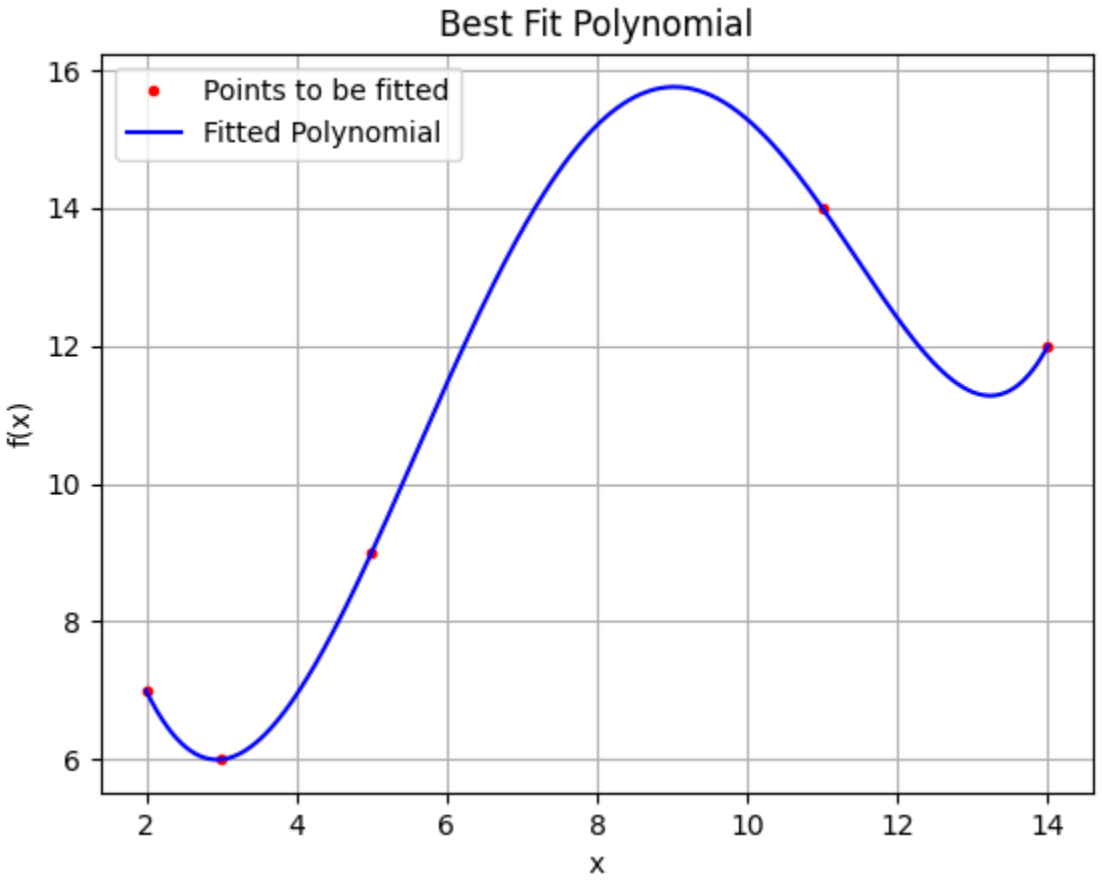
CS5016: Computational Methods and Applications
Assignment 5: Least-Square Function Approximations

Sujit Mandava
112001043

Q1. We build on the previously created Polynomial class. We create a new method bestFitPolynomialPoint in the class. It takes two arguments, points and n. Points is a list of points for which we have to find the best fitting polynomial (with least error). n dictates the maximum degree the best fitting polynomial can have.

$$(\Sigma^{n}_{k=0} a_k) \ * \ (\Sigma^{m}_{i=1} x_i^{j+k}) \ = \ (\Sigma^{m}_{i=1} y_i x_i^{j}) \ \ \forall j \in \{0, 1, \ldots, n\}$$

Assuming the first term is a, second S and the third to be b, we compute S and b by fixing a k for every value a_k. We then get a matrix S of dimension $(n + 1) * (n + 1)$ and a vector b of dimension $(n + 1)$. We solve the algebraic equation $Sa = b$ to get the coefficients a that have the least error when fit to the given points.
For the points (2,7), (3,6), (14,12), (11,14), and (5,9), we get the following graph.

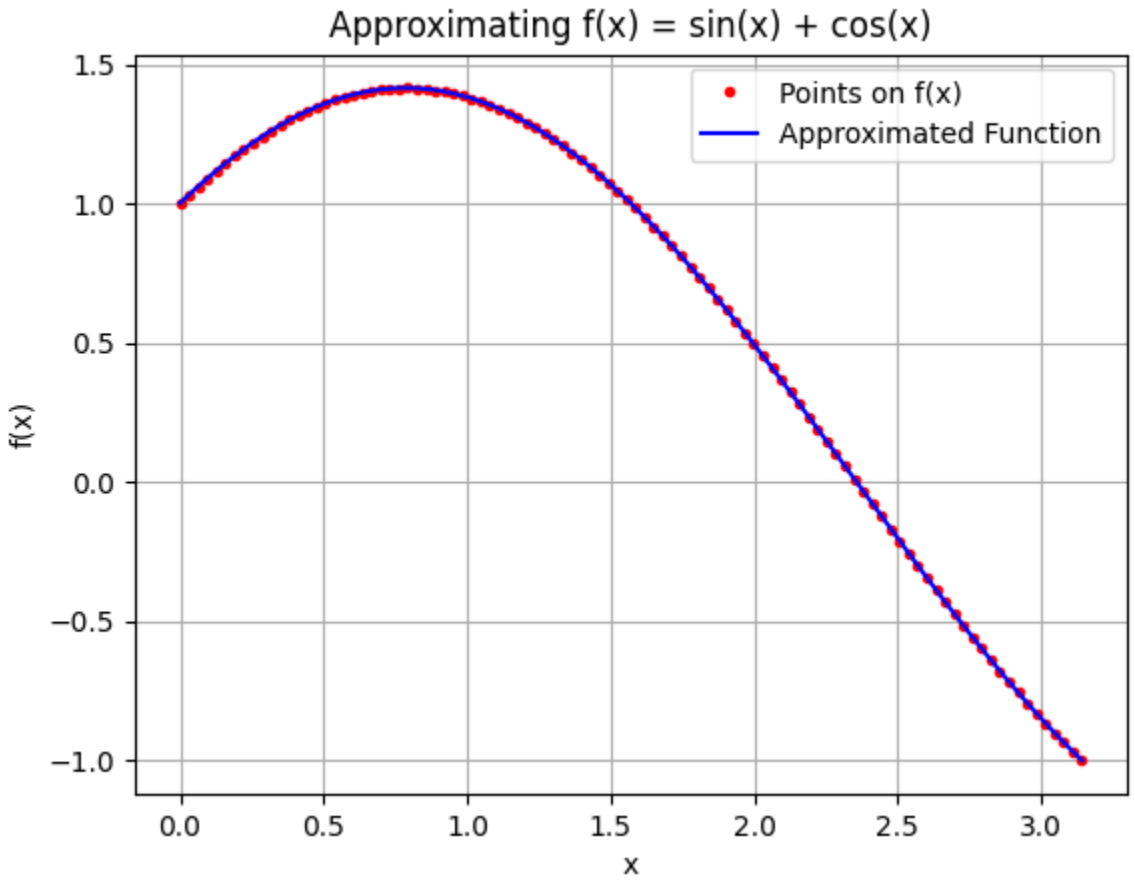


Q2. We build on the previously created Polynomial class. We create a new method bestFitPolynomialInterval in the class. It takes one argument n. It dictates the maximum degree the best fitting polynomial can have. The task here is to find the polynomial that best approximates the function $f(x) = sin(x) + cos(x)$ in the range $[0, \pi]$.

The principle is applied to this problem as well. We solve the following normal equations:

$$(\Sigma_{k=0}^{n} a_k)(\int_0^{\pi} x^{j+k}) = (\int_0^{\pi} x^{j} f(x)) \quad \forall j \in \{0, 1, \ldots, n\}$$

Here, the first term is a, second S and the third is . We compute S and b by fixing a k for every value a_k. We then get a matrix S of dimension $(n + 1) * (n + 1)$ and a vector b of dimension $(n + 1)$. We solve the algebraic equation $Sa = b$ to get the coefficients a that have the least error when fit to the given function. We get the following graph after plotting the approximation and the actual function:

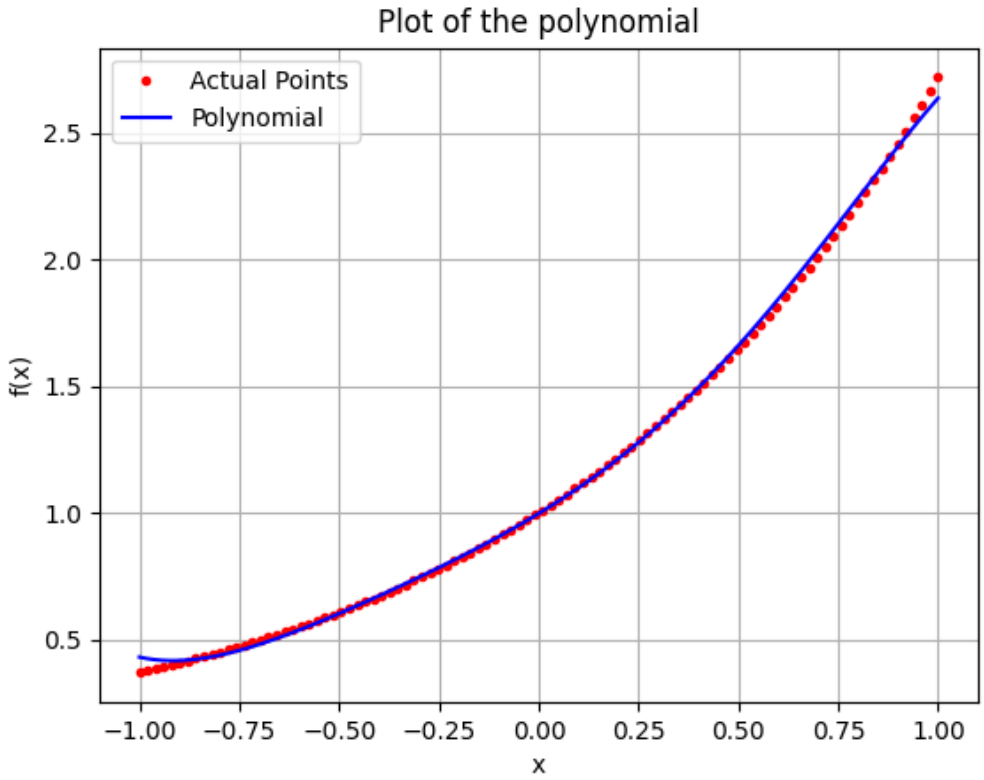


Q3. We add a new method to the Polynomial class, nLegendre. It takes one argument n and returns an object of the Polynomial class representing the nth Legendre polynomial. Legendre polynomials can be computed using the following equation:

$$L_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} (x^2 - 1)^n$$

We simply calculate the value of $L_n(x)$ for the given n and return the result as an object of the polynomial class.

Q4. We use the method created in Q3 to approximate the function e^x in the range $[-1, 1]$ using the first n Legendre polynomials. The new method estimateExp takes one argument, n which gives us the number of Legendre polynomials to be used for the approximation. Then, we use the formula to compute the approximate value of a function using orthogona; polynomials. By plotting the original points against the approximations at each x, we get



Q5. We add a new method to the Polynomial class called nChebyshev. It takes one argument, n which is a non-negative integer and returns the nth Chebyshev polynomial. They are computed using the following expression:

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

$$T_n(x) = cos(ncos^{-1}(x))$$

Using these relations, we compute the value of $T_n(x)$.

Q6. We create a new method to verify the orthogonality of the first 5 Chebyshev polynomials with respect to the weight function $w(x) = \frac{1}{\sqrt{1-x^2}}$ in the range [-1,1]. We first compute the first 5 polynomials using the above mentioned method. Then we create a matix consisting of all values m_{ij} that computes $\int_{-1}^{1} w(x)\phi_i(x)\phi_j(x)dx$ for all i, j. If the function is orthogonal, then the

matrix obtained is a diagonal matrix. Upon computing the required values, we get the following output:

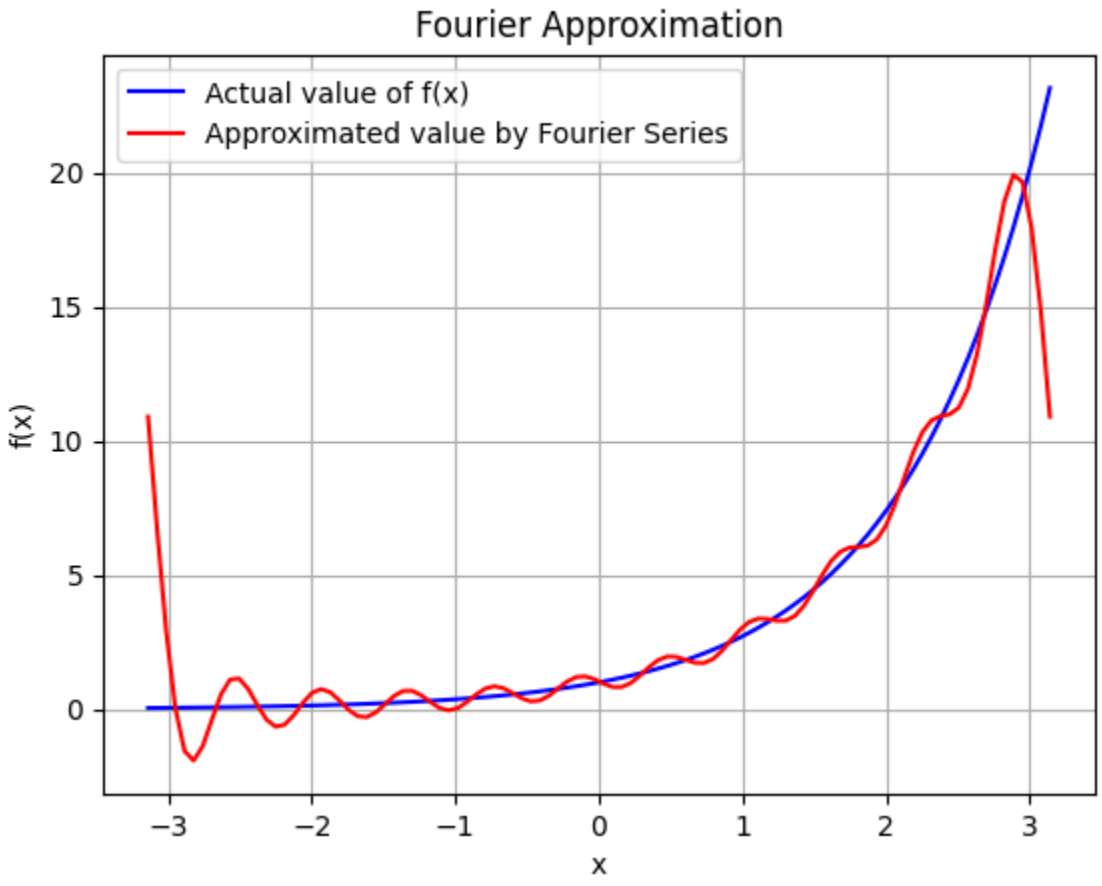
```
[3.141592653589591, 0, 0, 0, 0]
[0, 1.5707963267946803, 0, 0, 0]
[0, 0, 1.5707963267948821, 0, 0]
[0, 0, 0, 1.5707963267927536, 0]
[0, 0, 0, 0, 1.5707963267929148]
```

As expected, the matrix is orthogonal and thus orthogonality is verified.

Q7. We add a new method fourierApproximation to use the Fourier series and approximate the value of e^x in the range $[-\ \pi,\ \pi\ ]$. It takes one optional argument n that tells us the number of terms from the series we should consider for the approximation. The default value is set to 10.

$$S_n(k) = \frac{a_0}{2} + \sum_1^n a_k cos(kx) + \sum_1^n b_k cos(kx)$$

Using the above equation, we compute the approximation of the function, represented by $S_n(x)$. We calculate the approximations of the function for 100 values of x and plot the generated function against the original. The graph obtained is:



Q8. We use the scipy.fft module to enable fast computation of the previously mentioned a_k and b_k. We convert a each number into a list of digits and pad the smaller number with zeroes so both numbers have the same digits. Then, we compute each list's 1D Fourier Transform and multiply the two. The resultant Fourier Transform is inverted back into a number. The output of this function would look like:

```
A = 31220334
B = 442721
Computed product using fft: 13821897488814.016
Actual Product: 13821897488814
```