Homework 6:

From

$$\begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix}^3 = \begin{bmatrix} 20 & 30 \\ 40 & 50 \end{bmatrix}$$

6.1 Use Scipy/Numpy to find x_1, x_2, x_3, and x_4 (you can use any Scipy/Numpy function for 6.1)
Please note that you can compare the answer that you get from 6.1 with the answer from homework 1, 1.4 (they are the same question, but at that time you were asked to use Excel solver to solve it).

6.2 Use Newton-Jacobian source code given in the lecture to find x_1, x_2, x_3, and x_4
The source code is in L06_SysNon.ipynb. Function name is newton_jacobian and jacobian. However, this time, your input f must be Numpy array of function, not list of function (as used in the lecture). Do modification as needed to make code work. To create Numpy array of function, you may use hstack, vstack, c_[ ], r_[ ] as appropriate.

6.3 Computationally counting the number of floating point operation using in 6.2. For example, the code

```
J[i,j] = (f[i](*x_plus) - f[i](*x_minus))/eps * 0.5
```

has

1) calculation of function with *x_plus input
2) calculation of function with *x_minus input
3) minus, division (by eps) and multiplication (by 0.5)

Thus, the number of floating point operation is
    2 * (number of floating point operation in function f) + 3
Put counter inside the for loop to do the counting.

6.4 Calculate the norm of Jacobian matrix in 6.2 in each iteration, and plot the norm of Jacobian versus number of iteration.

6.5 Calculate the error in each iteration. For example, in one iteration you have
$x_1 = 3$
$x_2 = 2.5$
$x_3 = 1.5$
$x_4 = 4.5$

The the error is the norm of the matrix calculated from $\begin{bmatrix} 3 & 2.5 \\ 1.5 & 4.5 \end{bmatrix}^3 - \begin{bmatrix} 20 & 30 \\ 40 & 50 \end{bmatrix}$ (which is called error matrix). The error matrix is 2x2. The norm is just each term in the error matrix square, add together, and then take the square root (root sum-squared of each element).
After you get error in each iteration, plot it against the number of iteration. You must have, error in each iteration in the vertical axis, and the number of iteration in the horizontal axis.

6.6) At your one choice, create the system of non-linear equations which consist of 4 equations and 4 unknown. Explicitly specify each of those 4 equations. At least, 2 of your equation must be non-linear equation. Then,

6.6a) use your own Newton-Jacobian method to solve it. You must either create your own function or use algorithm given in the lecture. You must not use scipy or other automatic non-linear system solver in 6.6a)

6.6b) plot the error from each iteration against the number of iteration. The error is to be found by substitute your answer in that particular iteration in to all 4 functions and calculate absolute value of left-hand-side minus right-hand-side.

6.6c) use Scipy to solve it.

6.6d) use Sympy to get the analytical solution for Jacobian matrix of the system of non-linear equation that you created.

6.6e) Show the Jacobian matrix that is evaluated at the initial guess of you x vector ($\vec{x}$).

6.7) From the data

x	y
0	100
1	102
2	104
3	106
4	108
5	110
6	112
7	114

Use the closed-form solution of least-square regression learnt in class which is

$$(\boldsymbol{X}^T\boldsymbol{X})\hat{\boldsymbol{\beta}} = \boldsymbol{X}^T\boldsymbol{y}$$

To get the coefficient for the case where

6.7a) Linear equation is used to fit x and y such that
$y = a_0 + a_1 x$ (coefficients to be found are a_0 and a_1)
6.7b) Second order polynomial equation is used to fit x and y such that
$y = a_0 + a_1 x + a_2 x^2$(coefficients to be found are a_0, a_1 and a_2)
Do not use Excel, sklearn, or other automatic least-square regression libraries. Use Numpy and Scipy that are for creating matrix and solving system of linear equations. Do not use scipy that is for solving regression problem.