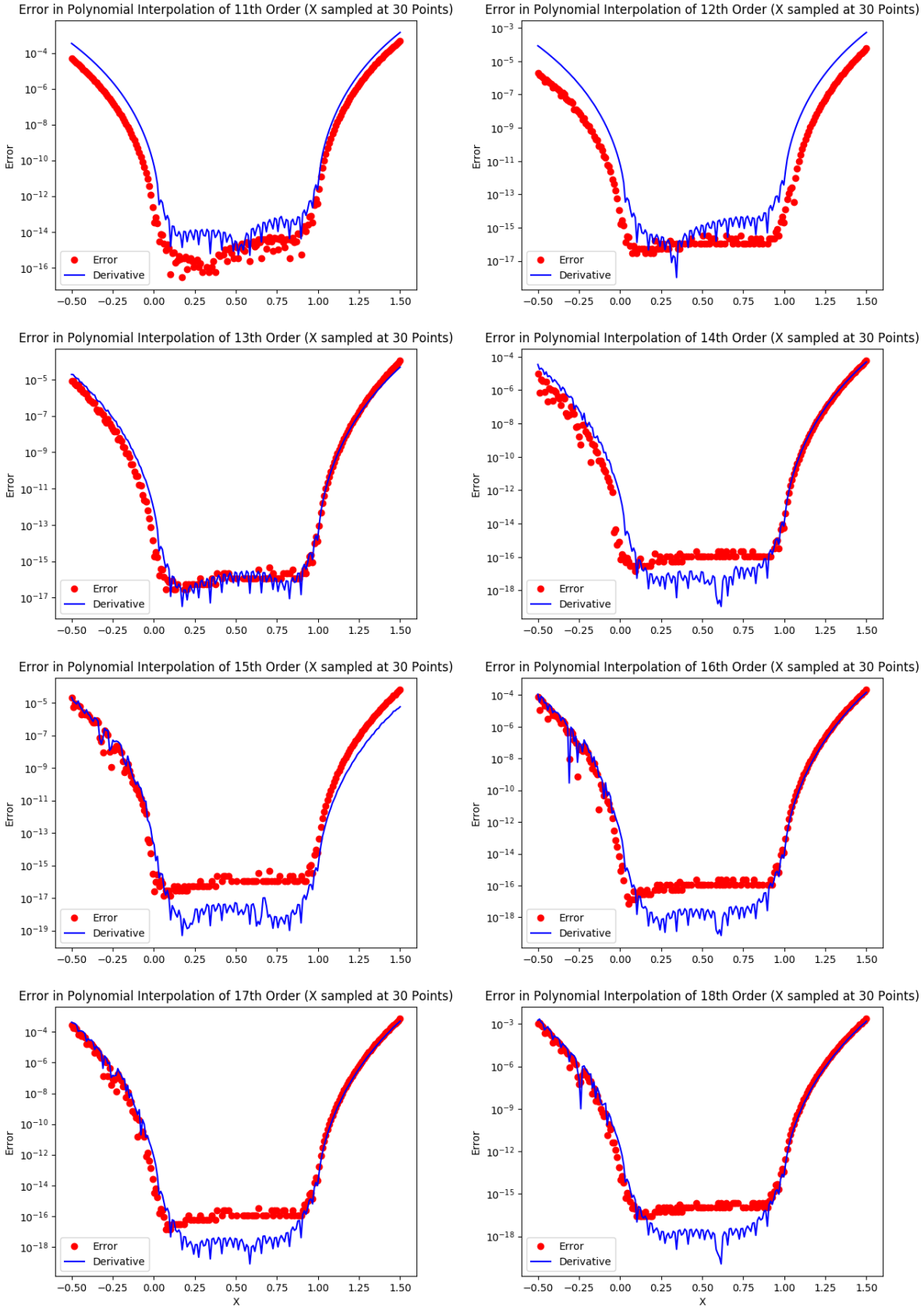

Error in Polynomial Interpolation of 6th Order (X sampled at 30 Points)
Error
Derivative
Error


Error in Polynomial Interpolation of 7th Order (X sampled at 30 Points)
Error
Derivative
Error


Error in Polynomial Interpolation of 8th Order (X sampled at 30 Points)
Error
Derivative
Error


Error in Polynomial Interpolation of 9th Order (X sampled at 30 Points)
Error
Derivative
X


Error in Polynomial Interpolation of 10th Order (X sampled at 30 Points)
Error
Derivative
X



We can observe that the error consistently reduces with increasing the order of interpolation. We know that, for an n^{th} order polynomial fit of a function, the error will be of the order $(n+1)$, i.e.,

as the order of interpolation increases, we obtain a better fit of the sampled data. Therefore, the error incurred reduces, as is evident from the plots shown above.

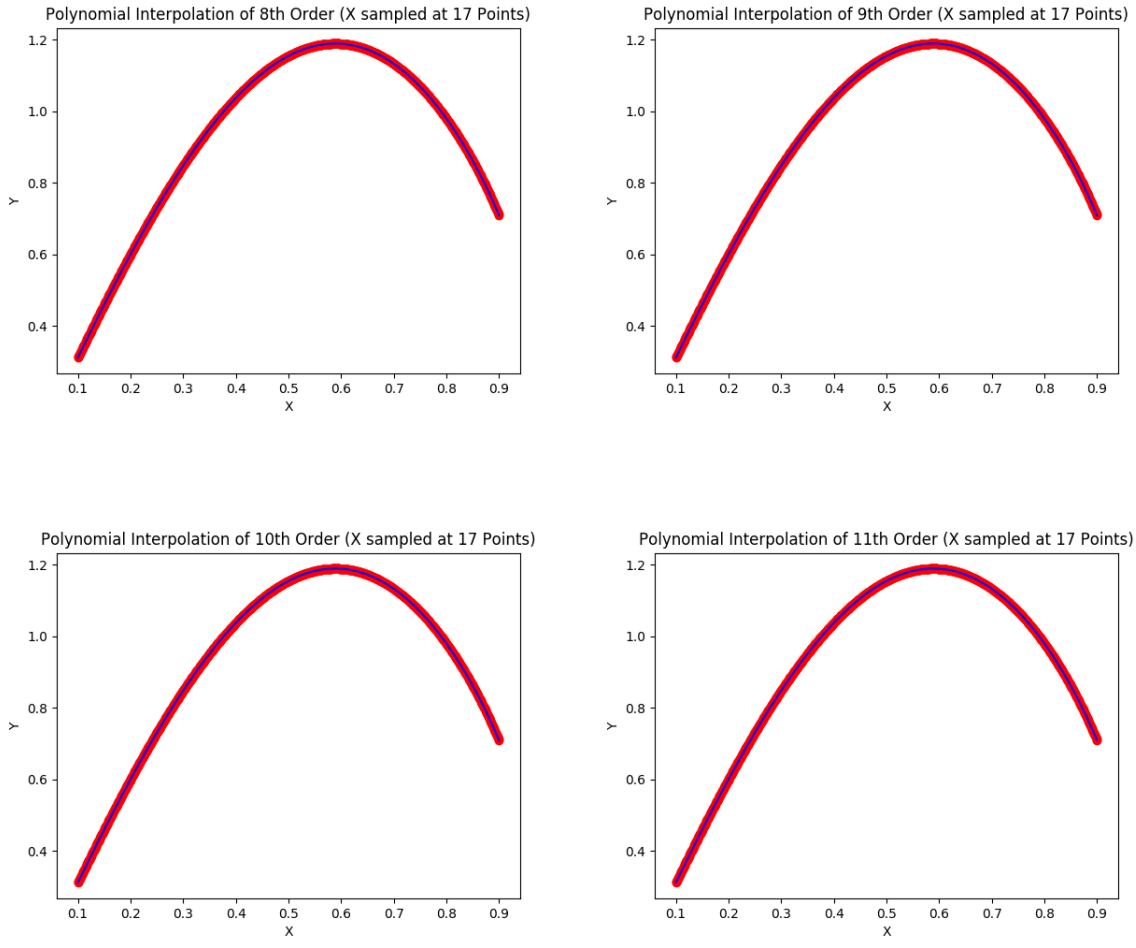
Question 5

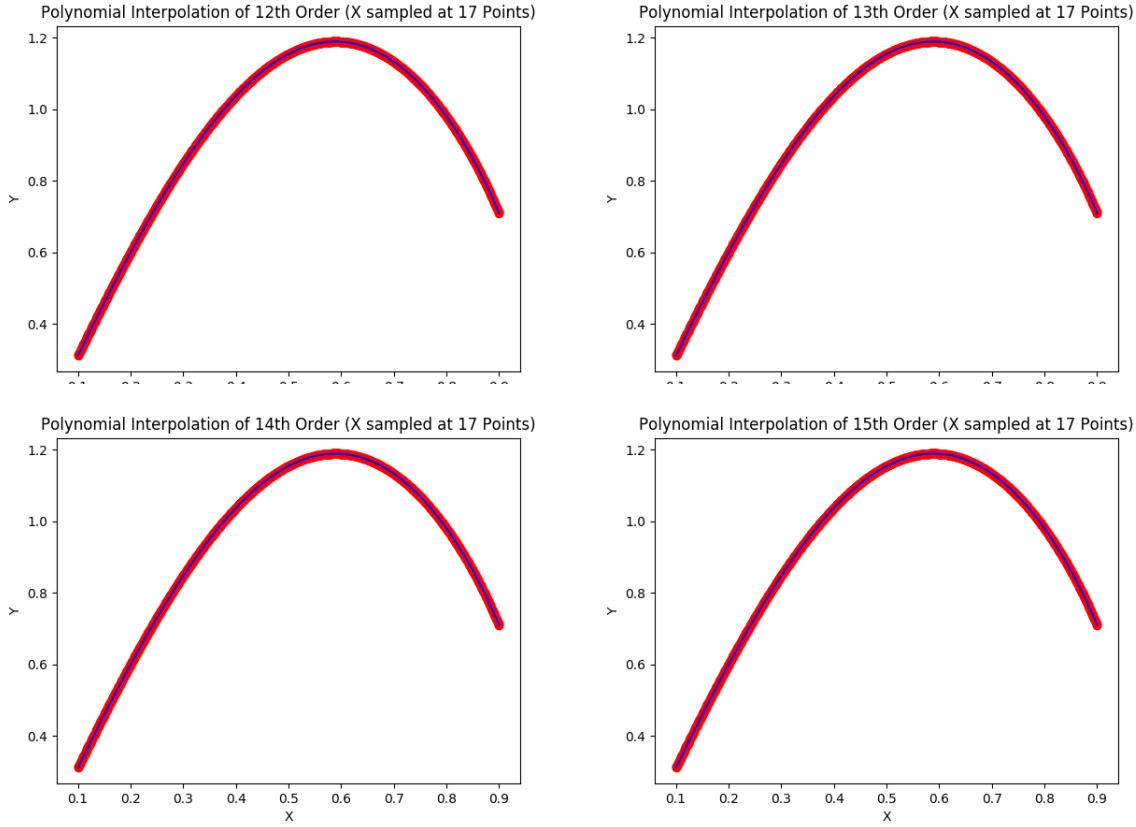
We perform the interpolation of the following function $f(x)$, in order to obtain a 6-digit accurate representation of the function.

$$f(x) = \frac{sin(\pi x)}{\sqrt{1-x^2}}$$

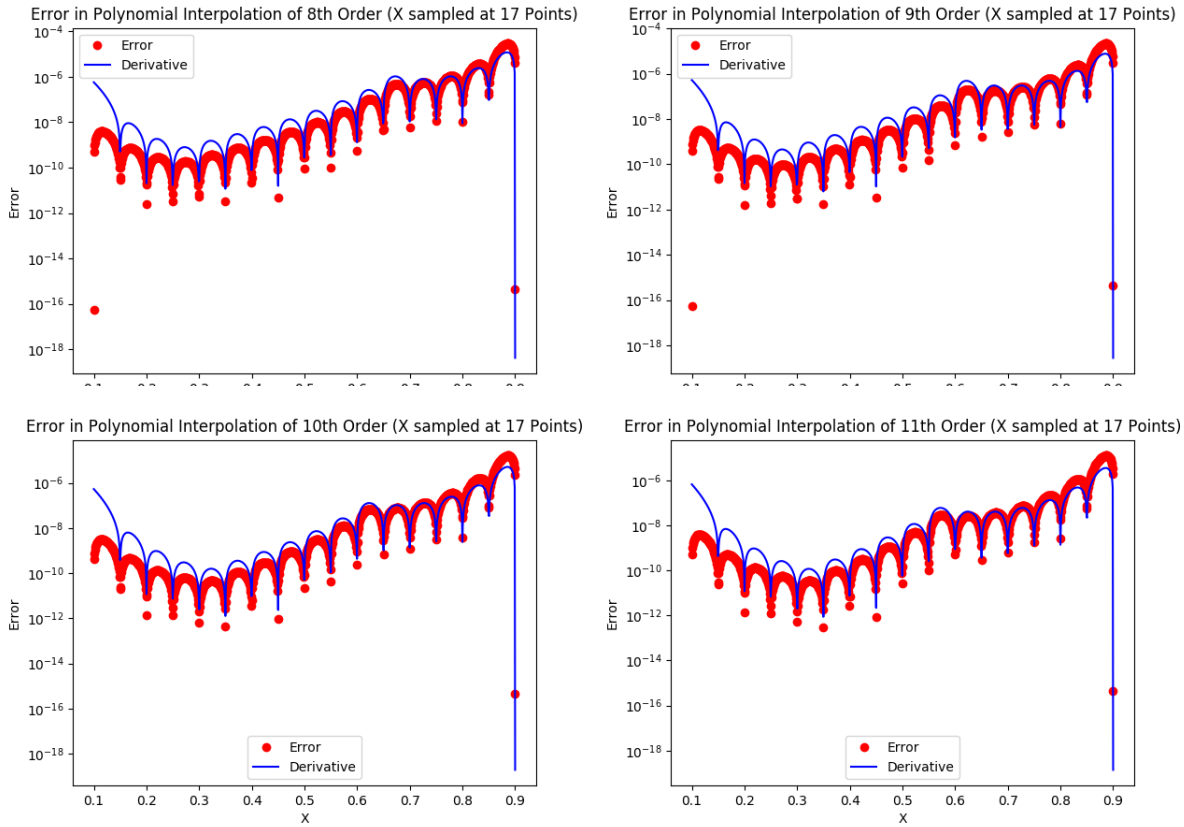
The function is sampled between $x = 0.1$ and $x = 0.9$, with a spacing of 0.05 i.e., 17 samples. We then perform *polynomial interpolation* across various orders and observe what orders of interpolation provide us with an accuracy of 6-digits (an error less than 0.00001, i.e., 10^{-5}).

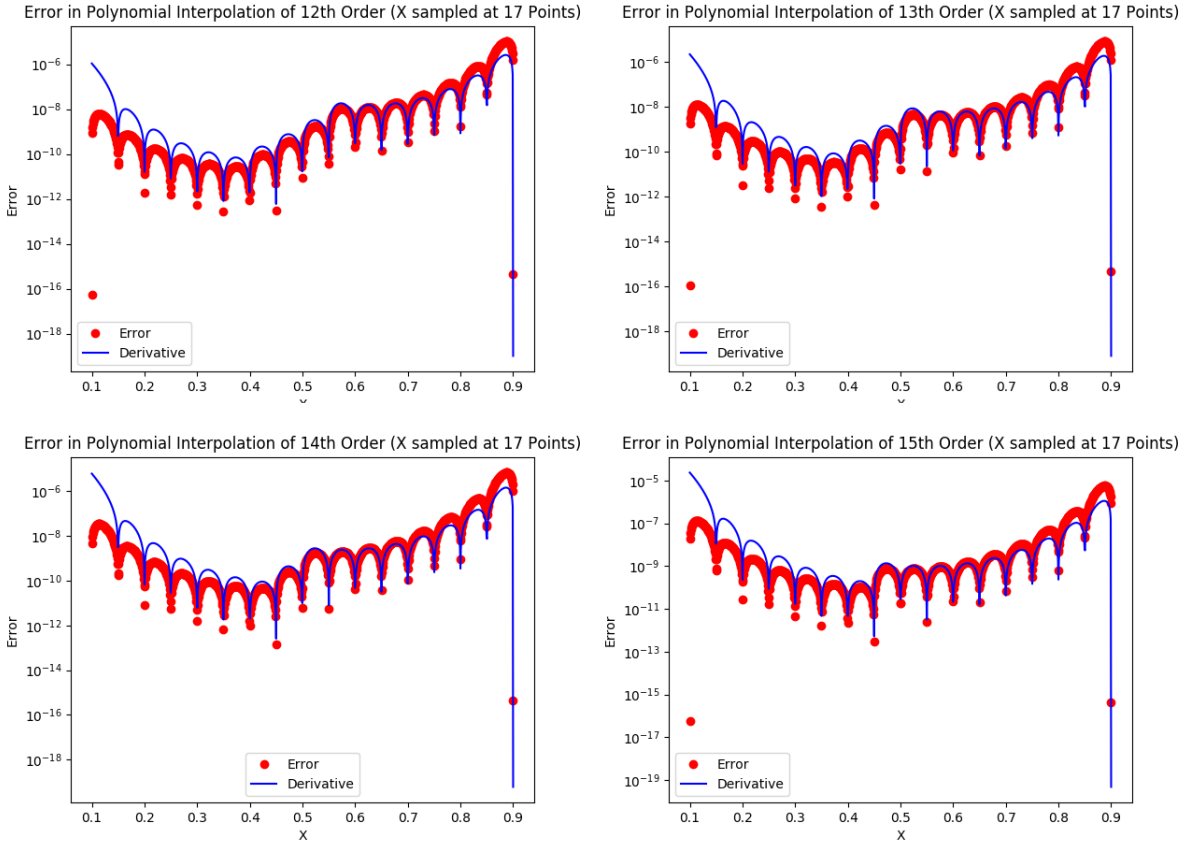
We vary the order of interpolation as $8, 9, ..., 14, 15$. The following plots are then obtained:





The *average errors* obtained in interpolation across these various orders is also plotted as follows:





The given function $f(x)$ is real and well-defined in the interval of $(-1, 1)$. It is undefined at the points $x = \pm 1$, and is imaginary elsewhere. Therefore, $f(x)$ is analytic on all points in $(-1, 1)$ and the function has a radius of convergence $(R) = 1$, i.e., the function is differentiable everywhere in the region $(-1, 1)$ and can be locally represented by a convergent power series at any point inside this region.

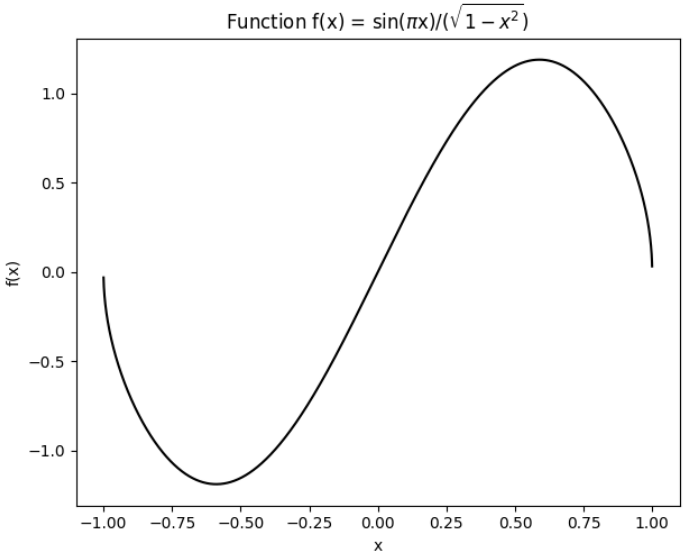


Figure 6: $f(x) = \frac{sin(\pi x)}{\sqrt{1-x^2}}$

Behaviour of $f(x)$ near $x = \pm 1$

The function $f(x) = \frac{sin(\pi x)}{\sqrt{1-x^2}}$ is undefined at $x = \pm 1$, because the denominator becomes zero. But, as x approaches ± 1 ($x \to \pm 1$), the numerator, i.e., $sin(\pi x)$ tends to zero ($sin(\pi x) \to 0$). Therefore, the behaviour of the function $f(x)$ near $x = \pm 1$ can be analysed by computing the *limit* of the function as it approaches $x = \pm 1$.

$$\lim_{x \to \pm 1} f(x) = \lim_{x \to \pm 1} \frac{sin(\pi x)}{\sqrt{1-x^2}} = 0$$

Order of Interpolation for 6-digit Accuracy

While interpolating the function $f(x) = \frac{sin(\pi x)}{\sqrt{1-x^2}}$, we vary the order of interpolation from 8,9,...,14,15 and obtain the *maximum error* for each order of interpolation. We observe that an *order of interpolation* (n) ≥ 13 will provide us with 6-digit accuracy (an error less than 0.00001, i.e., 10^{-5}) within the region of $(-1, 1)$.

Conclusion

- We studied the relationship between the order of interpolation and the accuracy of interpolation, i.e., we observed that with an increase in the order of interpolation, the accuracy is improved (error is minimized).
- We also studied the trend between the sampling rate (number of points at which the function is sampled) and the accuracy obtained in interpolation.

***

Appendix

```
from scipy import *
from matplotlib.pyplot import *
import weave

def PolyInt(xarr, yarr, xx, n):
    M = len(xarr)
    N = len(xx)
    c = zeros((n+1,1))
    d = zeros((n+1,1))
    yy = zeros(xx.shape)
    dyy = zeros(xx.shape)

    code = """ Insert C Implementation of Polynomial Interpolation here """

    weave.inline(code, ['xarr', 'yarr', 'M', 'xx', 'yy', 'dyy', 'N', 'c', 'd', 'n'],
     compiler = 'gcc')
    return ([yy, dyy, xx])

def SINXX2(LENX,XX,N):
    X = linspace(0,1,LENX)
    AVGERR = []
    MAXERR = []

    for n in N:
        [YY, dYY, XX] = PolyInt(X, sin(X + X**2), XX, int(n))
        Y0 = sin(XX + XX**2)

        AVGERR.append(mean(abs(YY - Y0)))
        MAXERR.append(max(abs(YY - Y0)))
        print("Average & Maximum Error in Polynomial Interpolation of %dth Order (X
    sampled at %d Points): %f and %f" % (n, len(X), mean(abs(YY - Y0)), max(abs(YY -
     Y0))))

        plot(XX, YY, 'ro', XX, Y0, 'b')
        title('Polynomial Interpolation of %dth Order (X sampled at %d Points)' % (n
    , len(X)))
        xlabel('X')
        ylabel('Y')
        #savefig('PolyIntOrder%dX%d.png' % (n, len(X)))
        show()
        semilogy(XX, abs(YY - Y0), 'ro', XX, abs(dYY), 'b')
        title('Error in Polynomial Interpolation of %dth Order (X sampled at %d
    Points)' % (n, len(X)))
        xlabel('X')
        ylabel('Error')
        legend(['Error', 'Derivative'])
        #savefig('PolyIntErrorOrder%dX%d.png' % (n, len(X)))
        show()
```

```
    return ([AVGERR, MAXERR])

# QUESTION 1
n = [4] # Order of Polynomial Interpolation
LENX = 5    # Number of points to sample X at
XX = linspace(-0.5,1.5,200)
[A1, M1] = SINXX2(LENX,XX,n)
print("Average Error in Polynomial Interpolation of %dth Order (X sampled at %d
    Points): %f" % (n[0], LENX, A1[0]))
print("Maximum Error in Polynomial Interpolation of %dth Order (X sampled at %d
    Points): %f" % (n[0], LENX, M1[0]))

# QUESTION 2
n = [4] # Order of Polynomial Interpolation
LENX = 30   # Number of points to sample X at
XX = linspace(-0.5,1.5,200)
[A2, M2] = SINXX2(LENX,XX,n)
print("Average Error in Polynomial Interpolation of %dth Order (X sampled at %d
    Points): %f" % (n[0], LENX, A2[0]))
print("Maximum Error in Polynomial Interpolation of %dth Order (X sampled at %d
    Points): %f" % (n[0], LENX, M2[0]))

# QUESTION 3 & 4
n = array([3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]) # Order of Polynomial
    Interpolation
LENX = 30   # Number of points to sample X at
XX = linspace(-0.5,1.5,200)
[A3, M3] = SINXX2(LENX,XX,n)
FIG, AX = subplots()
AX.plot(n, A3, 'r', label = 'Average Error')
AX.plot(n, M3, 'b', label = 'Maximum Error')
AX.set_title('Average and Maximum Error in Polynomial Interpolation (X sampled at %d
     Points)' % (LENX))
AX.set_xlabel('Order of Polynomial Interpolation')
AX.set_ylabel('Error')
AX.legend()
#savefig('AvgMaxErrorPolyIntX%d.png' % (LENX))
show()

# QUESTION 5
n = array([8,9,10,11,12,13,14,15]) # Order of Polynomial Interpolation
X = arange(0.1,0.95,0.05)
Y = sin(X*pi)/(sqrt(1-X**2))
XX = linspace(0.1,0.9,1000)
MAXERR = []

for i in n:
    [YY, dYY, XX] = PolyInt(X, Y, XX, i)
    Y0 = sin(XX*pi)/(sqrt(1-XX**2))
```

```
    print("Maximum Error in Polynomial Interpolation of %dth Order (X sampled at %d
    Points): %f" % (i, len(X), max(abs(YY - Y0))))
    MAXERR.append(max(abs(YY - Y0)))
    plot(XX, YY, 'ro', XX, Y0, 'b')
    title('Polynomial Interpolation of %dth Order (X sampled at %d Points)' % (i,
    len(X)))
    xlabel('X')
    ylabel('Y')
    #savefig('Q5PolyIntOrder%d.png' % (i))
    show()
    semilogy(XX, abs(YY - Y0), 'ro', XX, abs(dYY), 'b')
    title('Error in Polynomial Interpolation of %dth Order (X sampled at %d Points)'
    % (i, len(X)))
    xlabel('X')
    ylabel('Error')
    legend(['Error', 'Derivative'])
    #savefig('Q5PolyIntErrorOrder%d.png' % (i))
    show()

for err in MAXERR:
    if (err < 0.00001):
        print("Order of Polynomial Interpolation that produces less than 0.000001
    error: %d" % (n[MAXERR.index(err)])); break;
```