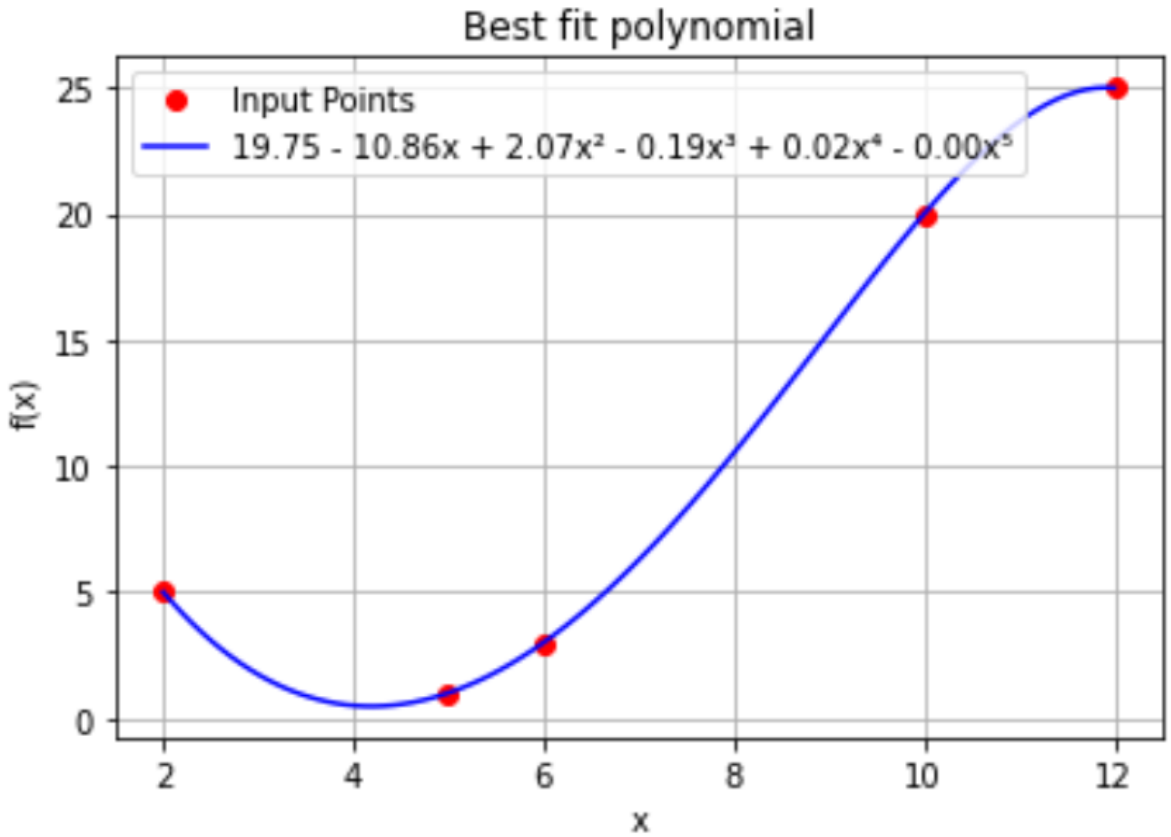
CS5016: Computational Methods and Applications

Lab Report - 5

111901030
Mayank Singla

Q1.

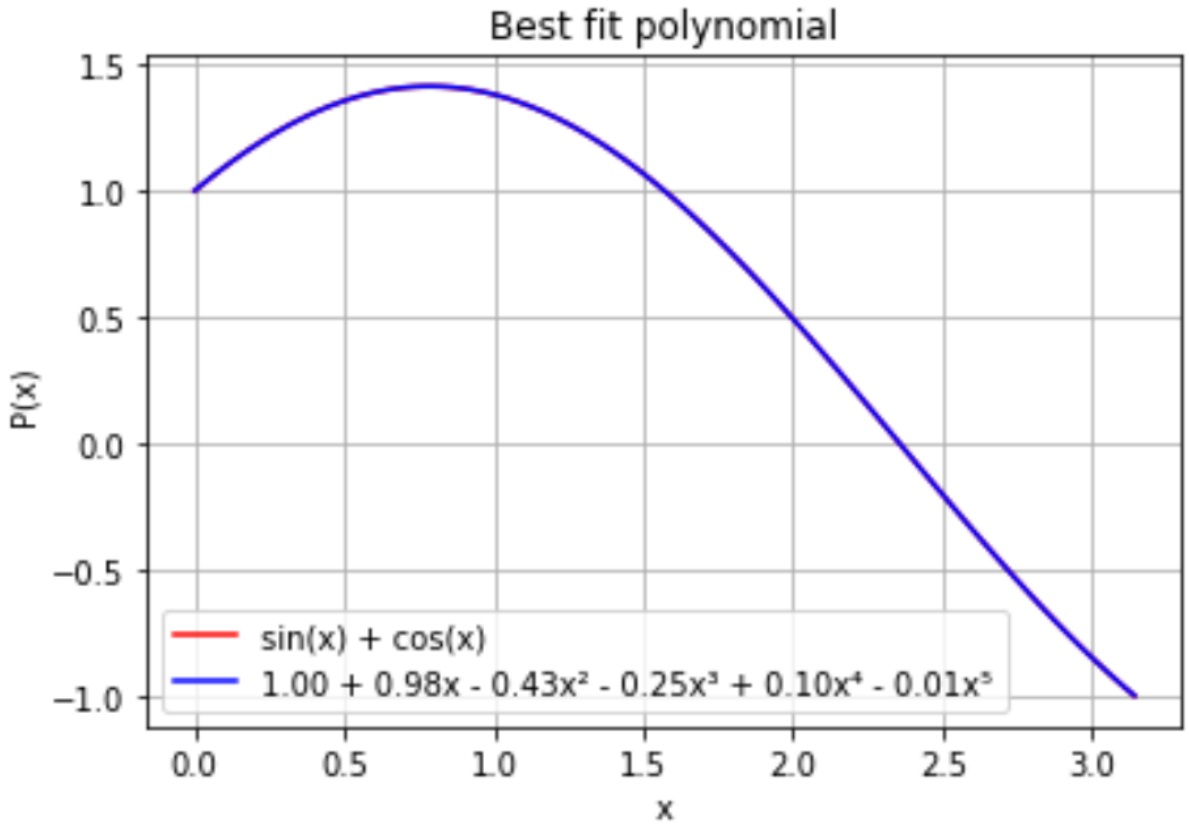
I am creating the function `bestFitPoints` To find the best fit points for the input points, I am using the normal form of equations that we derived in the lectures for the polynomial and representing it as the solution to the system of linear equations SA = b. Hence, I am computing the vector b and S first from the formula and then using the `np.linalg.solve` function to compute the vector A which are the final coefficients of the polynomial. Then I am simply plotting all the required things.
Here is an example output I obtained.



Q2.

I am creating the function `bestFitFunction`. Again using the same approach as before in the previous problem and using the formulas for the normal equations derived in theory lectures, I am here again computing the vectors S and b first from the input function and then computing the vector A using the `np.linalg.solve` function. Here, I am using `scipy.integrate.quad` function to evaluate the integral of the function. Then I am simply plotting all the required things. I am creating it as a general function that will take any function as input and compute the best fit for that.

Here is an example output I obtained.



Q3.

First of all, I am here overloading the `Polynomial` class with the `__pow__` function that is the exponential operator to compute the nth power of the polynomial. Then I am creating a function `computeNthLegendrePoly` to find the nth Legendre polynomial. I am simply using the formula for the polynomial by making use of the overloaded exponential function and the derivative function to compute the polynomial.

Here is an example output I obtained.

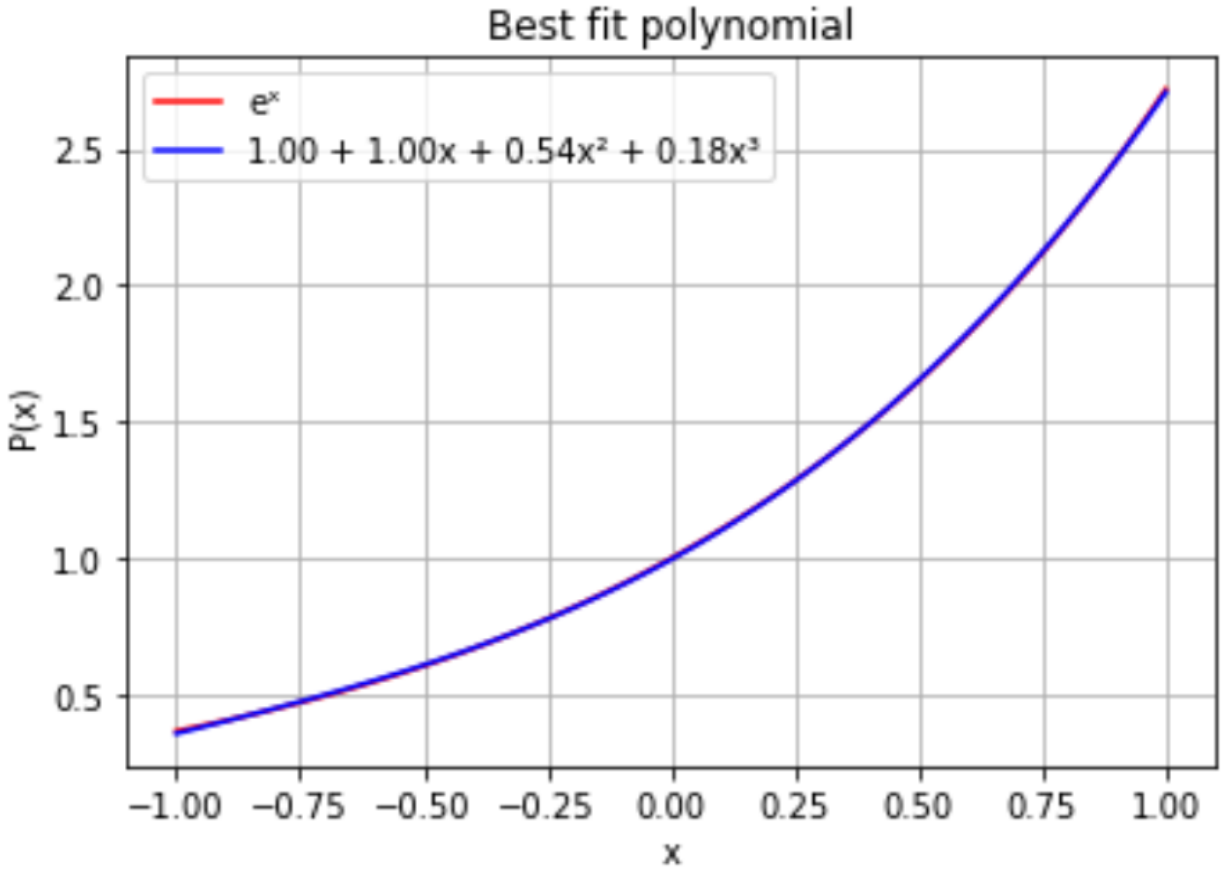
```python
# First Legendre Polynomial
l0 = p.computeNthLegendrePoly(0)
print(l0)

# Second Legendre Polynomial
l1 = p.computeNthLegendrePoly(1)
print(l1)

# Third Legendre Polynomial
l2 = p.computeNthLegendrePoly(2)
print(l2)
```

```
Coefficients of the polynomial are:
1.0
Coefficients of the polynomial are:
0.0 1.0
Coefficients of the polynomial are:
-0.5 0.0 1.5
```

Q4.

I am creating the function `bestFitLegendrePoly` to compute the best fit polynomial for the input function using first n Legendre polynomials. I am creating a nested function `w` to compute the weight function of the polynomial. Then I am first calculating and storing all the n Legendre polynomials and then I am generating the list of the coefficients of the resulting polynomial by using the standard formula derived in the theory lectures for the orthogonal polynomials and then generating the final polynomial and returning its value from the function. I am also plotting all the required things. I am

creating it as a general function that will take any function as input and compute the best fit for that.
Here is an example output I obtained.



Q5.
Here simply using the standard recursive function and the overload operators of the Polynomial class, I am creating a function computeNthChebyshevPoly to compute the nth Chebyshev polynomial using simple for loop and formula.
Here is an example output I obtained.

```
t0 = p.computeNthChebyshevPoly(0)
print(t0)

t1 = p.computeNthChebyshevPoly(1)
print(t1)

t2 = p.computeNthChebyshevPoly(2)
print(t2)
```

```
Coefficients of the polynomial are:
1
Coefficients of the polynomial are:
0 1
Coefficients of the polynomial are:
-1 0 2
```

Q6.

I am creating the function `demonstrateOrthogonalityChebyshevPoly` to demonstrate the orthogonality of the Chebyshev polynomials. I am simply computing all the Chebyshev polynomial and defining a nested function `w` to compute the weight function of the Chebyshev polynomial. Then simply using the standard definition of the orthogonality of the set of functions, I am evaluating the integral for all the possible pairs of polynomial functions in the form of a lower triangular matrix and displaying the final result.

Here is an example output I obtained for n = 5.

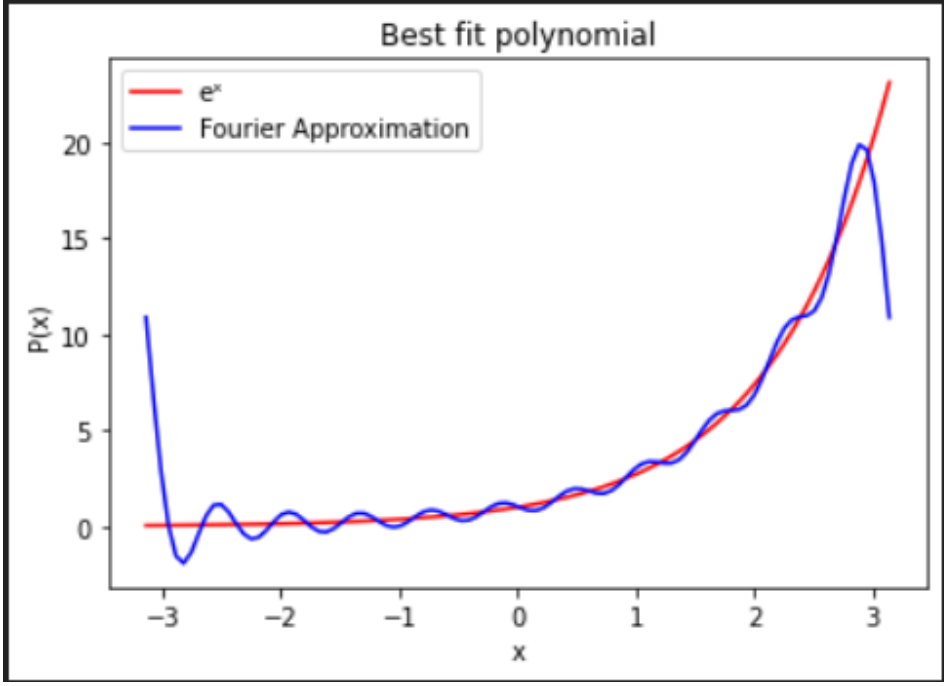
```
[[3.141592653589591], [0.0, 1.5707963267946803],
[-1.9864679913457918e-13, 0.0, 1.5707963267948821], [0.0,
6.349983952228556e-12, 0.0, 1.5707963267927536],
[-5.326059329524154e-12, 0.0, -2.1277398354660173e-12, 0.0,
1.5707963267929148]]
```

Here we can see, that whenever i is not equal to j, the number is of the form e-12 which is approximately equal to 0, and for other cases, it is a positive real number.

Q7.

I am creating the function `bestFitFourierSeries` to compute the best-fit Fourier approximation of the input function. I am creating it as a general method that will find the best fit for any input function. I am defining a nested function `w` to compute the weight function of the Fourier series. Then using the simple formula we saw in the theory lectures, I am computing the coefficients of the function using the `scipy.integrate` package to find the integration. Then after finding the coefficients I am using the formula for the Fourier transform to find the resultant value of the function for the sample points and plotted the required things.
Here is an example output I obtained.

```
a0 = 7.352155820749955, b0 = 0.0
a1 = -3.67607791037497774, b1 = 3.67607791037497774
a2 = 1.4704311641499912, b2 = -2.940862328299815
a3 = -0.7352155820749962, b3 = 2.2056467462249865
a4 = 0.43247975416176204, b4 = -1.72991901664704777
a5 = -0.28277522387499693, b5 = 1.413876119374991
a6 = 0.1987069140743238, b6 = -1.19224148444594
a7 = -0.14704311641499965, b7 = 1.0293018149049926
a8 = 0.113110089549998857, b8 = -0.9048807163999938
a9 = -0.08966043683841039, b9 = 0.8069439315457274
a10 = 0.072793621987622246, b10 = -0.727936219876233
```



Q8.

I am creating the function `computeProduct` that will take two integers as input and compute their product using the `scipy.fft` package. I am first making the input numbers in the polynomial form and then computing their FFT. Then I am multiplying the computes FFTs and take the inverse FFT of the product. Then I will obtain some complex quantities and their imaginary coefficients will be all near to 0. I am then representing those numbers in the base 10 notation and adding their real coefficients to get the final resultant product.

Here is an example output I obtained.

```
a = 123456789
b = 12345
computeProduct(a, b)
print(f"Actual Product: {a * b}")
```

```
Computed Product: 1524074060205
Actual Product: 1524074060205
```