CS5016: Computational Methods and Applications
Assignment 3: Linear System and Interpolation

Sujit Mandava
112001043

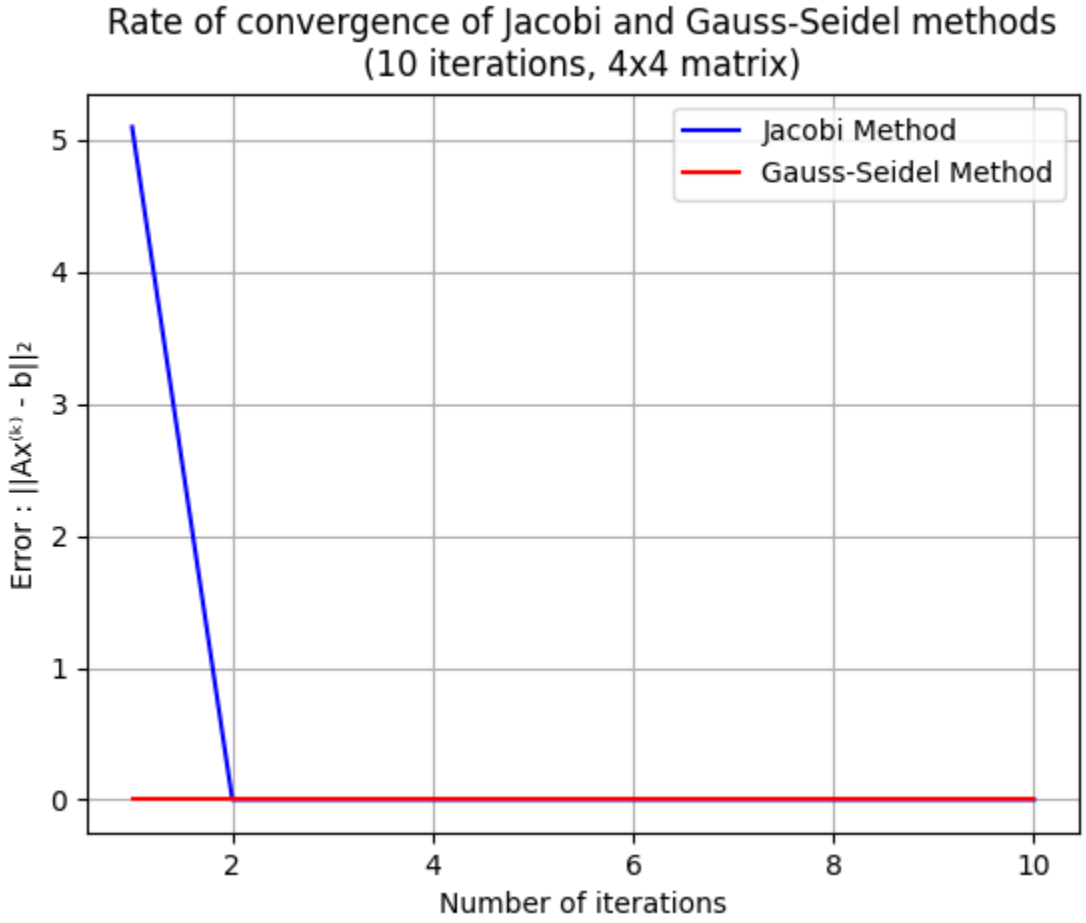
Q1. The class RowVectorFloat describes a mathematical row vector. It was implemented using a list. It has two attributes, the list of entries and the number of entries. The methods described in this class are:

1. __init__: Initializes a new row vector with the given list of values.
2. __str__: Converts the row vector into a string.
3. __len__: Returns the number of entries in the row vector.
4. __getitem__: Returns the value at the given index in the row vector.
5. __setitem__: Sets the value at the given index in the row vector.
6. __add__: Adds two row vectors element-wise and returns the result as a row vector. If the lengths are not equal, it raises an exception.
7. __rmul__, __mul__: Multiplies the row vector by a scalar and returns the result as a row vector. __rmul__ is called when the scalar is being multiplied to the right of the vector, whereas __mul__ is called when the scalar is to the left.

Q2. The class SquareMatrixFloat is an implementation of a square matrix as a list of RowVectorFloat elements. It has two attributes, the square matrix itself and the dimension of the same. The methods described in this class are:

1. __init__: Creates a new square matrix. It takes one argument, n, and creates a list of n instances of RowVectorFloat filled with 0's.
2. __str__: Converts the square matrix into a string.
3. sampleSymmetric: Generates random values for every element in the square matrix such that the diagonal elements are uniformly sampled from the range 0 to n, and the rest are uniformly sampled from the range 0 to 1.
4. toRowEchleonForm: Converts the matrix to its row echelon form. To do so, we first divide each row's elements by the diagonal element of that row. This makes all the diagonal elements of the square matrix 1. Then, we take the first row and use row transformations to make the entries in the first column below the diagonal 0. We then repeat the same by choosing the second row, third row, and so on, until all the entries below the diagonal are 0. Finally, we just set the last element of the diagonal to 1.
5. isDRDominant: Checks if the matrix is diagonally dominant (i.e., the absolute value of the diagonal element is greater than the sum of the absolute values of the off-diagonal elements in the corresponding row).
6. jSolve, gSolve: These methods are implementations of the Jacobi and Gauss-Seidel Methods of solving linear equations of the form Ax = b. Both methods take 2 arguments, vector(b), and the maximum number of iterations the algorithm repeats for. Both return a tuple containing a list of errors and the solution vector.

Q3. This is an extension to Q2, where we define a function visualizeConvergence. This function plots the error of the Jacobi and the Gauss-Seidel methods at every iteration up to the maximum number of iterations they are performed for.

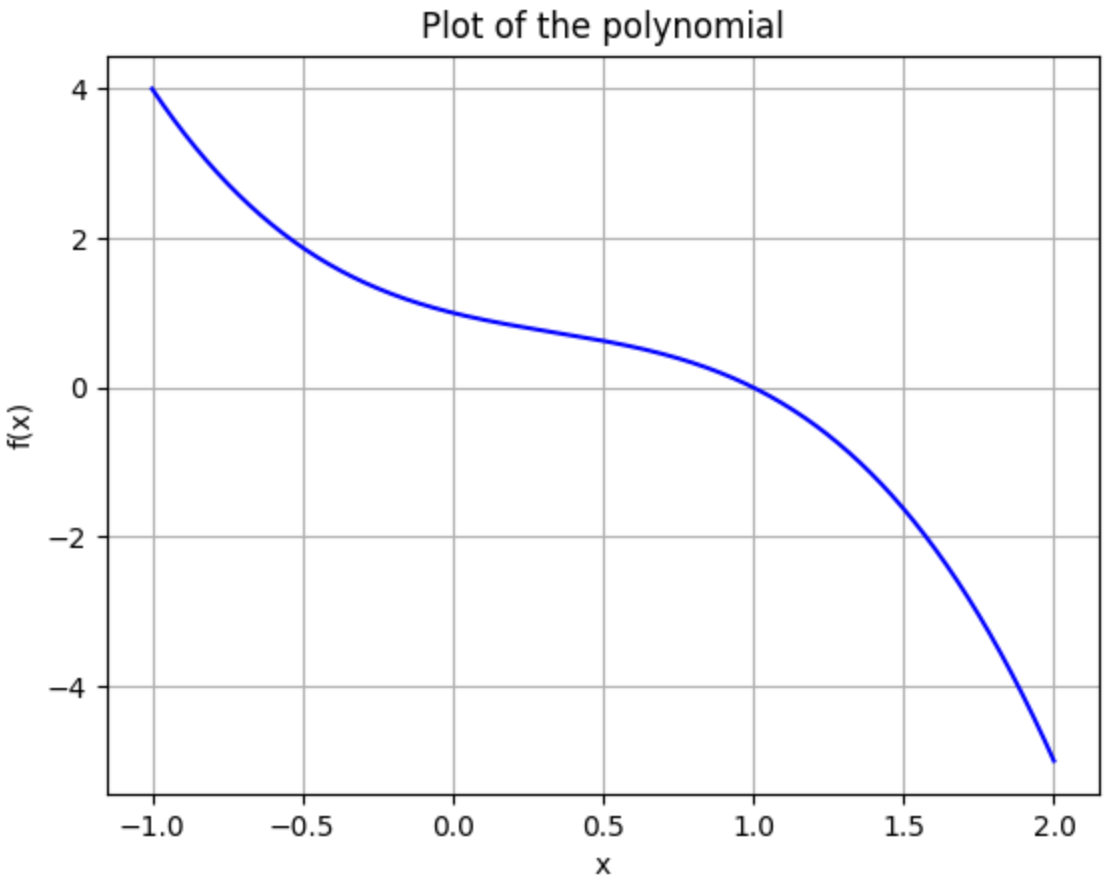


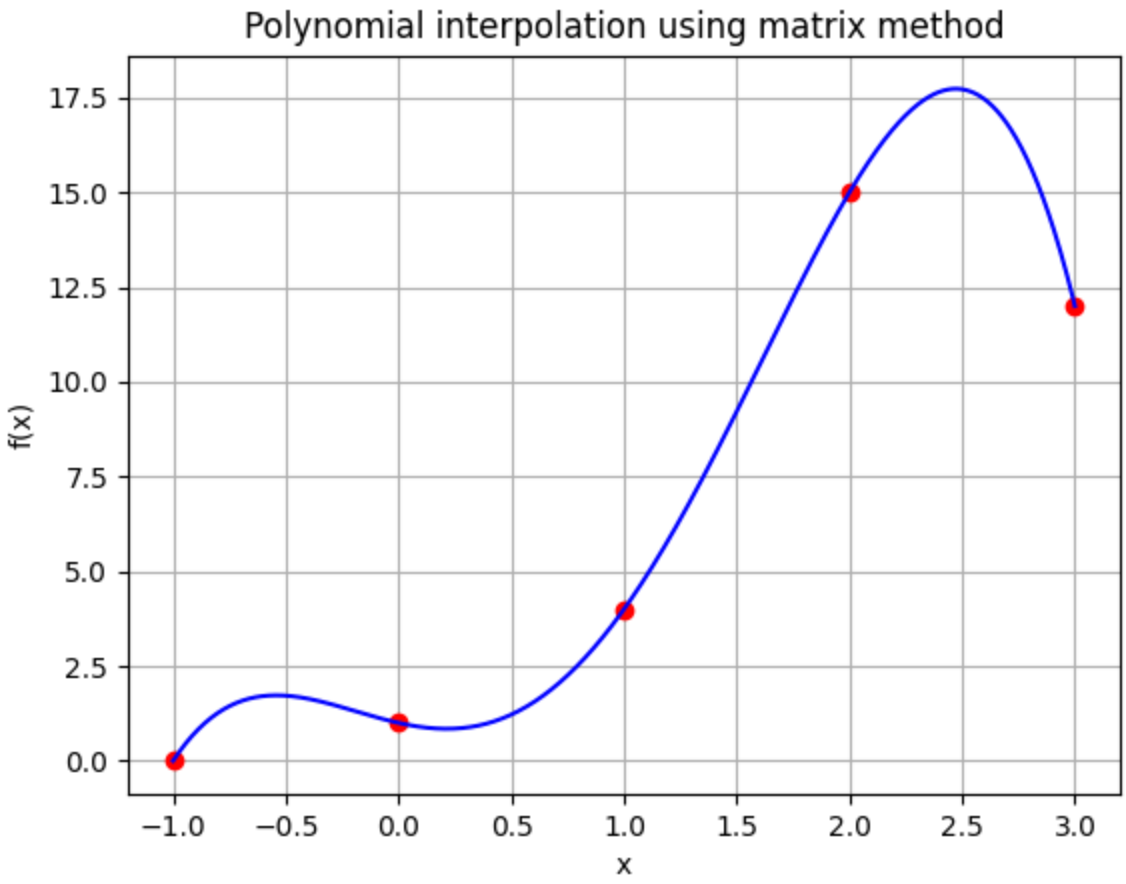
Q4. The class Polynomial represents a polynomial function with a list of coefficients as input. The class provides methods to perform operations such as addition, subtraction, multiplication, indexing, and evaluation of the polynomial function. The following methods are defined under it:

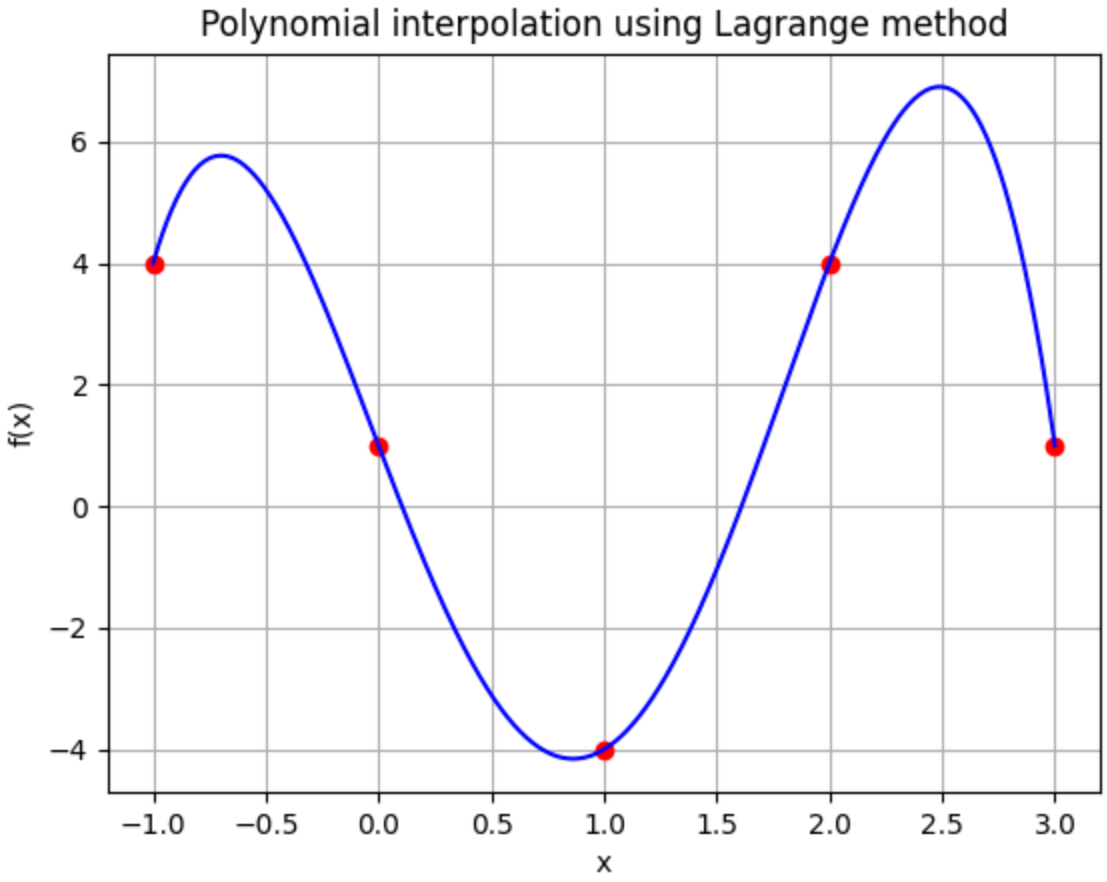
1. __init__: Initializes a list of numbers to represent a polynomial. The first number is the constant, the second the coefficient of the first-degree variable, and so on.
2. __str__: Returns a string representation of the polynomial, which is all the coefficients separated by spaces.
3. __add__, __sub__: Adds/subtracts two polynomials by doing the appropriate operation on the corresponding coefficients of the polynomials and returns a new polynomial instance.
4. __rmul__: Multiplies a scalar to the coefficients of the polynomial and returns a new polynomial.
5. __mul__: Depending on whether the multiplication is between two polynomials or a scalar, it does the appropriate operation. Case 2 is straightforward. For polynomial

multiplication, we create an array of 0's of size n+m-1, where n,m are the sizes of the polynomials. Then, we multiply each coefficient with the other polynomial and add it to the appropriate index of the new array. This array is now returned as a new polynomial.

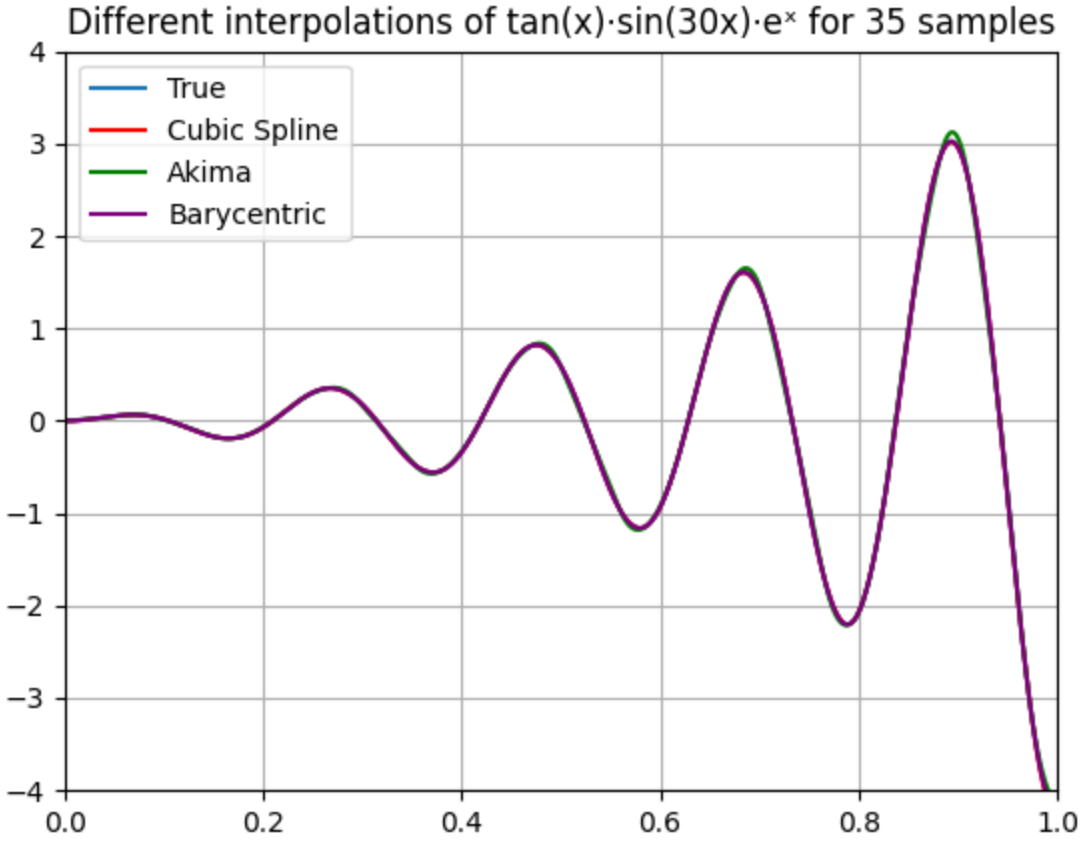
6. __getitem__, evaluate: Evaluates the value of the polynomial (f(x)) when x is given. Evaluate calls __getitem__, whereas __getitem__ can also be called by simply indexing the polynomial with x.
7. show: It plots the polynomial function on a graph given a range of x values. It takes 2 optional arguments, isMatrixMethod(boolean), and points. If the boolean flag is true, then it plots the points given as the second argument as well.
8. fitViaMatrixMethod: It performs polynomial interpolation using the matrix method, given a set of points. It creates a matrix and a vector of values and solves the system of linear equations to obtain the coefficients of the polynomial. It then creates a new Polynomial instance with the coefficients and plots the resulting polynomial function using the show method described above.
9. fitViaLagrangePoly: It performs polynomial interpolation using the Lagrange interpolation method, given a set of points. It creates a new Polynomial instance and adds a term for each point in the set, with each term being a polynomial that has a value of 1 at the point and a value of 0 at all other points. It then multiplies each term by the corresponding y value and adds them together to obtain the final polynomial. It then creates a new Polynomial instance with the coefficients and plots the resulting polynomial function using the show method described above.


Plot of the polynomial
4
2
0
−2
−4
f(x)
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x


Polynomial interpolation using matrix method
f(x)
x
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0



Q5. We try to animate the convergence of various interpolation methods using the FuncAnimation method in matplotlib. The animation will show the interpolation lines for cubic spline, Akima, barycentric interpolations, and the true values for the function we try to interpolate.


Different interpolations of tan(x)·sin(30x)·eˣ for 35 samples
True
Cubic Spline
Akima
Barycentric
4
3
2
1
0
−1
−2
−3
−4
0.0
0.2
0.4
0.6
0.8
1.0