Numerical Method

National Cheng Kung University Department of Engineering Science

Instructor: Chi-Hua Yu

HW 4

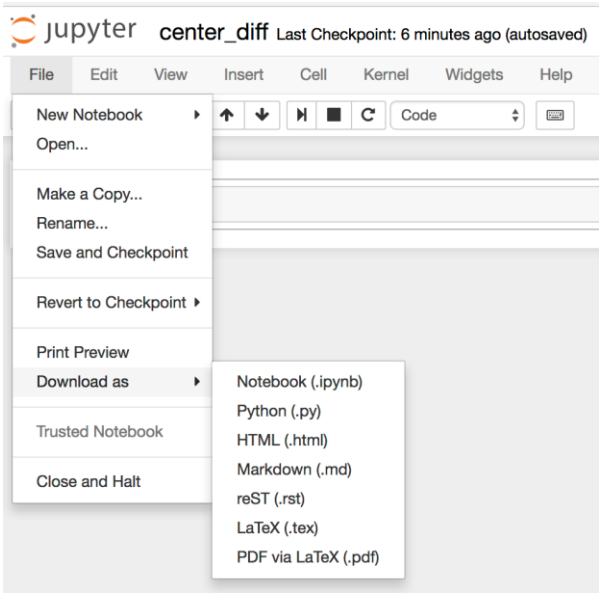
Programming, Due 09:00, Wednesday, March 30th , 2022

注意事項：

1. Homework 的時間為公布題目後至下次上課前結束(上課當天 09:00)。
2. 請在規定的時段內完成作業，並用你的學號與 HW number 做一個檔案夾 (e.g., N96091350_HW3), 將你的全部 ipynb 檔放入檔案夾，壓縮後上傳至課程網站 (e.g., N96091350_HW3.zip)，超過期限後不予補交。

Homework Submission Procedure (請仔細閱讀)

1. You should submit your Jupyter notebook and Python script (`*.py`, in Jupyter, click File, Download as, Python (`*.py`)).



2. Name a folder using your student id and lab number (e.g., n96081494_HW1), put all the pdf and all the Jupyter notebooks and python scripts into the folder and zip the folder (e.g., n96081494_HW1.zip).

3. Submit your lab directly through the course website.

1. (20%) Name your pdf file HW4_student id (e.g., HW4_n96081494.pdf). For compute process, please write them in a professional format and submit a pdf file. Please use Gauss–Jordan elimination to solve the following equations:

$$4x_1 + 3x_2 - 5x_3 = 2,$$
$$-2x_1 - 4x_2 + 5x_3 = 5,$$
$$8x_1 + 8x_2 = -3.$$

2. (40%) Name your Jupyter notebook `gauss_ jordan_elimination.ipynb` and Python script `gauss_ jordan_elimination.py`. Write a Python program to solve the equations by using the Gauss elimination method.

$$Ax = y$$
$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

The Gauss–Jordan elimination method is first turning the matrix *A* into an upper-triangular form. Then, the upper triangular form is turned to a diagonal form to solve the system of equations.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1^t \\ y_2^t \\ y_3^t \\ y_4^t \end{bmatrix}$$

Below is the running example

Sample 1

```
a = np.array([[1, 2, 3],[3, 4, 5],[3, 5, 5]])
y = np.array([2, 2, 5])

diagonal matrix:
array([[ 1. ,  0. ,  0. , -2.5],
       [ 0. ,  1. ,  0. ,  3. ],
       [-0. , -0. ,  1. , -0.5]])

x:
[ -2.5,  3. , -0.5]
```

Sample 2

```
a = np.array([[1, 2, 3, 4],[5, 4, 3, 2],[2, 1, 2, 4],[2, 1, 3, 4]])
y = np.array([4, 8, 5, 6])

diagonal matrix:
array([[ 1. ,  0. ,  0. ,  0. ,  1.4],
       [ 0. ,  1. ,  0. ,  0. , -0.6],
       [ 0. ,  0. ,  1. ,  0. ,  1. ],
       [-0. , -0. , -0. ,  1. ,  0.2]])
```

```
x:
[ 1.4, -0.6,  1. ,  0.2]
```

3. (40%) Name your Jupyter notebook `LU_decomposition.ipynb` and Python script `LU_decomposition.py`. Write a Python program to solve the equations by using the LU decomposition. The LU decomposition method aims to turn A into the product of two matrices L and U, where L is a lower triangular matrix while U is an upper triangular matrix.

$$Ax = LU_x = y \rightarrow \begin{bmatrix} l_{1.1} & 0 & 0 & 0 \\ l_{2.1} & l_{2.2} & 0 & 0 \\ l_{3.1} & l_{3.2} & l_{3.3} & 0 \\ l_{4.1} & l_{4.2} & l_{4.3} & l_{4.4} \end{bmatrix} \begin{bmatrix} u_{1.1} & u_{1.2} & u_{1.3} & u_{1.4} \\ 0 & u_{2.2} & u_{2.3} & u_{2.4} \\ 0 & 0 & u_{3.3} & u_{3.4} \\ 0 & 0 & 0 & u_{4.4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ m_{2.1} & 1 & 0 & 0 \\ m_{3.1} & m_{3.2} & 1 & 0 \\ m_{4.1} & m_{4.2} & m_{4.3} & 1 \end{bmatrix} \begin{bmatrix} u_{1.1} & u_{1.2} & u_{1.3} & u_{1.4} \\ 0 & u_{2.2} & u_{2.3} & u_{2.4} \\ 0 & 0 & u_{3.3} & u_{3.4} \\ 0 & 0 & 0 & u_{4.4} \end{bmatrix}$$

Where the elements below the diagonal elements ($m_{2.1}$, $m_{3.1}$, $m_{3.2}$, $m_{4.1}$, $m_{4.2}$, $m_{4.3}$) are the multipliers that multiply the pivot equations to eliminate the elements during the calculation. Now, we define $U_x = M$, then the above equations become:

$$LM = y$$

$$\begin{bmatrix} l_{1.1} & 0 & 0 & 0 \\ l_{2.1} & l_{2.2} & 0 & 0 \\ l_{3.1} & l_{3.2} & l_{3.3} & 0 \\ l_{4.1} & l_{4.2} & l_{4.3} & l_{4.4} \end{bmatrix} M = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

After we solve for M, we can use $U_x = M$ to solve for *x*.

$$U_x = M$$

$$\begin{bmatrix} u_{1.1} & u_{1.2} & u_{1.3} & u_{1.4} \\ 0 & u_{2.2} & u_{2.3} & u_{2.4} \\ 0 & 0 & u_{3.3} & u_{3.4} \\ 0 & 0 & 0 & u_{4.4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix}$$

Below is the running example

Sample 1

```
a = np.array([[1, 2, 3],[3, 4, 5],[3, 5, 5]])
y = np.array([2, 2, 5]

u_matrix:
[[ 1.  2.  3.]
```

```
 [ 0. -2. -4.]
 [ 0.  0. -2.]]

l_matrix:
[[1.  0.  0. ]
 [3.  1.  0. ]
 [3.  0.5 1. ]]

x:
[-2.5,  3. , -0.5]
```

Sample 2

```
a = np.array([[1, 2, 3, 4],[5, 4, 3, 2],[2, 1, 2, 4],[2, 1, 3, 4]])
y = np.array([4, 8, 5, 6])

u_matrix:
[[  1.    2.    3.    4. ]
 [  0.   -6.  -12.  -18. ]
 [  0.    0.    2.    5. ]
 [  0.    0.    0.   -2.5]]

l_matrix:
[[1.  0.  0.  0. ]
 [5.  1.  0.  0. ]
 [2.  0.5 1.  0. ]
 [2.  0.5 1.5 1. ]]

x:
[ 1.4, -0.6,  1. ,  0.2]
```