Numerical Methods - MAT202E

Homework II

Emre Anıl OĞUZ - 110190170

I. Programming language and used libraries

As a programming language Python is chosen due to my experience with it, more readable code for the review, and easy to code graphical figures.

```
import numpy as np
import matplotlib.pyplot as plt
from tabulate import tabulate
```

Figure 1. Used libraries

As seen in Figure 1 different libraries were used for this assignment. NumPy was chosen to make it much easier to work with matrices . For the graphical illustration, the matplotlib library is used. Finally, the tabulate library is used for showing outputs with a proper format. Any of the libraries not used for the computation as described in the homework document.

II. Question 1-A)

Decompose the coefficient matrix into LU form using Gauss elimination.

```
def decompose_matrix(matrix):

    n = len(matrix)
    U = matrix.copy()
    L = np.eye(n, dtype=np.double) #Identity Matrix
    P = np.eye(n, dtype=np.double)
    for i in range(n):
        for k in range(i, n):
            if ~np.isclose(U[i, i], 0.0):
                break
            U[[k, k+1]] = U[[k+1, k]]
            P[[k, k+1]] = P[[k+1, k]]

        factor = U[i+1:, i] / U[i, i]
        L[i+1:, i] = factor
        U[i+1:] -= factor[:, np.newaxis] * U[i]


    return P, L, U
```

Figure 2. Main loop of Question 1-A.

```
L
[1.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 1.0, 0.0, 0.0, 0.0, 0.0]
[-0.49999994306897183, -0.0045000004303985845, 1.0, 0.0, 0.0, 0.0]
[41.666669513218075, 0.24999996925724396, 55.55555027010725, 1.0, 0.0, 0.0]
[0.0, 0.0, -0.8888887462942491, -0.0026666683701554415, 1.0, 0.0]
[0.0, 0.0, 74.07407568594903, -5.4878699557382035e-08, -191.21532992856348, 1.0]
U
[87.82557, 0.0, -43.91278, 3659.399, 0.0, 0.0]
[0.0, 813199.7, -3659.399, 203299.9, 0.0, 0.0]
[0.0, -4.547473508864641e-13, 49.40188542499956, 2744.5489291666418, -43.91278, 3659.399]
[0.0, 2.5263739312371057e-11, 0.0, 457424.83649691206, -1219.8003432098399, -0.02510288020130247]
[0.0, -3.368497880739476e-13, 0.0, 0.0, 2.1264010463623957, -406.6004776406277]
[0.0, -3.072585228893665e-11, 0.0, 0.0, 0.0, 57784.95702774834]
```

Figure 3. Outputs of Question 1-A

III. Question 1-B)

Decompose the coefficient matrix into LU form using Cholesky decomposition.

```python
def transpose(L):
    """Transposes a lower triangular matrix L."""
    n = len(L)
    LT = [[0.0] * n for i in range(n)]
    for i in range(n):
        for j in range(n):
            LT[j][i] = L[i][j]
    return LT


def cholesky(A):
    '''
    Cholesky decomposition of a positive definite matrix A.
    Returns the lower triangular matrix L such that A = L*LT.
    '''



    n = len(A)

    L = [[0.0] * n for i in range(n)]

    for i in range(n):
        for k in range(i+1):
            tmp_sum = sum(L[i][j] * L[k][j] for j in range(k))

            if (i == k):
                L[i][k] = sqrt(A[i][i] - tmp_sum)
            else:
                L[i][k] = (1.0 / L[k][k] * (A[i][k] - tmp_sum))
    U = transpose(L)
    return L,U
```

Figure 4. Main loop of Question 1-B

```
L
[9.371529757728991, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 901.7758590692035, 0.0, 0.0, 0.0, 0.0]
[-4.685764345333671, -4.057991753934469, 7.02864748191283, 0.0, 0.0, 0.0]
[390.4804332485825, 225.4439370442256, 390.4803785122709, 676.3318981808504, 0.0, 0.0]
[0.0, 0.0, -6.247685648341726, -1.8035528806060646, 1.4582184494657844, 0.0]
[0.0, 0.0, 520.640565545066, -3.71162149983089e-05, -278.8337219225181, 240.38501830968644]
U
[9.371529757728991, 0.0, -4.685764345333671, 390.4804332485825, 0.0, 0.0]
[0.0, 901.7758590692035, -4.057991753934469, 225.4439370442256, 0.0, 0.0]
[0.0, 0.0, 7.02864748191283, 390.4803785122709, -6.247685648341726, 520.640565545066]
[0.0, 0.0, 0.0, 676.3318981808504, -1.8035528806060646, -3.71162149983089e-05]
[0.0, 0.0, 0.0, 0.0, 1.4582184494657844, -278.8337219225181]
[0.0, 0.0, 0.0, 0.0, 0.0, 240.38501830968644]
```

Figure 5. Outputs of Question 1-B

IV. Question 1-C)

Calculate the inverse of the coefficient matrix (use L and U matrices obtained in either part a or part b.

```python
def forward_substitution(L, b):

    n = L.shape[0]
    y = np.zeros_like(b, dtype=np.double);

    y[0] = b[0] / L[0, 0]

    for i in range(1, n):
        y[i] = (b[i] - np.dot(L[i,:i], y[:i])) / L[i,i]
    return y


def back_substitution(U, y):
    n = U.shape[0]
    x = np.zeros_like(y, dtype=np.double);
    x[-1] = y[-1] / U[-1, -1]

    for i in range(n-2, -1, -1):
        x[i] = (y[i] - np.dot(U[i,i:], x[i:])) / U[i,i]
    return x

def plu_inverse(A):
    n = len(A)
    b = np.eye(n)
    Ainv = np.zeros((n, n))
    P, L, U = decompose_matrix(A)
    for i in range(n):
        y = forward_substitution(L, np.dot(P, b[i, :]))
        Ainv[:, i] = back_substitution(U, y)
    return Ainv
```

Figure 6. Main loop of Question 1-C

```
A_Inv
[0.06131952042453837, 0.00048489323609521436, 0.12352882560973229, 0.00028394616417800196, 0.16341072936374015, 0.00021696413016617557]
[0.00048489323609521436, 6.069907692795018e-06, 0.0012872245721290705, 3.8092551802334848e-06, 0.0018383728395821918, 3.055707901365747e-06]
[0.12352882560973234, 0.0012872245721290711, 0.3640341802087725, 0.0014037177545774156, 0.571938587471895, 0.0011692811971389296]
[0.00028394616417800024, 3.8092551802334907e-06, 0.0014037177545774162, 1.0029902511479465e-05, 0.0029414045625191084, 8.82423304119061e-06]
[0.16341072936374024, 0.0018383728395821913, 0.5719385874718951, 0.002941404562519108, 1.1030259382231975, 0.003309084920435982]
[0.00021696413016617766, 3.055707901365759e-06, 0.0011692811971389322, 8.824233041190619e-06, 0.003309084920435856, 1.7305541985949733e-05]
```

Figure 7. Outputs of Question 1-C

V. Question 1-D)

Calculate the matrix condition number of the coefficient matrix using row-sum norm.

```python
def find_condition_number(A):
    """Finds the condition number of the matrix A, given the inverse matrix"""



    n = len(A)


    max_in_rows = np.amax(np.abs(A), axis=1)
    max_in_normal = np.amax(A, axis=1)



    for i in range(n):          #Normalize matrix
        for j in range(n):
            if max_in_rows[i] == max_in_normal[i]:
                A[i][j] = A[i][j]/max_in_rows[i]
            else:
                A[i][j] = A[i][j]/-max_in_rows[i]

    max_in_rows =np.amax(np.abs(A).sum(axis=1)) #Find max in  rows with sum of abs
    A_inv = plu_inverse(np.array(A))

    max_in_inv_rows = np.amax(np.abs(A_inv).sum(axis=1))


    cond_A = max_in_inv_rows*max_in_rows
    print("Condition number of A: ", cond_A)
```

Figure 8. Main loop of Question 1-D

```
Condition number of A:  24493.53799919013
```

Figure 9. Output of Question 1-D

VI. Question 2)

Use least squares regression to fit polynomials of order 1, 2 and 3 to the data given in the table (Use Gauss elimination with partial pivoting in the solution of linear equation systems). Compute the correlation coefficient for each fit. Plot 1st, 2nd, and 3rd order polynomial fits and the given discrete data.

```python
def gauss_elimination(A, b):

    temp_mat = np.c_[A, b] #combine A and b into one matrix


    n = temp_mat.shape[0] #number of rows

    #Loop over rows
    for i in range(n):


        p = np.abs(temp_mat[i:, i]).argmax() #pivot index


        p += i #pivot index with respect to original matrix


        if p != i:                              #swap rows
            temp_mat[[p, i]] = temp_mat[[i, p]]


        factor = temp_mat[i+1:, i] / temp_mat[i, i]            #Eliminate all entries below the pivot
        temp_mat[i+1:] -= factor[:, np.newaxis] * temp_mat[i]


    x = np.zeros_like(b, dtype=np.double);

    #Back substitution
    x[-1] = temp_mat[-1,-1] / temp_mat[-1, -2]


    for i in range(n-2, -1, -1):
        x[i] = (temp_mat[i,-1] - np.dot(temp_mat[i,i:-1], x[i:])) / temp_mat[i,i]    #Solve for each row

    return x
```

Figure 10. Gauss Elimination for Question 2

```python
def fit_polynomials(x,y,degree):
    x_range = degree+4
    y_range = degree+1
    n = len(x)

    sum_x_list = []
    sum_y_list = []
    new_x  = np.zeros((y_range,y_range))
    new_y = np.zeros((y_range,1))

    for i in range(x_range):
        sum_x_list.append(np.sum(x**i))
    for i in range(y_range):
        sum_y_list.append(np.sum((x**i)*y))
    for i in range(y_range):
        try:
            new_x[i,0] = sum_x_list[i]
            new_x[i,1] = sum_x_list[i+1]
            new_x[i,2] = sum_x_list[i+2]
            new_x[i,3] = sum_x_list[i+3]
        except:
            pass
    for i in range(y_range):
        new_y[i,0] = sum_y_list[i]

    coefficients = gauss_elimination(new_x,new_y)
```

Figure 11. First part of the main loop of Question 2

```python
if degree == 1:
    f = lambda x: coefficients[1][0]*x + coefficients[0][0]
elif degree == 2:
    f = lambda x: coefficients[2][0]*x**2 + coefficients[1][0]*x + coefficients[0][0]
elif degree == 3:
    f = lambda x: coefficients[3][0]*x**3 + coefficients[2][0]*x**2 + coefficients[1][0]*x + coefficients[0][0]

plt.plot(np.arange(-0.1,2,0.01),[f(x) for x in np.arange(-0.1,2,0.01)],label=f"Order {degree} Fit",color="red")

plt.legend()

plt.scatter(x,y,s=10)
plt.show()

Sr = np.sum((y-f(x))**2)

y_hat = np.sum(y)/n

St = np.sum((y-y_hat)**2)

r = np.sqrt((St-Sr)/St)
table = np.array([[degree,Sr,St,r]])
print(tabulate(table,headers=["Degree","Sr","St","r"]))
```

Figure 12. Second part of the main loop of Question 2

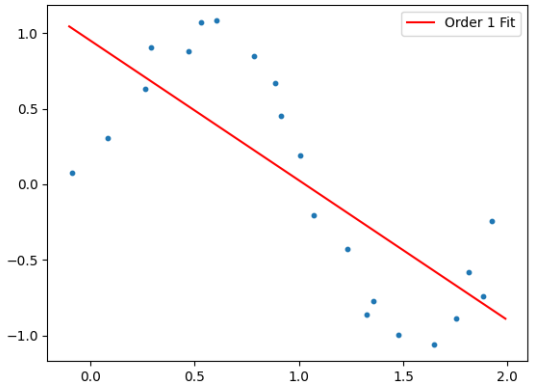

Figure 13. First order polynomial fit

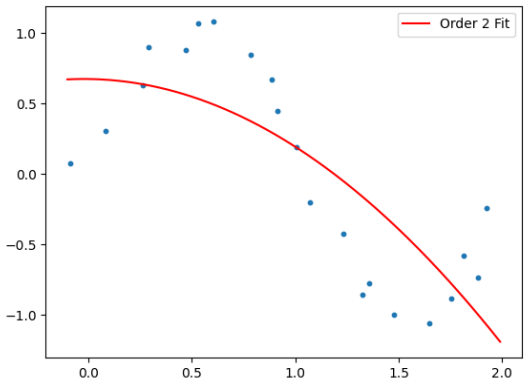

Figure 14. Second order polynomial fit

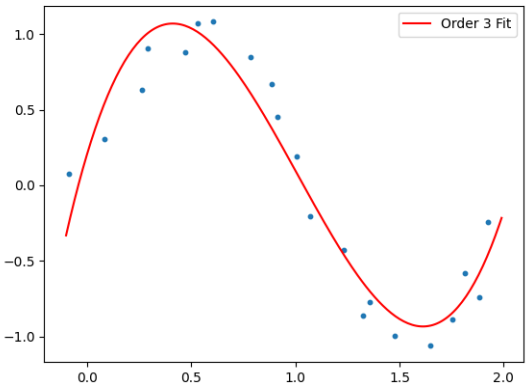

Figure 15. Third order polynomial fit

```
  Degree       Sr       St         r
--------  -------  ------  -------
       1  4.73188  11.223  0.76051
  Degree       Sr       St         r
--------  -------  ------  --------
       2  4.21969  11.223  0.789945
  Degree       Sr       St         r
--------  -------  ------  --------
       3  0.72483  11.223  0.967169
```

Figure 16. Outputs of Question 2

VII. Comments on Results of Question 2

With the knowledge that Figure 16 provided us, which shows correlation coefficients due to degrees of polynomials, the third degree of polynomial explains 96.7% of the variability of the data, which is better compared to other degrees of polynomials.

VIII. Question 3)

The variation of specific heats of air is presented in the table. By using this data, rebuild the given table for cp, cv and k between T=250K and T=500K with ΔT=10K steps by using Newton's divided difference interpolating polynomials or the Lagrange interpolating polynomials. Present your results in a new table.

```
def newton_poly(x,x_new,y):
    '''
    x: x-values
    x_new: x-value to be interpolated
    y: y-values
    '''

    n = len(y)
    coef = np.zeros([n, n])

    coef[:,0] = y

    for j in range(1,n):
        for i in range(n-j):
            coef[i][j] = \
           (coef[i+1][j-1] - coef[i][j-1]) / (x[i+j]-x[i])

    coef = coef[0,:]

    n = len(x) - 1
    p = coef[n]
    for k in range(1,n+1):
        p = coef[n-k] + (x_new -x[n-k])*p
    return p
```

Figure 17. Main loop of Question 3 (Newton's divided difference)

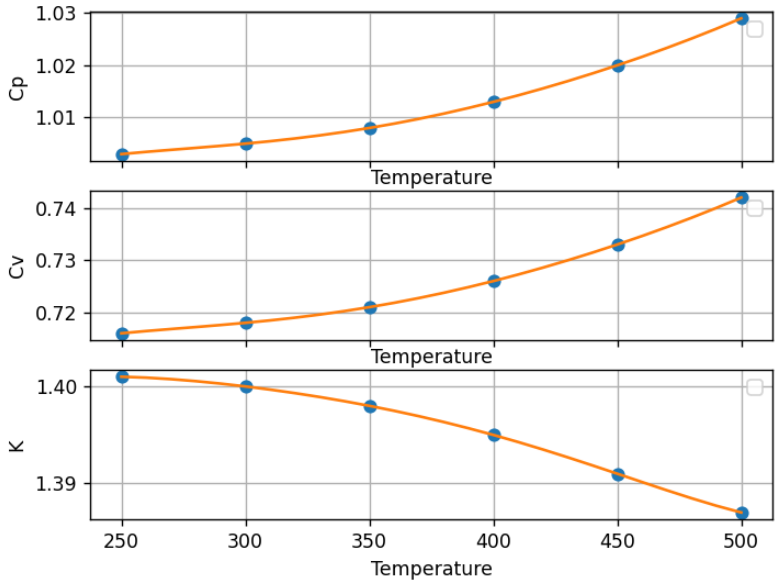

Figure 18. Graphical illustration of outputs of Question 3

Temperature	Cp	Cv	K
250	1.003	0.716	1.401
260	1.00343	0.716427	1.40091
270	1.00382	0.716816	1.40075
280	1.00419	0.717192	1.40054
290	1.00458	0.717581	1.40029
300	1.005	0.718	1.4
310	1.00547	0.718466	1.39967
320	1.00599	0.71899	1.39931
330	1.00658	0.719583	1.39891
340	1.00725	0.720251	1.39847
350	1.008	0.721	1.398
360	1.00883	0.721832	1.39749
370	1.00975	0.722748	1.39693
380	1.01075	0.723749	1.39633
390	1.01183	0.724834	1.39569
400	1.013	0.726	1.395
410	1.01425	0.727246	1.39427
420	1.01557	0.728571	1.3935
430	1.01697	0.729972	1.39269
440	1.01845	0.731448	1.39185
450	1.02	0.733	1.391
460	1.02163	0.734629	1.39014
470	1.02334	0.736337	1.38929
480	1.02513	0.73813	1.38846
490	1.02701	0.740014	1.38769
500	1.029	0.742	1.387

Figure 19. New table for interpolated temperature values