112001041
Sidharth chadha

Coding assignment 3

q1)

This question asked us to make instances of row vectors and use some predefined methods to do basic calculations on them.

```
r = RowVectorFloat([1,2,3])
print(len(r))
print(r)
print(r[2])
print(r[-1])
r=r*2
print(r)
r=r+r
print(r)
r2=RowVectorFloat((2,4,5))
r=-1*r+3*r2
print(r)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL COMMENTS

```
(base) sid@sid-HP-Spectre-x360-Convertible-13-aw0xxx:~/Documen
3
1 2 3
3
3
2 4 6
4 8 12
2 4 3
```

q2)
This follows the first ques to make use of row vectors to make a square matrix and perform some operations on it also using some predefined methods in python.

```python
r = SquareMatrixFloat(3)
r.sampleSymmetric()
print(r)
r.toRowEchelonForm()
print(r.isDRDominant())
print(r)

# s = SquareMatrixFloat(4)
# s.sampleSymmetric()
# (e, x) = s.jSolve([1, 2, 3, 4], 10)
# print(x)
# print(e)



s = SquareMatrixFloat(4)
s.sampleSymmetric()
(err, x) = s.gsSolve([1, 2, 3, 4], 10)
print(x)
print(err)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL COMMENTS

```
0.30 0.55 0.25
0.37 0.25 2.76

False
1.00 1.13 1.39
0.00 1.00 -0.77
0.00 0.00 1.00
```

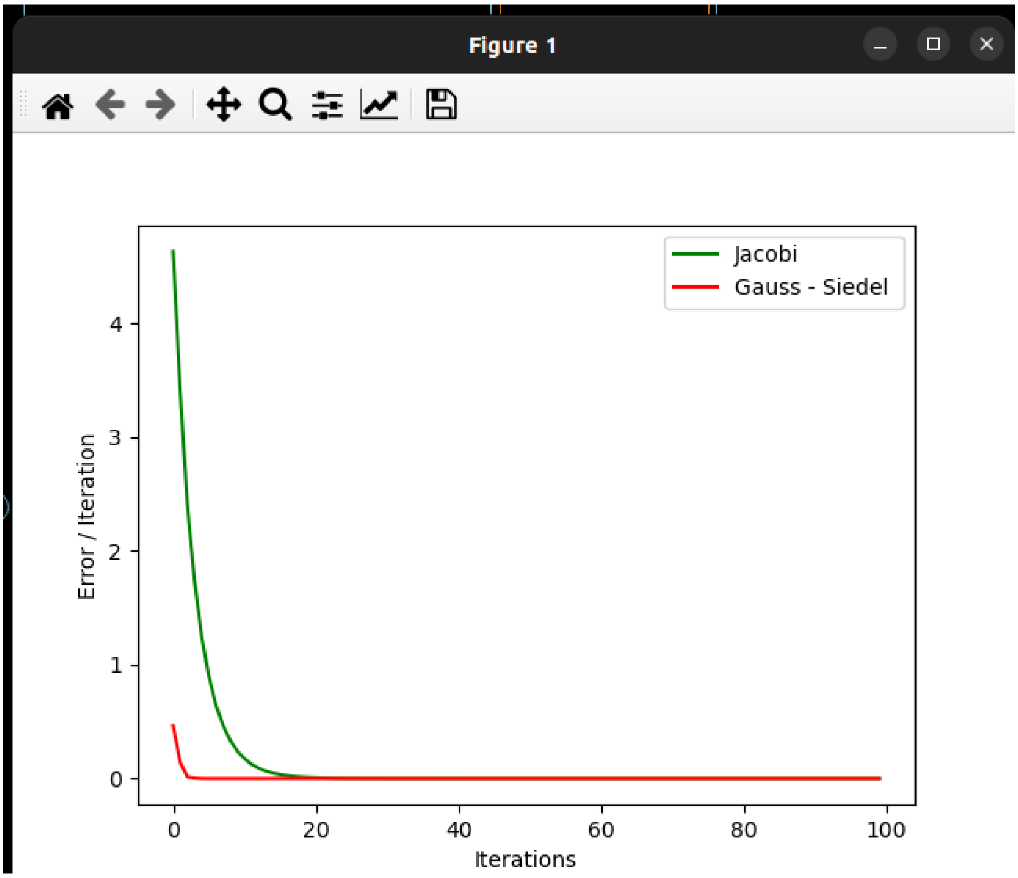
q3)

For this we used a n*n matrix, and kept sampling until its row dominant and then used the jacobi and gauss seidel method until convergence.

We then run the errors received against the number of iterations.

note-> we get a different graph(error/iteration) because of sampling

the error decreases with the iterations. But the rate of decrease of error is more in case of Gauss - Seidel Convergence.



q4)
In this question we used the matrix method and lagrange methods
In matrix method we substitute given x in a polynomial of degree 1 less than the total given points. Then we solve the matrix using the np.linalg.solve method. While in lagrange method we used the lagrange formula.

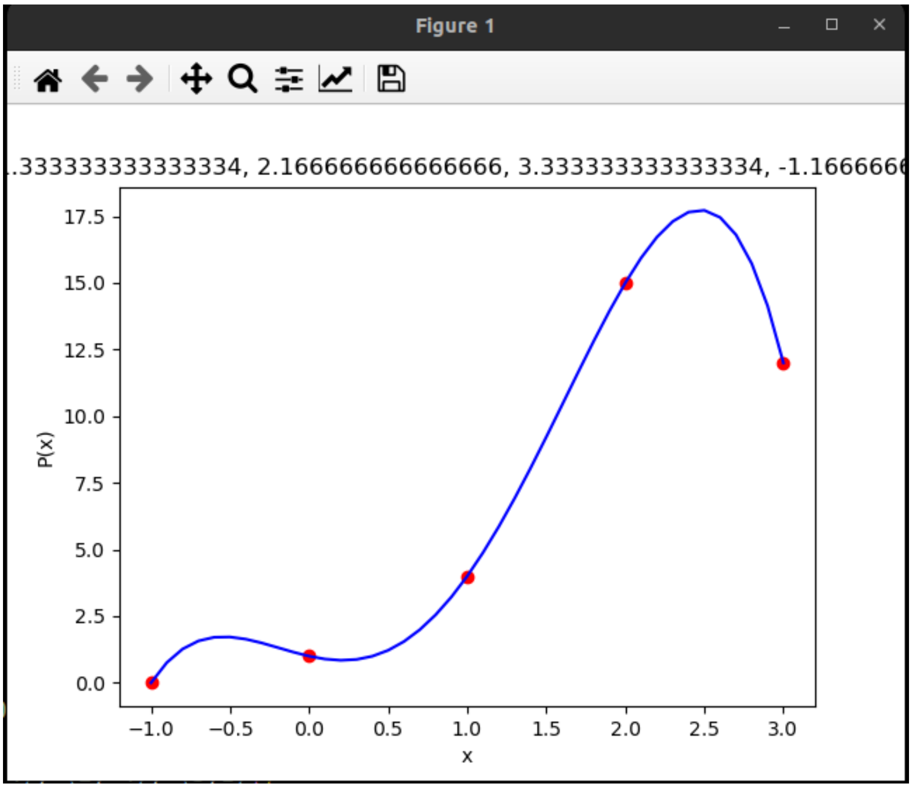


Figure 1
.333333333333334, 2.166666666666666, 3.333333333333334, -1.166666
P(x)
x

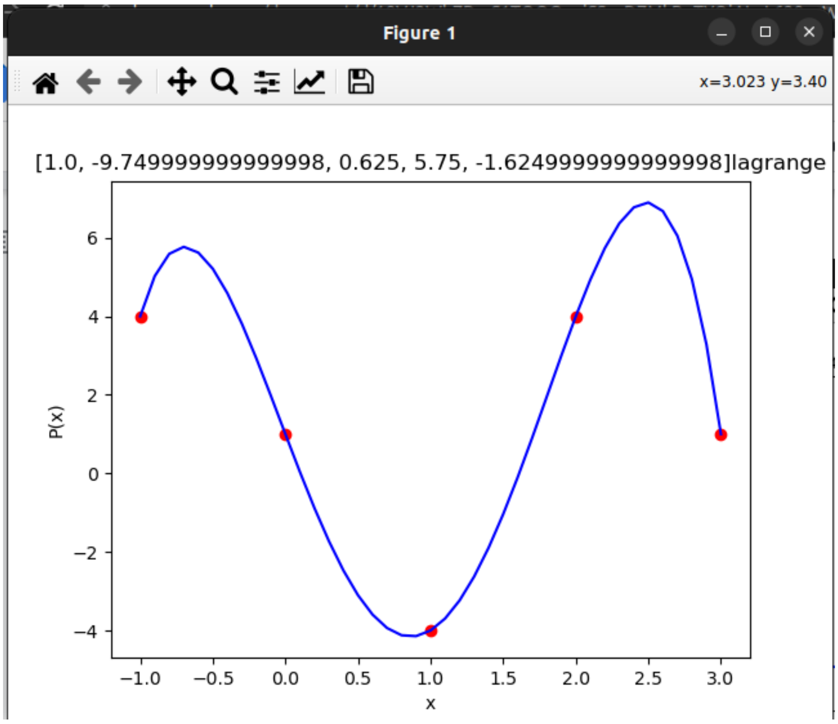


Figure 1
x=3.023 y=3.40
[1.0, -9.749999999999998, 0.625, 5.75, -1.6249999999999998]lagrange
P(x)
x

q5)

We have created a function that can run a loop over three different functions to generate datasets. We will also create two datasets: one that contains true values to help us calculate absolute values, and another that contains 50 random points within the range of 0 to 1. During each iteration of the loop, we will select a new value from the true values dataset to use as a reference for interpolating our graphs. We will perform three types of interpolation: Barycentric interpolation, Akima Interpolation, and Cubic Spline.

We applied this on 2 functions and created animation for it:

```
tan(x).sin(50x).e^x
```

```
3x^3 - 7x^2 - 2x + 5.5
```