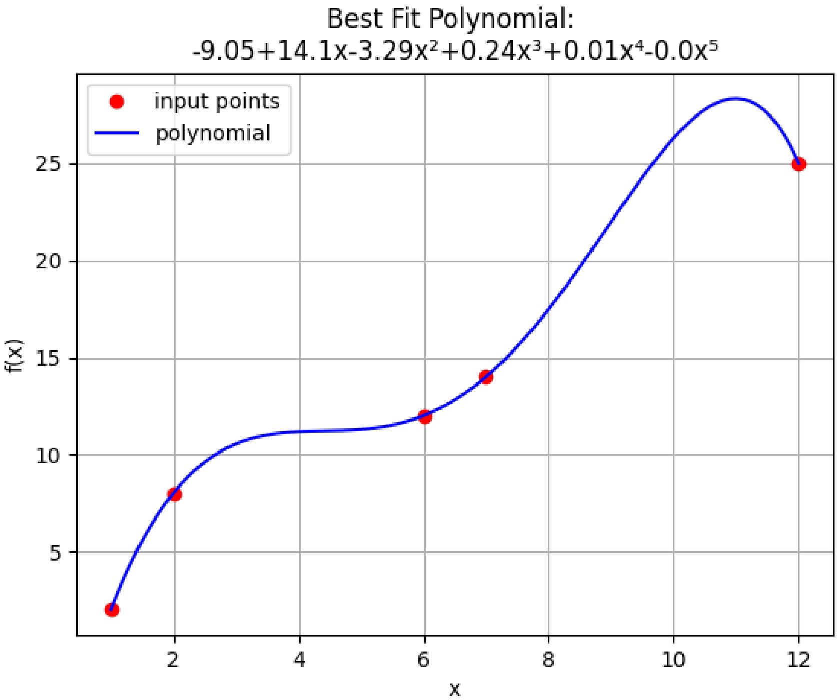
Coding Assignment 5
Least-Square Function Approximations

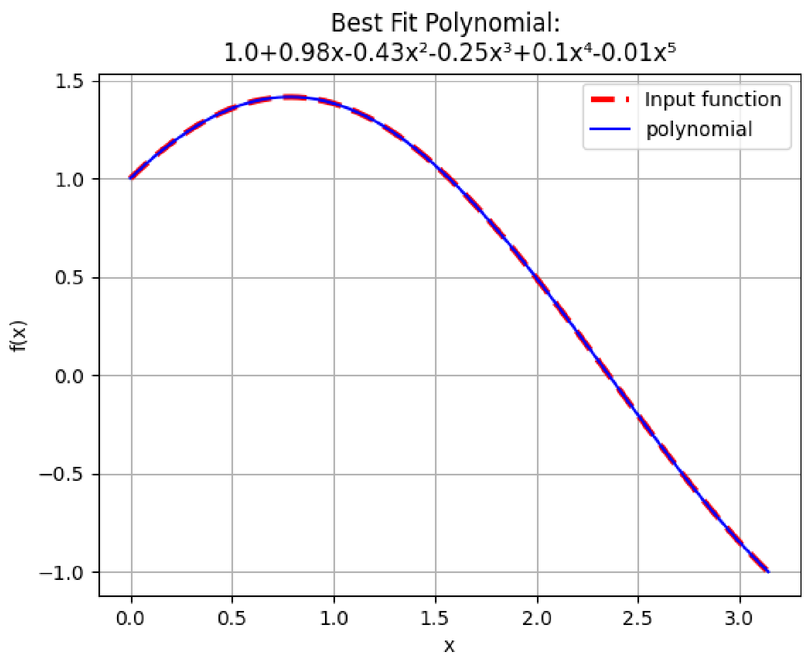
-Yukta Salunkhe
-112001052

Q1.Approach mentioned in comments
=>



<u>Test case1:</u>
BestFitPolynomial of degree 5 that best fits for the points : ([(1, 2), (2, 8), (6, 12), (7, 14), (12, 25)]) is plotted above.

Q2.Approach mentioned in comments
=>
<u>Test case1:</u>
BestFitPolynomial of degree 5 for f(x) = sin(x)+cos(x) in the interval [0,pi] is plotted below.



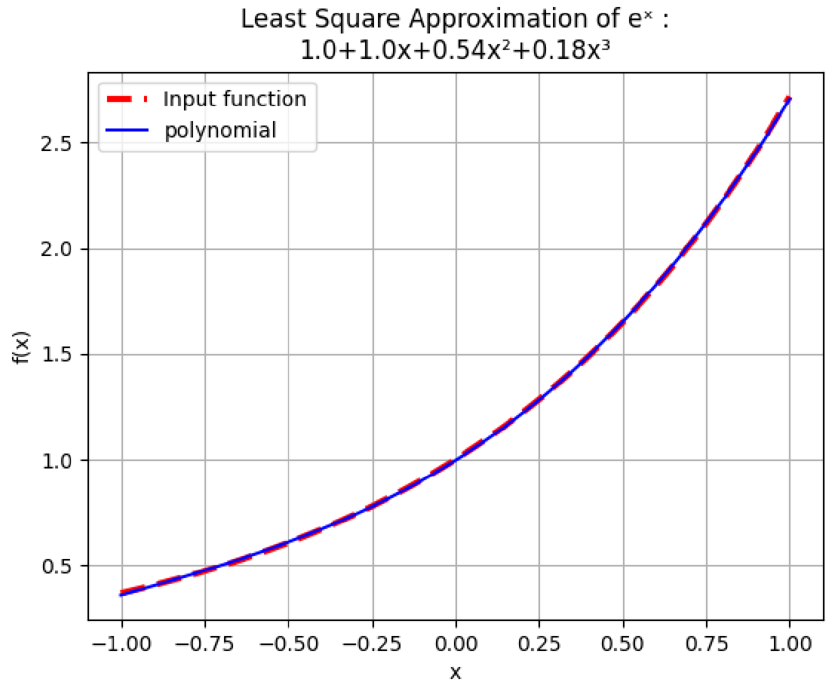
Q3.Approach mentioned in comments
=>
<u>TestCase1:</u>
computeNthLegendrePolynomial(0)
Coefficients of the polynomial are:
1.0

<u>TestCase2:</u>
computeNthLegendrePolynomial(1)
Coefficients of the polynomial are:
0.0 1.0

<u>TestCase3:</u>
computeNthLegendrePolynomial(2)
Coefficients of the polynomial are:
-0.5 0.0 1.5

<u>TestCase4:</u>
computeNthLegendrePolynomial(3)
Coefficients of the polynomial are:
0.0 -1.5 0.0 2.5

Q4.Approach mentioned in comments
=>



Least Square Approximation for function e^x is plotted above along with the actual function. Both the functions overlap as is visible above.

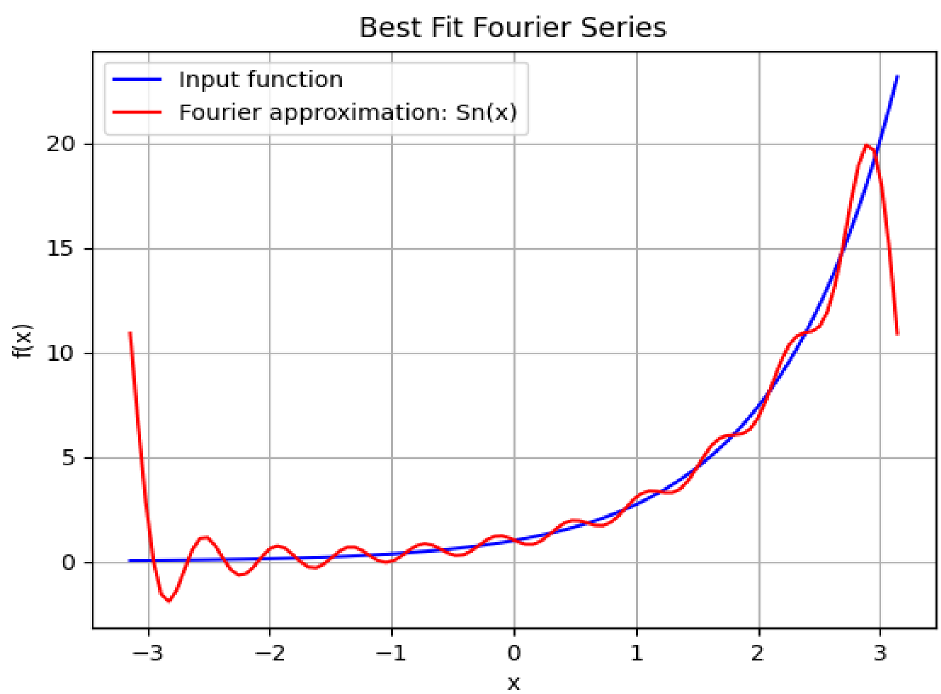
Q5.Approach mentioned in comments
=>
<u>TestCase1:</u>
computeNthChebyshevsPoly(0)
Coefficients of the polynomial are:
1

<u>TestCase2:</u>
computeNthChebyshevsPoly(1)
Coefficients of the polynomial are:
0 1

<u>TestCase3:</u>
computeNthChebyshevsPoly(2)
Coefficients of the polynomial are:
-1 0 2

Q6.Approach mentioned in comments
=> When we calculate the integral of product of pairs of chebyshev's polynomials (considered 5) with weight function $w(x) = 1/\sqrt{(1 - x^2)}$ we get a diagonal matrix with all the non-diagonal elements as 0. Thus it depicts that the Chebyshev polynomials are numerically orthogonal with respect to the weight function w(x).

Q7.Approach mentioned in comments
=>



Plot for Best Fit Fourier Approximation S10(x) for the function e^x in the interval [−π, π] is as above.

The Coefficients of S10 (x):
a0 : 7.352155820749955 b0 : 0.0
a1 : -3.6760779103749774 b1 : 3.6760779103749766
a2 : 1.4704311641499912 b2 : -2.9408623282999806
a3 : -0.7352155820749959 b3 : 2.205646746224986
a4 : 0.43247975416176176 b4 : -1.7299190166470475
a5 : -0.28277522387499693 b5 : 1.4138761193749905
a6 : 0.19870691407432362 b6 : -1.1922414844459392
a7 : -0.1470431164149997 b7 : 1.0293018149049924
a8 : 0.11311008954999849 b8 : -0.9048807163999935

a9 : -0.08966043683841039 b9 : 0.8069439315457274
a10 : 0.07279362198762257 b10 : -0.7279362198762331

Q8.Approach mentioned in comments
=>
<u>TestCase1:</u>

For a = 9322356, b = 8922002:

Actual Product: 83174078876712
Product calculated using Fast Fourier Transformation is: 83174078876711.97

As seen in this example, both give approximately the same ans. Thus within time complexity of O(n log n), product of large numbers can be found using Python's `scipy.fft` package.