Lab Report
Assignment-3

Q1. In this question I have created a class called *RowVectorFloat*
Having 2 functions namely *value()* and *index()* that are for the validation purpose.

- *value()* - checks whether the argument is list of int or float data type or not. If not, then raise an Exception.
- *index()* - checks whether the given index is not int data type or not and that the index in well within the bound, if any of these conditions are not met then an exception is raised.

Next we have the constructor *__init__* function that creates a copy of the list and assign it to *vector* variable of the object. Then we have *__str__* function that will help us in printing the instance of the class.

After that we have a couple of inbuilt functions that will help us in computation, such as

- *__len__* : to compute length of vector.
- *__getiten__* : to retrieve the element at a given index.
- *__setitem__* : to change the element at a given index.
- *__add__ & __mul__* : functions to perform addition and multiplication operations on vectors.
- *__radd__ & __rmul__* : functions that will solve the problem of right addition and multiplications respectively.

After the code there are 4 sample test cases.
One such test case with the output is:

INPUT:
```
r = RowVectorFloat([7, 1, 0, 5])
print(r)
print(len(r))
print(r[3])
r[2] = 3
print(r)
```

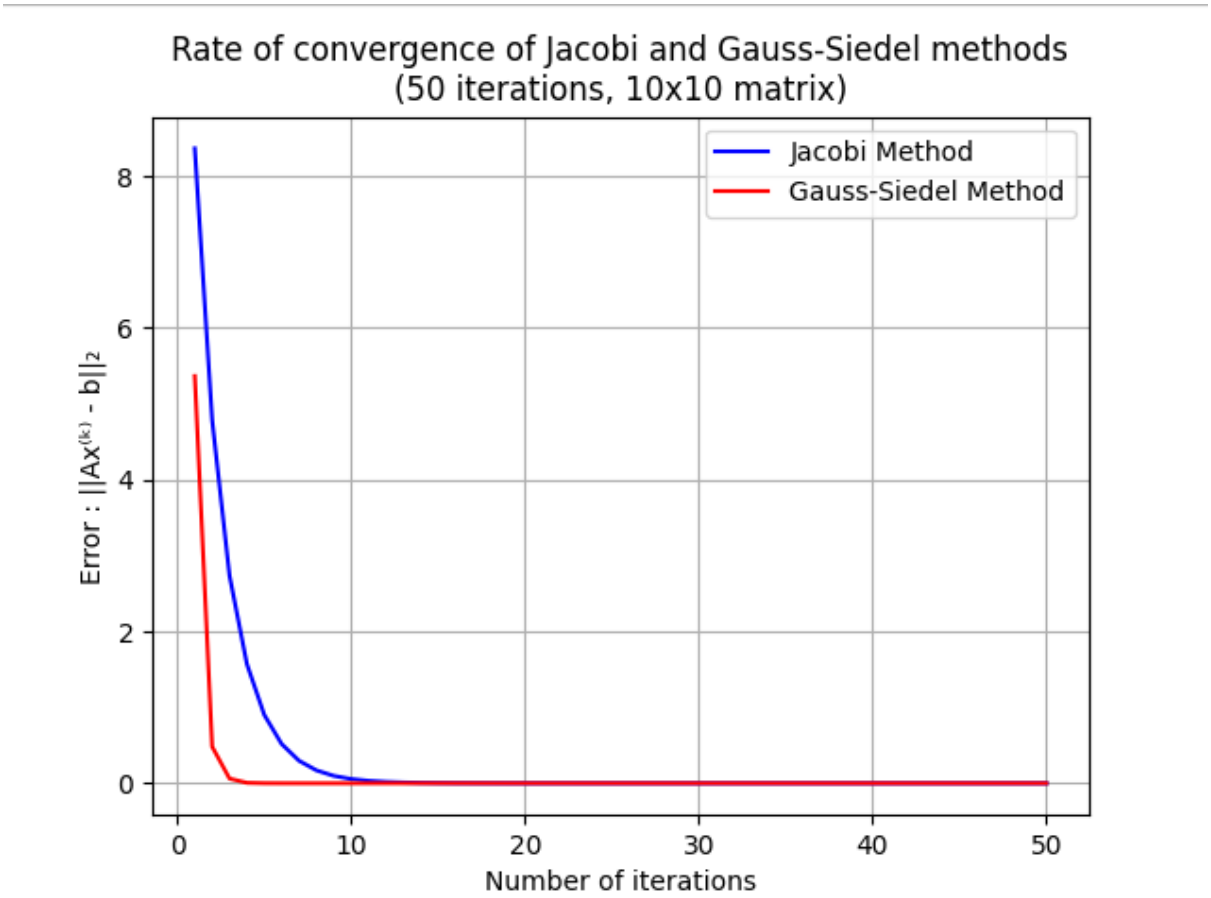
OUTPUT:
```
7 1 0 5
4
5
7 1 3 5
```

Q2. This question is an extension of the previous question with an extra class called *SquareMatrixFloat* that represents a square matrix of list of *RowVectorFloat* objects. The inbuilt function such as *__init__ & __str__* are used to initialize the Square Matrix and to print the matrix respectively.
Other functions with their significance are:

- *SampleSymmetric()* : function to sample the matrix according to the criteria mentioned in the question. The *random.uniform* function is used to sample the random numbers uniformly between 0 & 1.
- *toRowEchelonForm()* : function that will convert the matrix into its row echelon form through elementary matrix row operations.
  Mehtod:
  - Pivot the matrix
    - Find the pivot, i.e., the 1st non zero element in the 1st column.
    - Interchange rows.
    - Multiply the elements of pivot row by its inverse so as to make the pivot=1.
    - Add multiples of pivot row to each row so as to make every element under the pivot column=0.
  - Repeat the pivot
    - Repeat the above procedure while ignoring the pivot of previous rows.
    - Continue until no pivot is left to be processed.
- *dr()* : function to check whether the matrix is diagonally row dominant or not.
- *value()* : functions to checl the validity of the value, i.e., the value is a list of int or float data type only.
- *itrn()* : function that will perform the iterations for the Jacobi or Gauss-Siedel method with a flag argument that implies wether we have to do Jacobi iteration or the Gauss-Siedel iteration.
- *jSolve()* : function calls *itrn() with flag=true*, i.e., Jacobi iteration.
- *gsSolve()* : function calls *itrn()* with *flag=false,* i.e., Gauss-Seidel iteration.

Q3. This question is just the visualisation of the previous question using the *matplotlib* library of the python library. Here Im using the following function to implement the same.

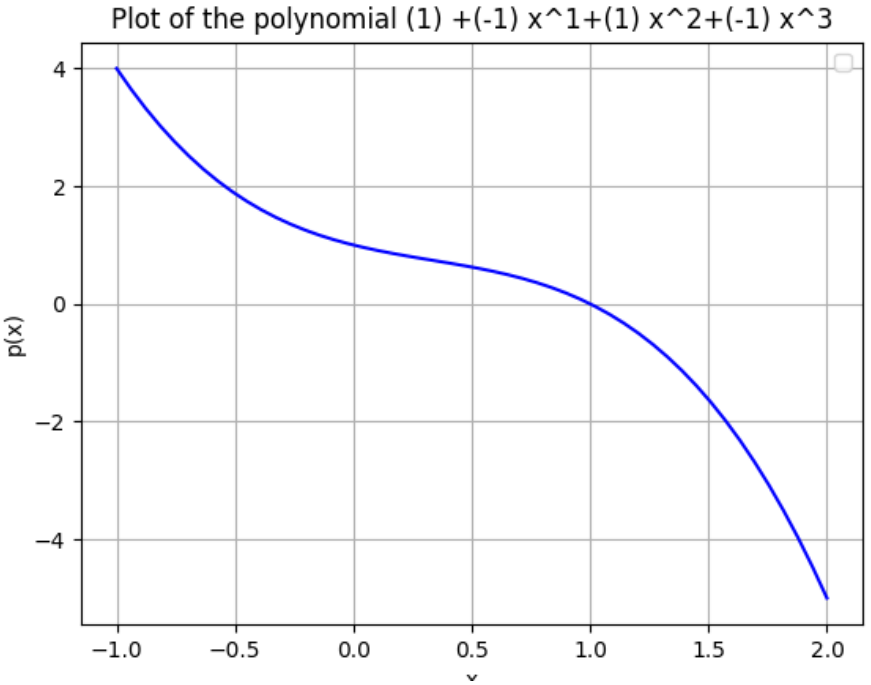
graph() : function to visualize the rate of convergence of Jacobi and Gauss-Siedel method of linear system with diagonally dominant row with square symmetric matrix. After generating the same I get the error values through *jSolve() & gsSolve()* function. After that Im plotting the graph between no. of iterations and the error in each iteration through the *matplotlib* library. One such obtained graph is the following:

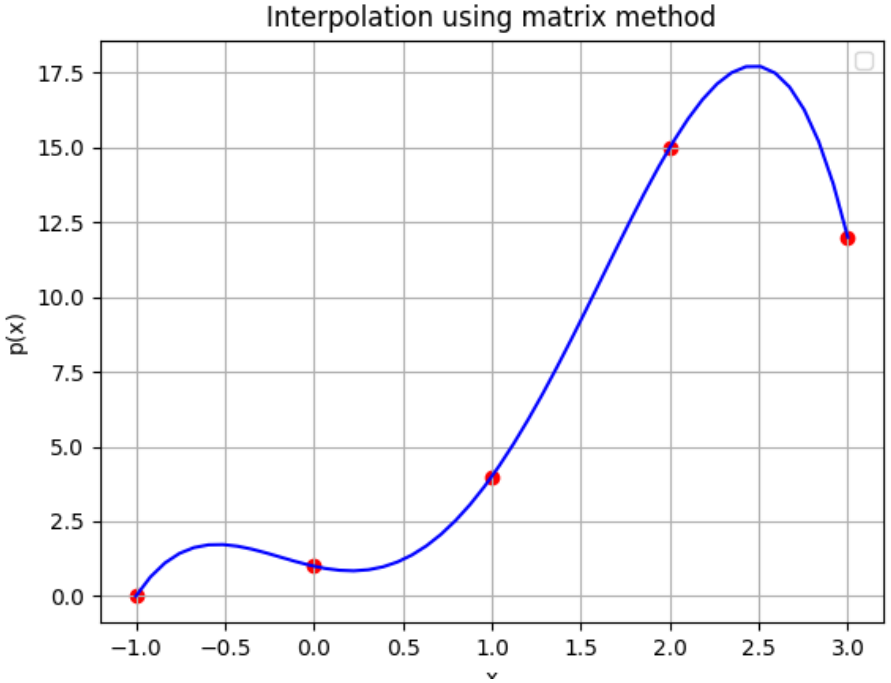


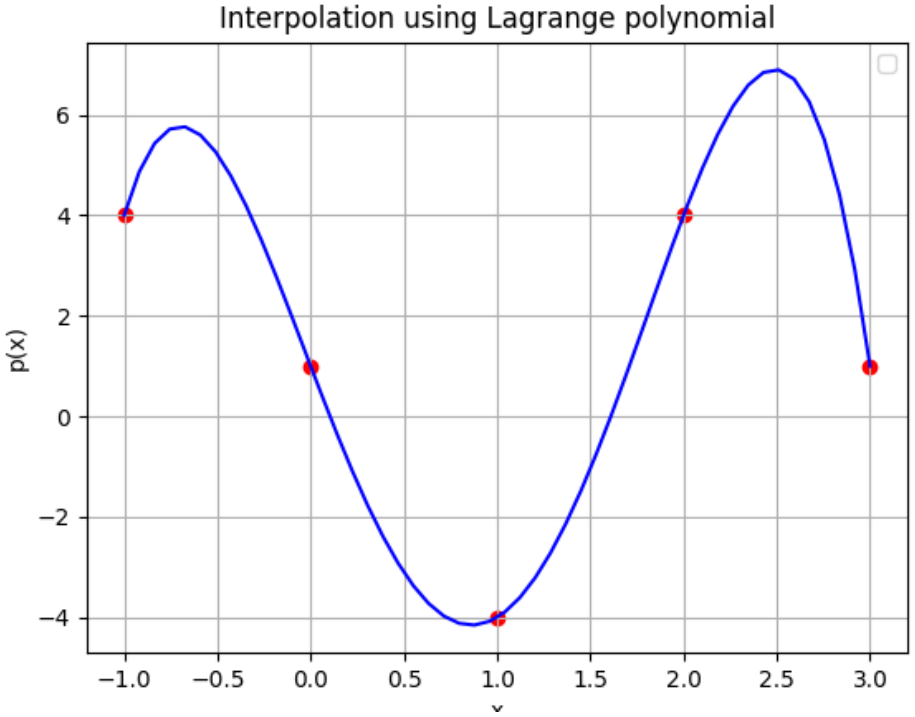
Q4. In this question we are supposed to make a class *Polynomal* that will create the polynomial equation and perform different operations on that equation. Following functions are created to implement the same. *__init__* is a constructor used to initialize the object, *__str__* function is used to print the object. Other inbuilt function such as *__add__*, *__sub__*, *__mul__*, *__radd__*, *__rmul__*, *__getitem__* is used to perform different arithmetic operations on the coefficients of the polynomials. Other functions are:

- *temp()* : function to create a temporary list with coefficient values as 0.
- *calc()* : function to do the different calculations such as addition, subtraction and multiplication on the polynomial.
- *plot()* : function to plot the graph.
- *fitViaMatrixMethod()* : function that fits, using the idea of linear systems, a polynomial to the points passed as its argument.
- *fixViaLagrangePoly()* : function that computes the Lagrange polynomial for the points passed as argument.

The Graph obtained for the given test cases are:




Interpolation using matrix method
p(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


Interpolation using Lagrange polynomial
p(x)
x
6
4
2
0
−2
−4
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0

Q5. In this question we are required to create an animation for interpolation. For that we are using the *FuncAnimation* method of the *matplotlib* library. *CubicSpline, Akima1DInterpolator, BarycentricInterpolator* are the 3 methods used for the animation. Following functions are used to generate the animation.

- ***__init__*** : function to initialise all the necessary details.
- *graph()* : function to update the curves or plots.
- *eval()* : function to give the test case for the graph interpolation.
- *animate()* : function to plot the graphs required.