Project in Numeral Analysis

Ioannis Chalvatzis 10731
Nikos Karapatsias 69420

1 Introduction

The presentation of the Project will consist of two parts. The first will be on the Approximation Problem and the second on LU and QR Factorization - Hilbert matrix.

The code will be presented at first piece by piece providing explanation on what it does. At the end the whole code will be available alongside all the documentation with the sources that we used.

2 Approximation Problem

2.1 Creation of the 't' vector

The t vector is a 50-spot vector with t being part of [0,1] with a step of 0,02.

The algorithm that we used is:

```

t = [0]
for i in range(1, 50):
    t.append(i / 50)
t = np.array(t)

```

2.2 Creation of the 'y' function

We created the y function following the instructions of the Project.

```

y = np.cos(4 * t) + 0.1 * np.random.randn(t.shape[0])

```

2.3 Creation of the Vandermonde matrix

Before we start finding the approximations we need to create the Vandermonde matrix that will help us with all three methods.

To create it we used the function **vander** from the numpy library.

```

Vandermonde_matrix = np.vander(t, deg +1, increasing=True)

```

The parameter *deg* describes what degree of polynomial we want. In this case it has the value 4 assigned. Running the algorithm we had this result:

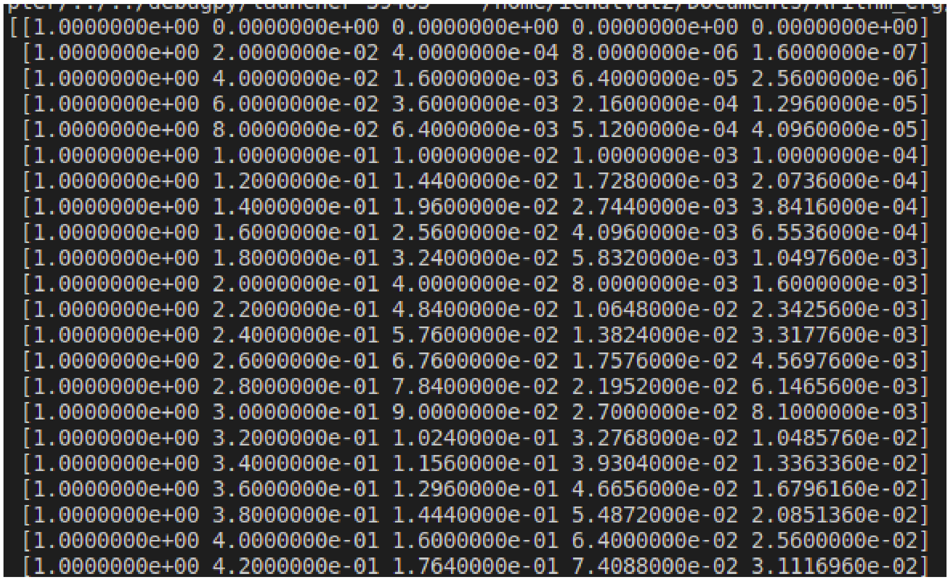

Figure 1: The result is what we were expecting. A 50x5 matrix with the correct values.

2.4 Least Square Method

To find an approximation with the Least Square Method we used the function **lstsq** from the numpy.linalg library. Then we took the coefficient and build the polynomial.

```
coefficient, residuals, rank, singular_values = lstsq(Vandermonde_matrix, y)
poly_1 = Polynomial(coefficient)
```

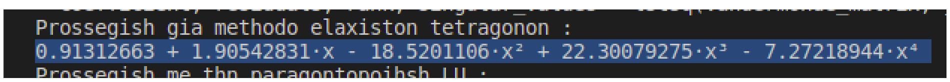

Figure 2: The polynomial approximation using the Least Square method.

2.5 LU - QR Factorization

2.5.1 LU

To implement this Factorization we used the functions **lu-factor** and **lu-solve** from the scipy.linalg library. Then we used a similar function to create the polynomial.

```
A = Vandermonde_matrix .T @ Vandermonde_matrix
b = Vandermonde_matrix .T @ y
LU, piv = lu_factor(A)
coefficient2 = lu_solve((LU, piv), b)
poly_2 = np.poly1d(coefficient2[::-1])
```

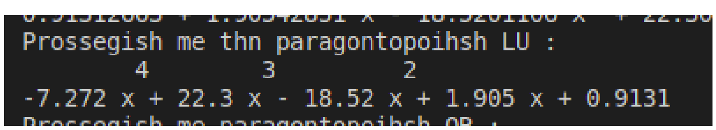

Figure 3: The polynomial approximation using the LU method.

2.5.2 QR

To implement this Factorization we also used the function **qr** from the scipy.linalg library. Then we used the same function to create the polynomial.

```
Q, R = qr(Vandermonde_matrix)
QTY = Q.T @ y
coeffisient3 = np.linalg.solve(R[:deg + 1, :], QTY[:deg + 1])
poly_3 = np.poly1d(coeffisient3[::-1])
```

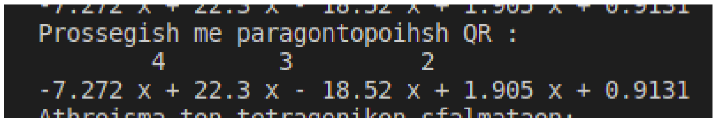

Figure 4: The polynomial approximation using the QR method.

We can see that the results are identical between the QR-LU methods but they have a very small difference with the Least Square method.

2.6 Sum of Square Errors (SSE)

We calculated the errors by subtracting the true values we get from 'y' from the approximation evaluated at 't'. Then we calculate the Sum of Square Errors.

```
errors_LS = y - poly_1(t)
square_errors_LS = np.sum(errors_LS**2)
```

```
errors_LU = y - poly_2(t)
square_errors_LU = np.sum(errors_LU**2)
```

```
errors_QR = y - poly_3(t)
square_errors_QR = np.sum(errors_QR**2)
```

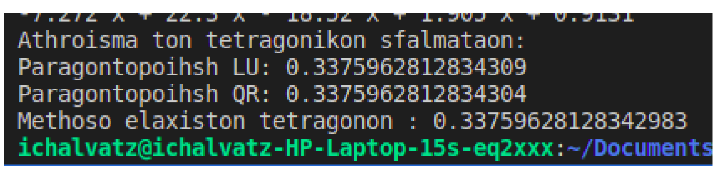

Figure 5: The SSE results.

We can see that the results are similar and only have differences in the last digits.

2.7 Diagram

In the diagram we presented the approximation curves we got from every method and also the data points from 't-y'. The algorithm for the diagram is:

```python
plt.scatter(t, y,10,'black', label='Data Points')
plt.plot(t, poly_2(t),'b--', label='LU Factorization',linewidth = 3)
plt.plot(t, poly_3(t),'r-', label='QR Factorization',linewidth = 1.5)
plt.plot(t, poly_1(t),'g:', label='Least Squares', linewidth = 3.5)
plt.title('Approximations')
plt.xlabel('t')
plt.ylabel('y')
plt.legend()
text = f'SSE (LU Factorization): {square_errors_LU:.4f}\nSSE (QR Factorization): {square_errors_QR:.4f}\nSSE (Least Squares)
plt.text(0.7, 1.1, text, transform=plt.gca().transAxes)
plt.show()
```

The result is:

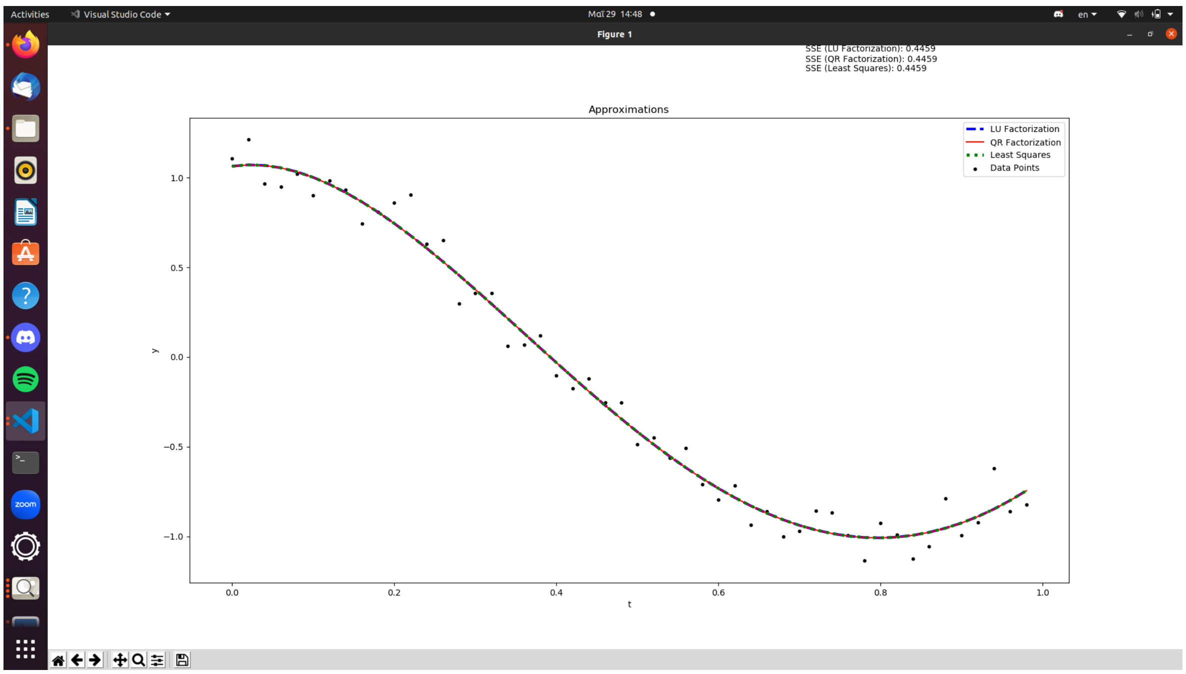

Figure 6: The Diagram.

2.8 Complete code

```python
import numpy as np
from numpy.polynomial import Polynomial
from numpy.linalg import lstsq
from scipy.linalg import lu_factor, lu_solve
from scipy.linalg import qr
import matplotlib.pyplot as plt


#DImiourgia toy t dianismatos
t = [0]
for i in range(1, 50):
    t.append(i / 50)
t = np.array(t)

#dimiourgia y
```

```
y = np.cos(4 * t) + 0.1 * np.random.randn(t.shape[0])

#dimourgia tou Vandermonde matrix gia tin methodo twn elaxiston tetragwnwn thn paragontopoihsh LU kai QR
deg = 4
Vandermonde_matrix = np.vander(t, deg +1, increasing=True)
print(Vandermonde_matrix)

#METHODOS TWN ELAXISTVN TETRAGONWN
coefficient, residuals, rank, singular_values = lstsq(Vandermonde_matrix, y)


poly_1 = Polynomial(coefficient)

errors_LS = y - poly_1(t)
square_errors_LS = np.sum(errors_LS**2)
print("Prossegish gia methodo elaxiston tetragonon : ")
print(poly_1)


#PARAGONTOPOIHSH LU
A = Vandermonde_matrix .T @ Vandermonde_matrix
b = Vandermonde_matrix .T @ y

LU, piv = lu_factor(A)

coefficient2 = lu_solve((LU, piv), b)

poly_2 = np.poly1d(coefficient2[::-1])

errors_LU = y - poly_2(t)
square_errors_LU = np.sum(errors_LU**2)
print("Prossegish me thn paragontopoihsh LU :")
print(poly_2)

#PARAGONTOPOIHSH QR
Q, R = qr(Vandermonde_matrix)

QTY = Q.T @ y

coeffisient3 = np.linalg.solve(R[:deg + 1, :], QTY[:deg + 1])

poly_3 = np.poly1d(coeffisient3[::-1])

errors_QR = y - poly_3(t)
square_errors_QR = np.sum(errors_QR**2)
print("Prossegish me paragontopoihsh QR :")
print(poly_3)


print("Athroisma ton tetragonikon sfalmataon:")
print("Paragontopoihsh LU:", square_errors_LU)
print("Paragontopoihsh QR:", square_errors_QR)
print("Methoso elaxiston tetragonon :", square_errors_LS)

# Kataskeyi tou diagrammatos
plt.scatter(t, y,10,'black', label='Data Points')
plt.plot(t, poly_2(t),'b--', label='LU Factorization',linewidth = 3)
plt.plot(t, poly_3(t),'r-', label='QR Factorization',linewidth = 1.5)
plt.plot(t, poly_1(t),'g:', label='Least Squares', linewidth = 3.5)
```

```python
plt.title('Approximations')
plt.xlabel('t')
plt.ylabel('y')
plt.legend()



# prosthiki twn SSE timwn sto diagrama
text = f'SSE (LU Factorization): {square_errors_LU:.4f}\nSSE (QR Factorization): {square_errors_QR:.4f}\nSSE (Least Squares)
plt.text(0.7, 1.1, text, transform=plt.gca().transAxes)

# emfanisi diagramatos
plt.show()
```

2.9 References and Sources

- Introduction in Numeral Analysis (Book) -Author Leonidas Pitsoulis
- Guide on Overleaf LATEX
- Matplotlib
- Numpy
- Least Square Regression in Python