

ALADS - Assignment - 2

Chenglong Li

April 2024

1 Part A

1.1 Question 1

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 5 & 10 \\ 3 & 10 & 16 \end{pmatrix}, b = \begin{pmatrix} 3 \\ 7 \\ 13 \end{pmatrix}$$

1.1.1 a

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 5 & 10 \\ 3 & 10 & 16 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 0 & 4 & 5 \end{pmatrix}$$

row 2 \leftarrow row 2 $- 2 \times$ row 1

row 3 \leftarrow row 3 $- 3 \times$ row 1

then

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 0 & 4 & 5 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 0 & 0 & -9 \end{pmatrix}$$

row 3 \leftarrow row 3 $- 4 \times$ row 2

So we can know L, U, P

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 4 & 1 \end{pmatrix}$$

$$U = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 0 & 0 & -9 \end{pmatrix}$$

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

1.1.2 b

we can know $A = LU$

$$\begin{aligned} Ax &= b \\ \implies LUx &= b \end{aligned}$$

we set $Ux = C$ we can get

$$\begin{aligned} Ax &= b \\ \implies LUx &= b \\ \implies LC &= b \end{aligned}$$

we set $C = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$, we can know

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 4 & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 7 \\ 13 \end{pmatrix}$$

so we can know

$$\begin{aligned} c_1 &= 3 \\ c_2 &= 1 \\ c_3 &= 0 \\ C &= \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \\ 0 \end{pmatrix} \end{aligned}$$

according $Ux = C$, we can know

$$\begin{aligned} Ux &= \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 0 & 0 & -9 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \\ &= \begin{pmatrix} 3 \\ 1 \\ 0 \end{pmatrix} \end{aligned}$$

we can get

$$\begin{aligned} x_1 &= 1 \\ x_2 &= 1 \\ x_3 &= 0 \\ x &= \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \end{aligned}$$

1.1.3 c

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 5 & 10 \\ 3 & 10 & 16 \end{pmatrix}$$

Matrix A is symmetric. So we can use Cholesky-decomposition. According **a** we can know

$$U = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 0 & 0 & -9 \end{pmatrix}$$
$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 4 & 1 \end{pmatrix}$$

we take diagonal elements out of U .

$$U = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -9 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{pmatrix}$$
$$D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -9 \end{pmatrix}$$
$$L^T = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{pmatrix}$$

I think this matrix A can not be used Cholesky-decomposition. Because matrix A has negative eigenvalues.

1.2 Question 2

Condition Number is used to measure the sensitivity of a function, matrix, or problem, i.e., the degree to which small changes in the input affect the output. In numerical computation, condition numbers help us understand the magnitude of errors that may occur during computation. In particular, when solving systems of linear equations or performing matrix operations, high condition numbers can lead to numerical instability and significant errors, while low condition numbers usually indicate that the computation will be more stable and reliable.

```
import numpy as np
import pandas as pd
A = np.array([
    [0,6, -2, 0, 0, 0],
    [0,4, 1, 2, 2, 0],
    [0, 0, 0, 0, 2, 4],
```

```

    [1, -1, 0, 1, 0, 0],
    [0, 0, 1, 0, -1, 1],
    [0, 0, 0, 1, -1, 1]
])
A_condition_number = np.linalg.cond(A)
A_condition_number

```

Condition Number A = 12.28.

If we use

$$\epsilon_{output} \approx K \times \epsilon_{input}$$

Therefore, the expected relative error in the solution can be estimated to be 12.28%, assuming that the errors are propagated in the worst-case manner as indicated by the condition number.

2 Part B

2.1 Question 3

2.1.1 a

Output look at Figure 1.

```

[1.      0.76829311  0.60958208 ... 0.64992538  0.6806953  0.67765189]
[2.      1.53658622  1.21916416 ... 1.29985076  1.3613906  1.35530379]
[3.      2.30487933  1.82874624 ... 1.94977614  2.04208591  2.03295568]
[4.      3.07317244  2.43832832 ... 2.59970152  2.72278121  2.71060757]
[5.      3.84146555  3.0479104  ... 3.2496269  3.40347651  3.38825947]
[6.      4.60975866  3.65749247 ... 3.89955228  4.08417181  4.06591136]
[7.      5.37805177  4.26707455 ... 4.54947767  4.76486711  4.74356325]
[8.      6.14634488  4.87665663 ... 5.19940305  5.44556242  5.42121515]
[9.      6.91463799  5.48623871 ... 5.84932843  6.12625772  6.09886704]

[10.      7.6829311  6.09582079 ... 6.49925381  6.80695302
    6.77651893]

[11.      8.45122421  6.70540287 ... 7.14917919  7.48764832
    7.45417083]
...
[20.      15.36586221  12.19164158 ... 12.99850761  13.61390604
    13.55303786]
Method A =
103.87317199999987

```

Figure 1: Method A

2.1.2 b

Output look at Figure 2.

```

[1.      0.76829311 0.60958208 ... 0.64992538 0.6806953 0.67765189]
[2.      1.53658622 1.21916416 ... 1.29985076 1.3613906 1.35530379]
[3.      2.30487933 1.82874624 ... 1.94977614 2.04208591 2.03295568]
[4.      3.07317244 2.43832832 ... 2.59970152 2.72278121 2.71060757]
[5.      3.84146555 3.0479104 ... 3.2496269 3.40347651 3.38825947]
[6.      4.60975866 3.65749247 ... 3.89955228 4.08417181 4.06591136]
[7.      5.37805177 4.26707455 ... 4.54947767 4.76486711 4.74356325]
[8.      6.14634488 4.87665663 ... 5.19940305 5.44556242 5.42121515]
[9.      6.91463799 5.48623871 ... 5.84932843 6.12625772 6.09886704]

[10.      7.6829311 6.09582079 ... 6.49925381 6.80695302
6.77651893]
[11.      8.45122421 6.70540287 ... 7.14917919 7.48764832
7.45417083]
...
[20.      15.36586221 12.19164158 ... 12.99850761 13.61390604
13.55303786]
Method B=
0.1048270000000023

```

Figure 2: Method B

2.1.3 c

Output look at Figure 3.

```

[[ 1.      0.76829311 0.60958208 ... 0.64992538 0.6806953
0.67765189]
 [ 2.      1.53658622 1.21916416 ... 1.29985076 1.3613906
1.35530379]
 [ 3.      2.30487933 1.82874624 ... 1.94977614 2.04208591
2.03295568]
 ...
[18.      13.82927599 10.97247742 ... 11.69865685 12.25251544
12.19773408]
[19.      14.5975691 11.5820595 ... 12.34858223 12.93321074
12.87538597]
[20.      15.36586221 12.19164158 ... 12.99850761 13.61390604
13.55303786]]
Method C =
4.921498999999983

```

Figure 3: Method C

2.1.4 d

Output look at Figure 4.

```

(0, 0)      1.0
(1, 0)      0.768293110432165
(2, 0)      0.6095820791249414
(3, 0)      0.6851168829281282
(4, 0)      0.7342091994648069
(5, 0)      0.4928309004094137
(6, 0)      0.6533116917506447
(7, 0)      0.5087099392153661
(8, 0)      0.5450688032829656
(9, 0)      0.6219653116391878
(10, 0)     0.6297613967849265
(11, 0)     0.3783340222043075
(12, 0)     0.5770938342077814
(13, 0)     0.44497092554125506
(14, 0)     0.5473212672061203
(15, 0)     0.4978016244839013
(16, 0)     0.5806368349376895
(17, 0)     0.377386270724878
(18, 0)     0.5720889130969798
(19, 0)     0.5339189986000271
(20, 0)     0.42000621139334976
(21, 0)     0.3420306337893293
(22, 0)     0.4307727965556303
(23, 0)     0.4717843631489195
(24, 0)     0.5441995353282013
...
(4718, 19)  13.613906039539014
(4719, 19)  13.553037864422999
Method D =
0.01947499999999991

```

Figure 4: Method D

2.2 Question 4

Matrix A is sparse.

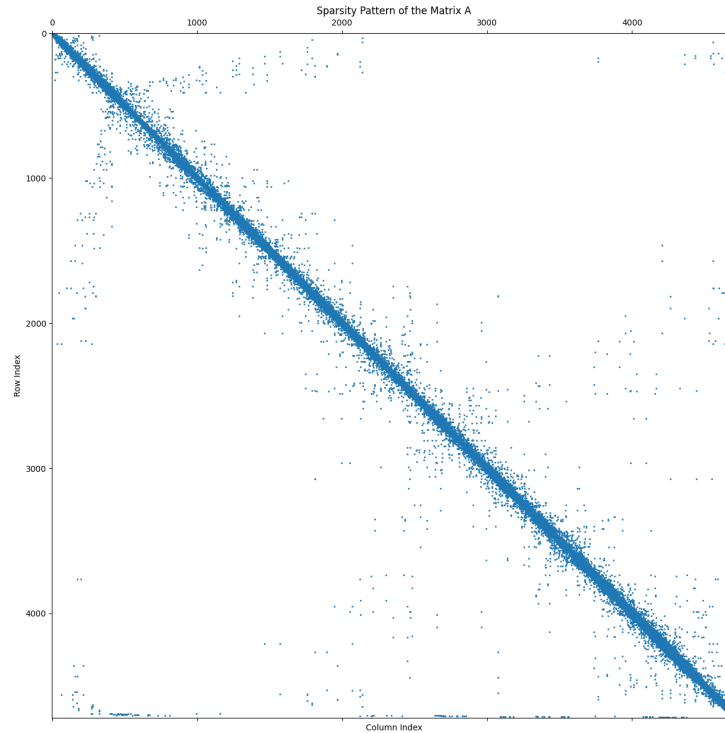


Figure 5: Sparsity Pattern of the Matrix A

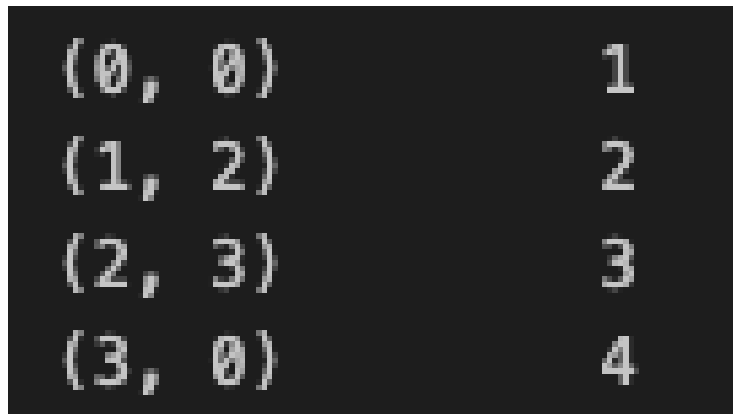
2.3 Question 5

The sparse matrix format is a storage scheme that significantly reduces memory usage when dealing with matrices that have a large number of zero entries. Instead of storing all entries including the zeros, the sparse format only stores the non-zero elements and their locations.

- **Compressed Sparse Row (CSR):** It stores the matrix by three one-dimensional arrays that represent non-zero values, the extents of rows, and column indices.
- **Compressed Sparse Column (CSC):** It is similar to CSR but focuses on column operations. It uses three arrays to represent column indices, row extents, and non-zero values.
- **Coordinate List (COO):** It stores a list of (row, column, value) tuples.

We use CSR to store the little sparse matrix.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \\ 4 & 0 & 0 & 0 \end{pmatrix}$$



(0, 0)	1
(1, 2)	2
(2, 3)	3
(3, 0)	4

Figure 6: Storing sparse matrix using CSR

2.4 Appendix

Code for Assignment 2 ([Github repository](#))