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

112001051
VM Sreeram

Question 1

The data member in the class is made private for making it safe from malicious users. The argument for initialisation of `RowVectorFloat` must always be a list of integers or floats. It is made possible to `print(rvf)` by `__repr__()`. It is made possible to find `len(rvf)` by `__len__()`. It is also made possible to get and set item by usual square bracket method. `+` and `*` operators are overloaded so that objects can be added if they are of same type, and can be multiplied with another `RowVectorFloat` or a number.

Question 2

The data members in the class are made private for making it safe from malicious users. The argument for initialisation of `SquareMatrixFloat` must always be an integer representing the dimension of matrix. The data is internally stored in the format of list of `RowVectorFloat`s. It is made possible to `print(smf)` by `__repr__()`. A method is implemented to set item by usual square bracket method for debugging purposes, which is commented out in the submitted version.

It is possible to randomly sample a symmetric matrix, where diagonal entries are sampled uniformly from 0 to len, and other entries are sampled uniformly from 0 to 1, such that the matrix is symmetric. Sampling is done using `np.random.uniform()` function.

A method `toRowEchelonForm()` is implemented to convert the `SquareMatrixFloat` to reduced row echelon form, using elementary row transformation algorithm. The algorithm, for each row, sets the diagonal entry to 1 by the elementary row operation of dividing by a constant. Then it uses this row to set entries below it to 0 by the elementary row operation of subtracting with a row.

A method `isDRDominant()` is implemented with an optional argument of strictness. The method computes the sum of all entries in a row, and compares it with the double of diagonal entry. If strictness is enforced, the method will return True if and only if it (i) is diagonally row dominant and (ii) for at least one row, the sum of non-drag-ele entries is strictly greater than the drag-ele entry.

A helper private function is implemented to compute error which is helpful for `jSolve` and `gsSolve` methods. The function computes value of the term $||Ax^k - b||_2$.

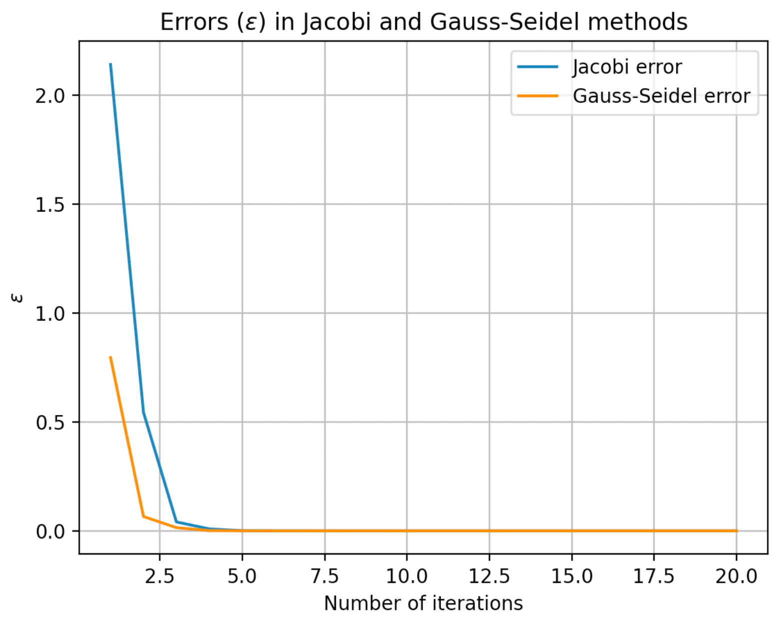
`jSolve` and `gsSolve` methods are implemented as shown in class. `jSolve` expects that the matrix is strictly diagonally dominant by row, and vector *b* should be of same size as that of the matrix. `gsSolve` expects that the vector *b* should be of same size as that of the matrix.

Question 3

The function `visualisejSgsS()` uses the previously implemented `jSolve` and `gsSolve` methods. The function takes two arguments — n: dimension of SquareMatrixFloat and m: number of iterations to visualise. We symmetric sample until we get a matrix which is strictly diagonally dominant by row. Then, *b* was constructed with length same as that of the matrix, and values randomly sampled from 1 to length using `random.randrange()`.

It was observed that Jacobi method's converges to 0 slower than the Gauss-Seidel method.

A sample output graph is displayed below for convenience.



Question 4

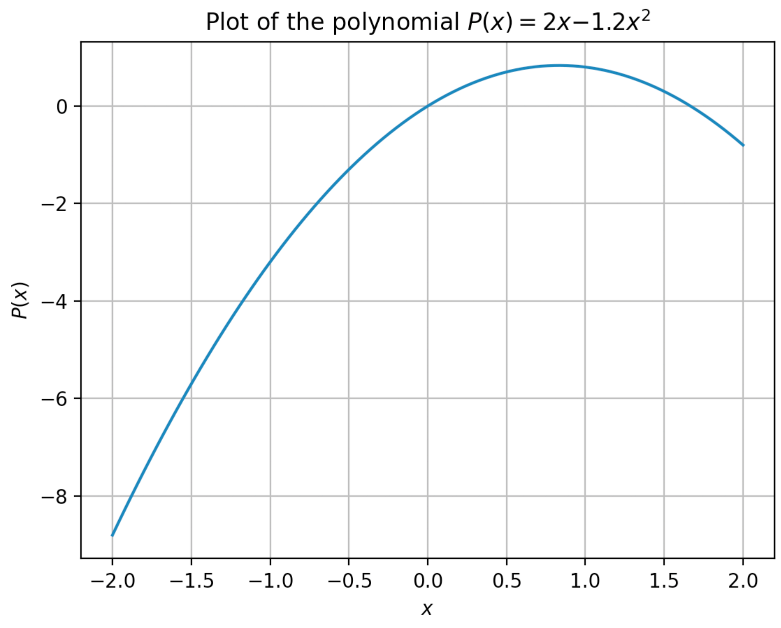
The `Polynomial` class is created to represent abstract polynomial. It should be initialised with a list of numbers.

The data members are declared privately. It is possible to print the coefficients of an object of the class using the usual `print(obj)`, which was implemented using `__str__()`. +, −, `*` operators were overloaded, handling necessary cases. It is possible to get the value of the polynomial at some `x`, by usual `obj[x]`. This was implemented by `__getitem__()`.

While multiplying with a floating point number, it is possible to have precision inconsistencies inbuilt in Python. This was handled by rounding each coefficient to as many decimal places as the multiplied floating point number is.

A private helper function `__title()` was implemented which returns the stringified LaTeX representation of the polynomial. The function handles corner cases like not printing leading + for equations like $4x^2$, printing x rather than x^1, and so on.

The method `show` plots and displays the graph. A sample output is shown below.

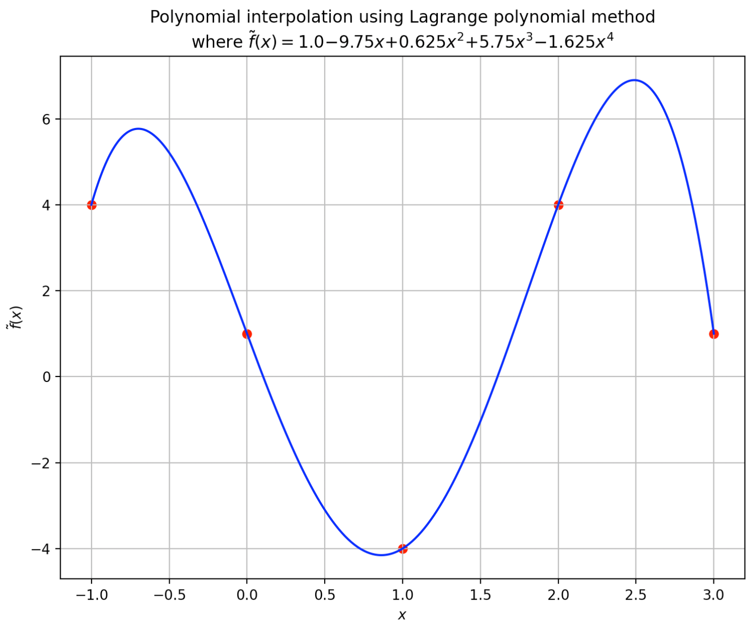


Another private helper function `__round_vals()` was implemented which rounds all coefficients upto 4 decimal places. This was implemented so that it doesn't look too long while we print it.

The functions `fitViaMatrixMethod()` and `fitViaLagrangePoly()` were implemented.

The function `fitViaMatrixMethod()` was implemented with the help of `np.linalg.solve()` module. The A and b for this are constructed by looping through each point and computing the necessary values. The given points were also plotted along with the fitted polynomial.

The function `fitViaLagrangePoly()` was implemented following the algorithm mentioned in the class. The given points were also plotted along with the fitted polynomial. A sample output is shown below.



Question 5

This problem was solved using the functions `CubicSpline`, `Akima1DInterpolator`, `BarycentricInterpolator` from the module `scipy.interpolate`. `FuncAnimation` from the module `matplotlib.animation` was used to create the animation.

The `update()` function uniformly samples the required number of points in the range [0,1] and computes actual value of the function, and sends it to the imported interpolation functions to get the fitted polynomials. This is plotted by sampling in small interval (500 samples). The true function is also similarly plotted.

The output is saved as `interpolation_animation.gif` in the same folder for convenience.