

# How to use Google Colab

Masaaki Nagahara

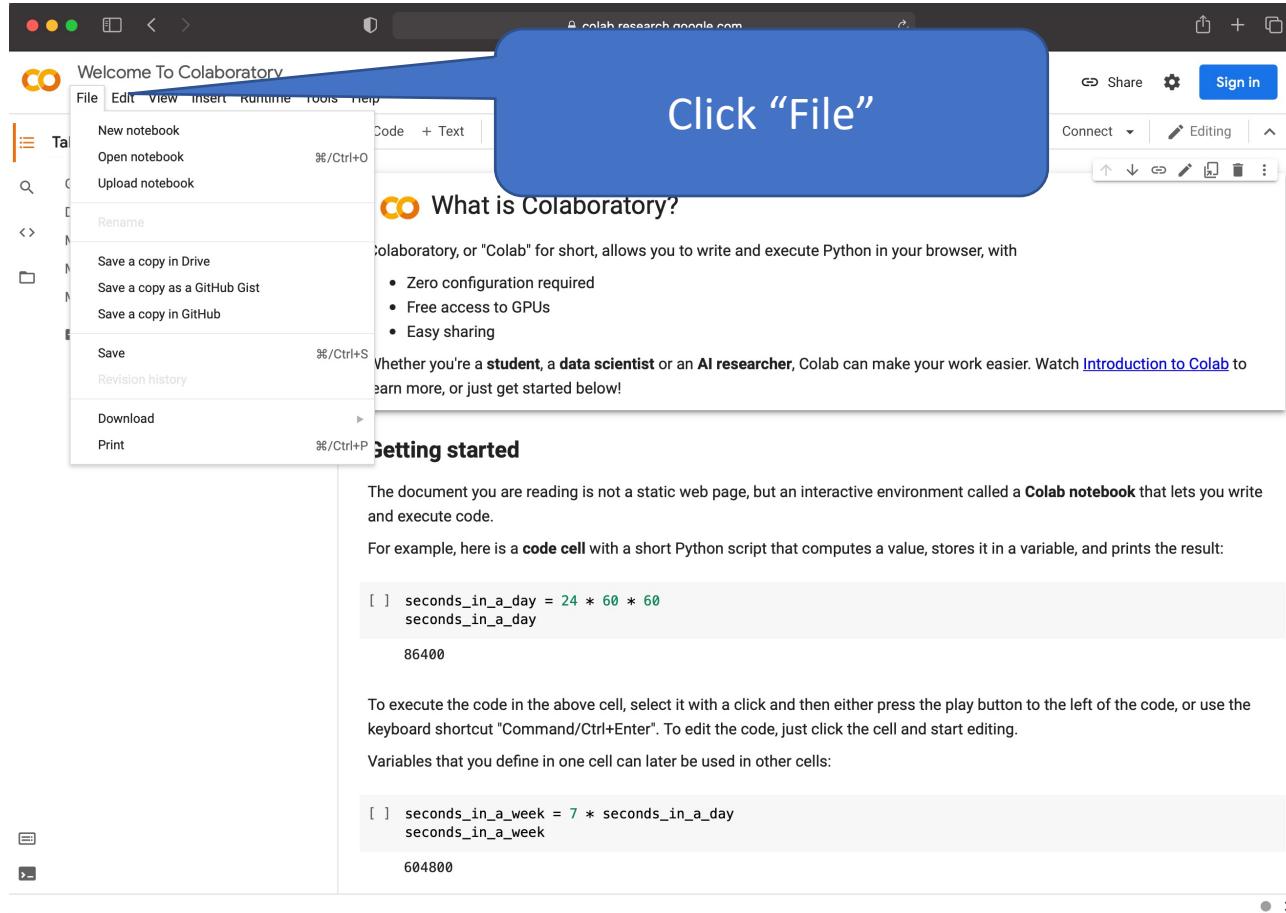
The University of Kitakyushu

# Open your web browser

<https://colab.research.google.com>

The screenshot shows the Google Colaboratory interface. At the top, there's a navigation bar with icons for file operations, a search bar containing 'colab.research.google.com', and user account options ('Share', 'Sign in'). Below the bar is a toolbar with buttons for 'Code', 'Text', 'Copy to Drive', 'Connect', and 'Editing'. On the left, a sidebar titled 'Table of contents' lists sections like 'Getting started', 'Data science', 'Machine learning', 'More Resources', and 'Machine Learning Examples'. The main content area displays the 'What is Colaboratory?' page, which includes a brief introduction, a bulleted list of features (zero configuration required, free access to GPUs, easy sharing), and a call to action to watch the 'Introduction to Colab' video. Below this, a section titled 'Getting started' is expanded, showing a code cell that calculates the number of seconds in a day (24 \* 60 \* 60) and prints the result (86400). A note explains how to execute code by selecting it and pressing Command/Ctrl+Enter. Further down, another code cell calculates the number of seconds in a week (7 \* seconds\_in\_a\_day) and prints the result (604800). The bottom of the interface shows standard window control buttons (minimize, maximize, close).

# .ipynb file (Jupyter Notebook) for Lecture 2



Welcome To Colaboratory

File Edit View Insert Runtime Tools Help

New notebook

**Open notebook**

Upload notebook

Rename

Save a copy in Drive

Save a copy as a GitHub Gist

Save a copy in GitHub

Save

Revision history

Download

Print

Code + Text

Share Sign in

What is Colaboratory?

Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch [Introduction to Colab](#) to learn more, or just get started below!

**Setting started**

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
[ ] seconds_in_a_day = 24 * 60 * 60  
seconds_in_a_day  
86400
```

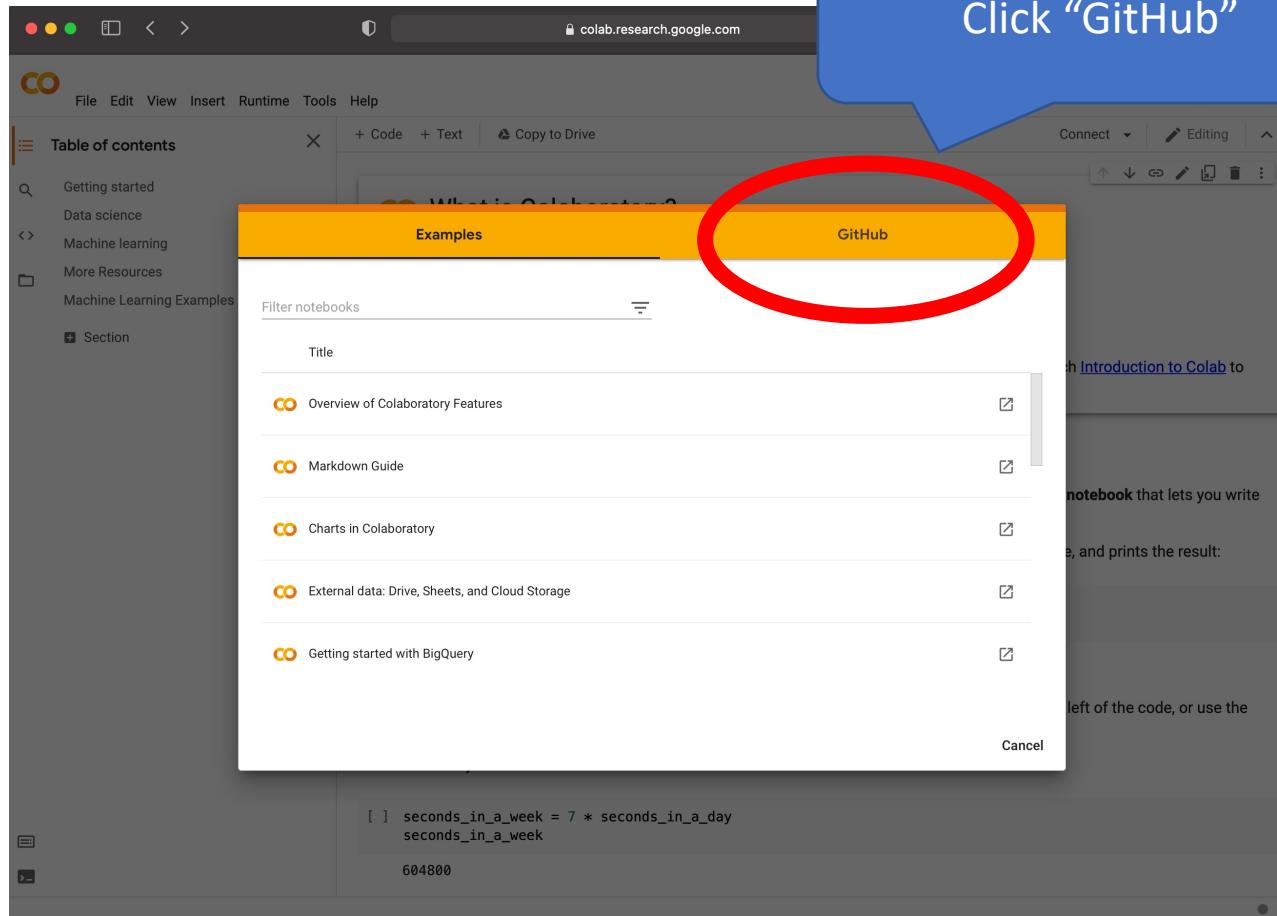
To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
[ ] seconds_in_a_week = 7 * seconds_in_a_day  
seconds_in_a_week  
604800
```

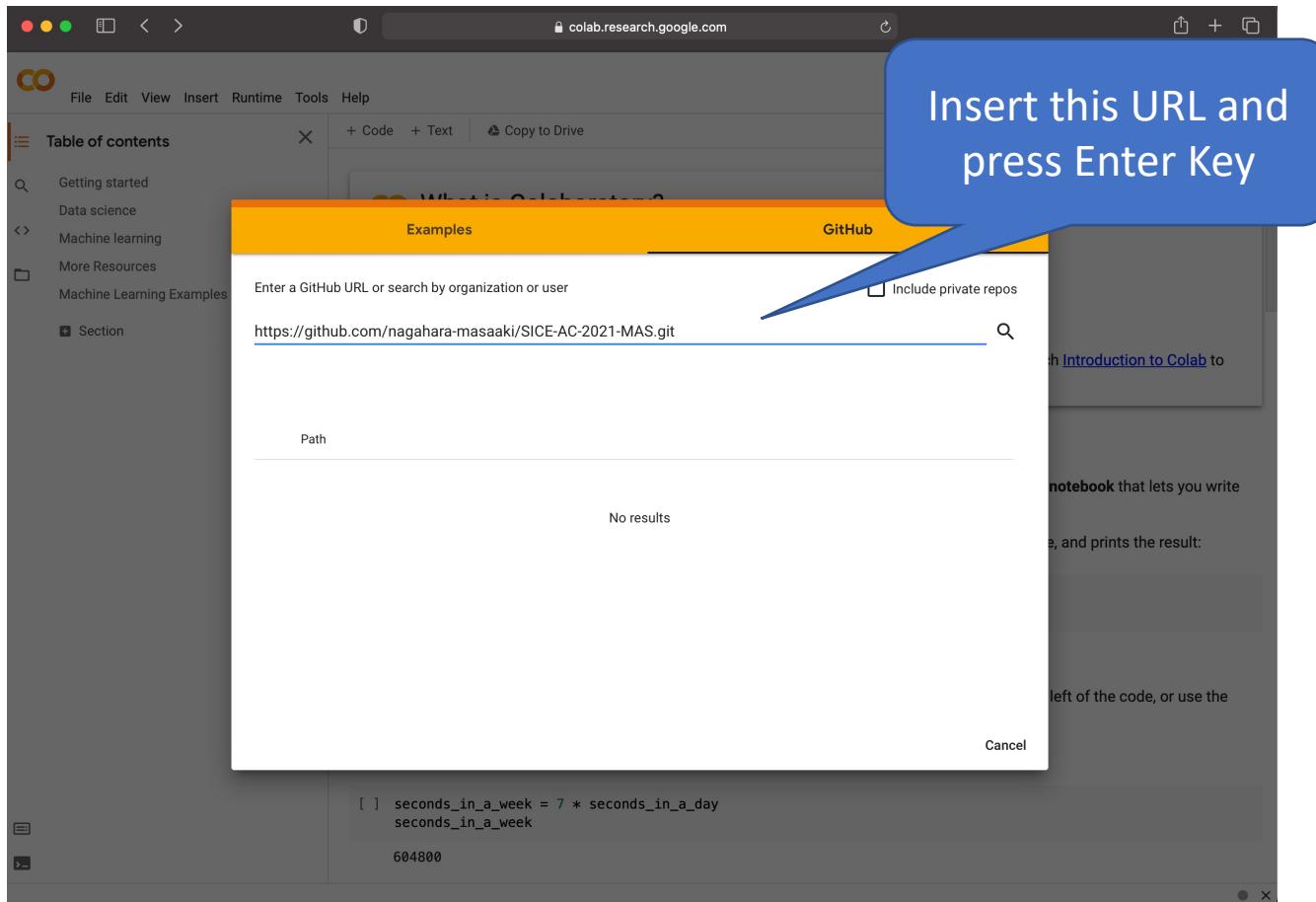
Choose “Open notebook”

# Open GitHub file

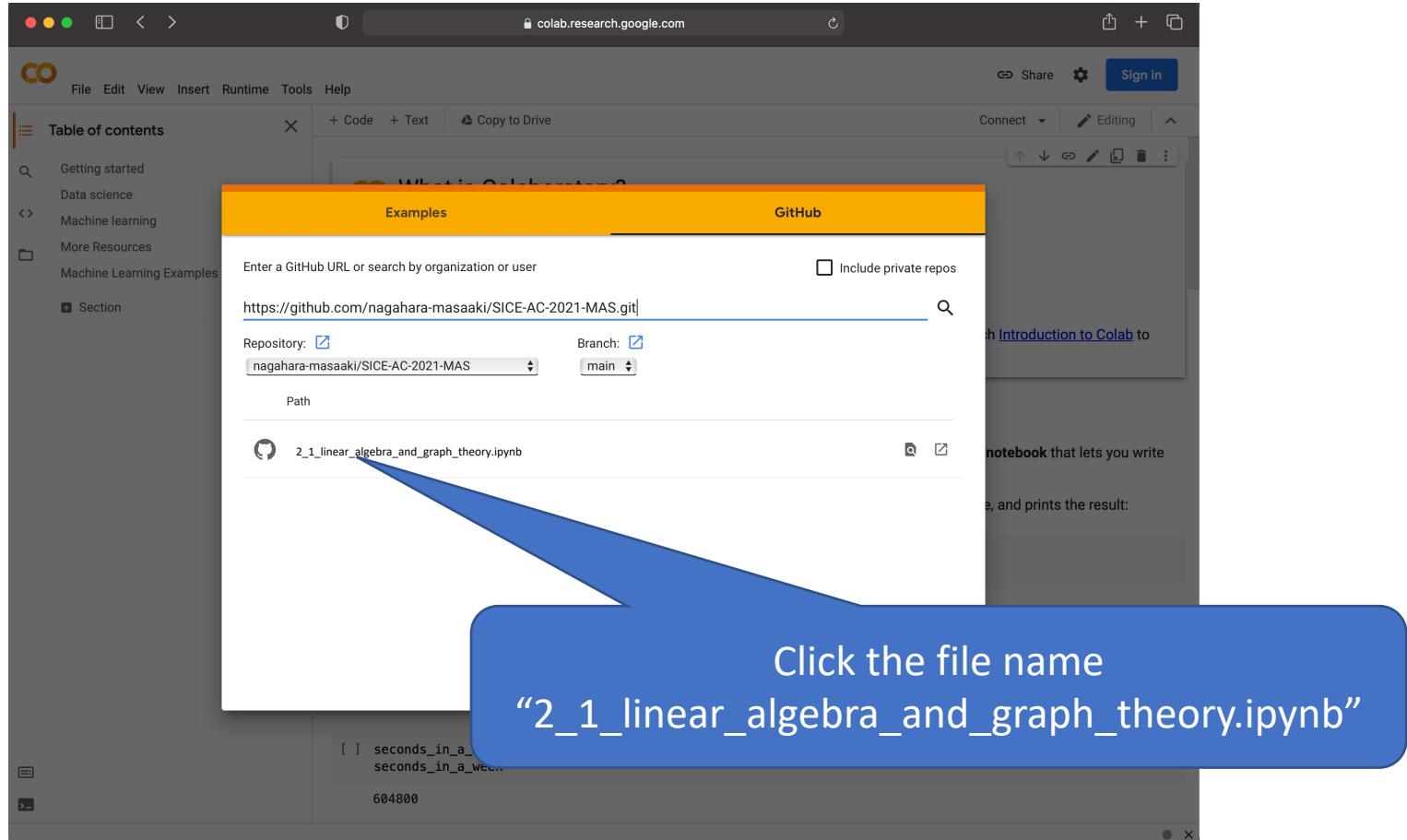


# Enter GitHub URL

<https://github.com/nagahara-masaaki/SICE-AC-2021-MAS.git>



# Choose the .ipynb file



# You will see

The screenshot shows a Google Colab notebook interface. The title bar reads "Springer Linear Algebra.ipynb". The menu bar includes File, Edit, View, Insert, Runtime, Tools, and Help. On the right, there are Share, Settings, and Sign in buttons. The main content area has a sidebar with sections like "+ Code", "+ Text", and "Copy to Drive". A search bar is present. The main content starts with a section titled "2. Linear Algebra and Graph Theory".  
**Abstract:** This lecture gives facts and theorems from linear algebra and graph theory that are important and used in control of multi-agent systems. Python codes are also provided, by which you obtain numerical results and well understand the mathematical facts.  
**Key Points**

- To analyze a graph (or a network), the **graph Laplacian** and the **Perron matrix** play an important role.
- The graph Laplacian (resp. the Perron matrix) has at least **one eigenvalue of 0** (resp. 1).
- The **connectivity** of a graph can be analyzed by the **multiplicity of the 0 (resp. 1) eigenvalue** of the graph Laplacian (resp. the Perron matrix).
- NumPy** and **NetworkX** packages are useful for computations in linear algebra and graph theory, respectively.

## 2.1 Coding in Python

In this lecture, we in particular introduce some facts and techniques in linear algebra and graph theory. For this purpose, we first need to **import useful packages** in Python. Namely, coding in Python starts with the following lines:

```
[ ] # Common initialization
import numpy as np
import networkx as nx
import numpy.linalg as LA
```

For simplicity, we set floating point precision of matrices and vectors to 2 by:

```
[ ] np.set_printoptions(precision=2)
```

## 2.2 Linear Algebra

# You can run Python code by



Press  button

Springer Linear Algebra.ipynb

File Edit View Insert Runtime Tools Help

Share Sign in

Abstract: This lecture gives facts and theorems from linear algebra and graph theory that are important and used in control of multi-agent systems. Python codes are also provided, by which you obtain numerical results and well understand the mathematical facts.

Key Points

- To analyze a graph (or a network), the **graph Laplacian** and the **Perron matrix** play an important role.
- The graph Laplacian (resp. the Perron matrix) has at least **one eigenvalue of 0** (resp. 1).
- The **connectivity** of a graph can be analyzed by the **multiplicity of the 0 (resp. 1) eigenvalue** of the graph Laplacian (resp. the Perron matrix).
- NumPy** and **NetworkX** packages are useful for computations in linear algebra and graph theory, respectively.

## 2.1 Coding in Python

In this lecture, we in particular introduce some facts and techniques in linear algebra and graph theory. For this purpose, we first need to **import useful packages** in Python. Namely, coding in Python starts with the following lines:

```
# Common initialization
import numpy as np
Run cell (F9/Ctrl+Enter)
cell has not been executed in this session
import numpy.linalg as LA
```

For simplicity, we set floating point precision of matrices and vectors to 2 by:

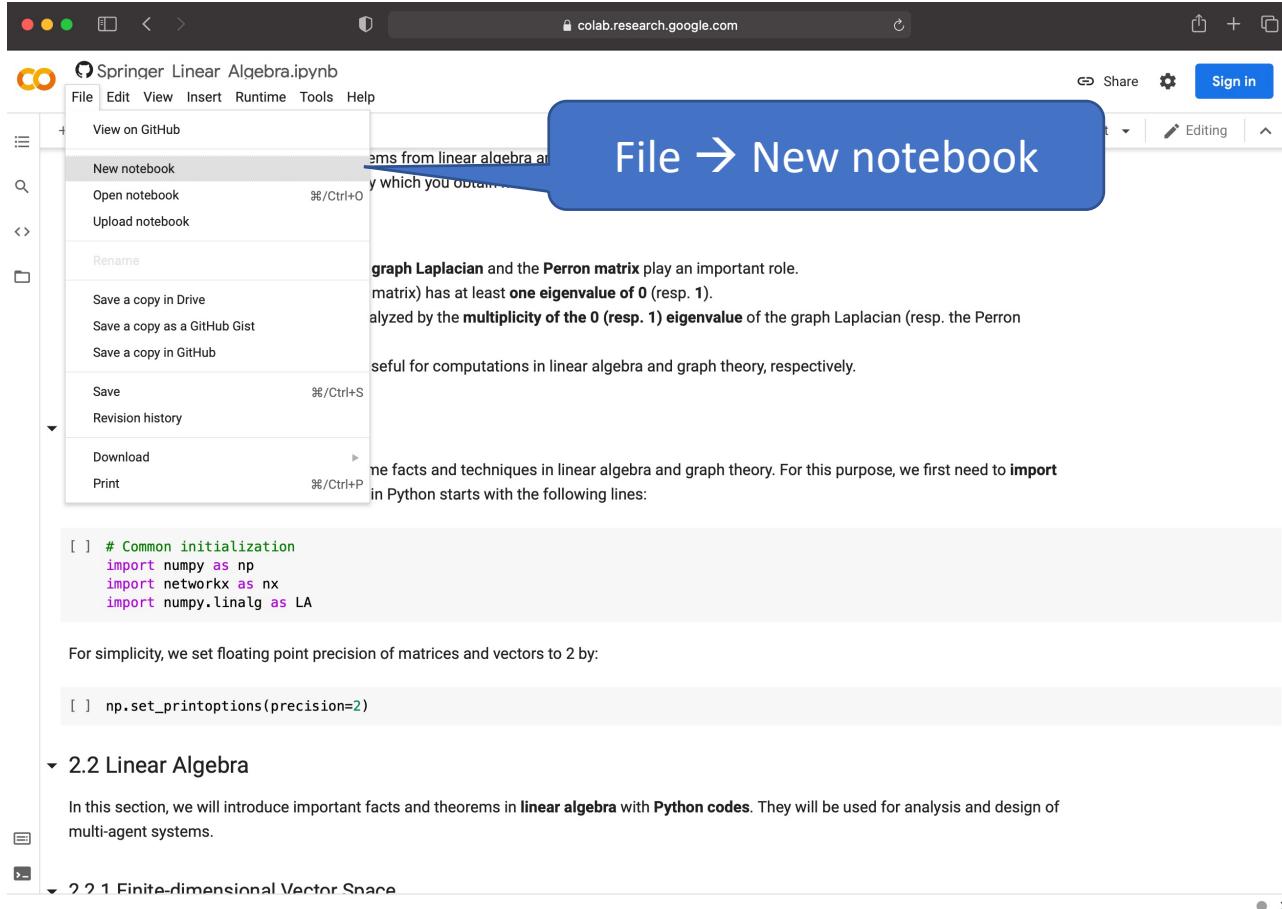
```
[ ] np.set_printoptions(precision=2)
```

## 2.2 Linear Algebra

In this section, we will introduce important facts and theorems in **linear algebra** with **Python codes**. They will be used for analysis and design of multi-agent systems.

### 2.2.1 Finite-dimensional Vector Space

# Open .py file (Lectures 3 and 4)



A screenshot of a web browser window showing the Google Colab interface. The address bar says "colab.research.google.com". The main content area displays a Jupyter notebook titled "Springer Linear Algebra.ipynb". A blue callout bubble points to the "New notebook" option in the "File" menu. The notebook content discusses linear algebra and graph theory, mentioning the Laplacian matrix and Perron matrix. It includes Python code for importing numpy and networkx. A section on linear algebra with Python codes is mentioned, along with a subsection on finite-dimensional vector spaces.

File → New notebook

graph Laplacian and the Perron matrix play an important role. matrix) has at least one eigenvalue of 0 (resp. 1). analyzed by the multiplicity of the 0 (resp. 1) eigenvalue of the graph Laplacian (resp. the Perron useful for computations in linear algebra and graph theory, respectively.

```
[ ] # Common initialization
import numpy as np
import networkx as nx
import numpy.linalg as LA
```

For simplicity, we set floating point precision of matrices and vectors to 2 by:

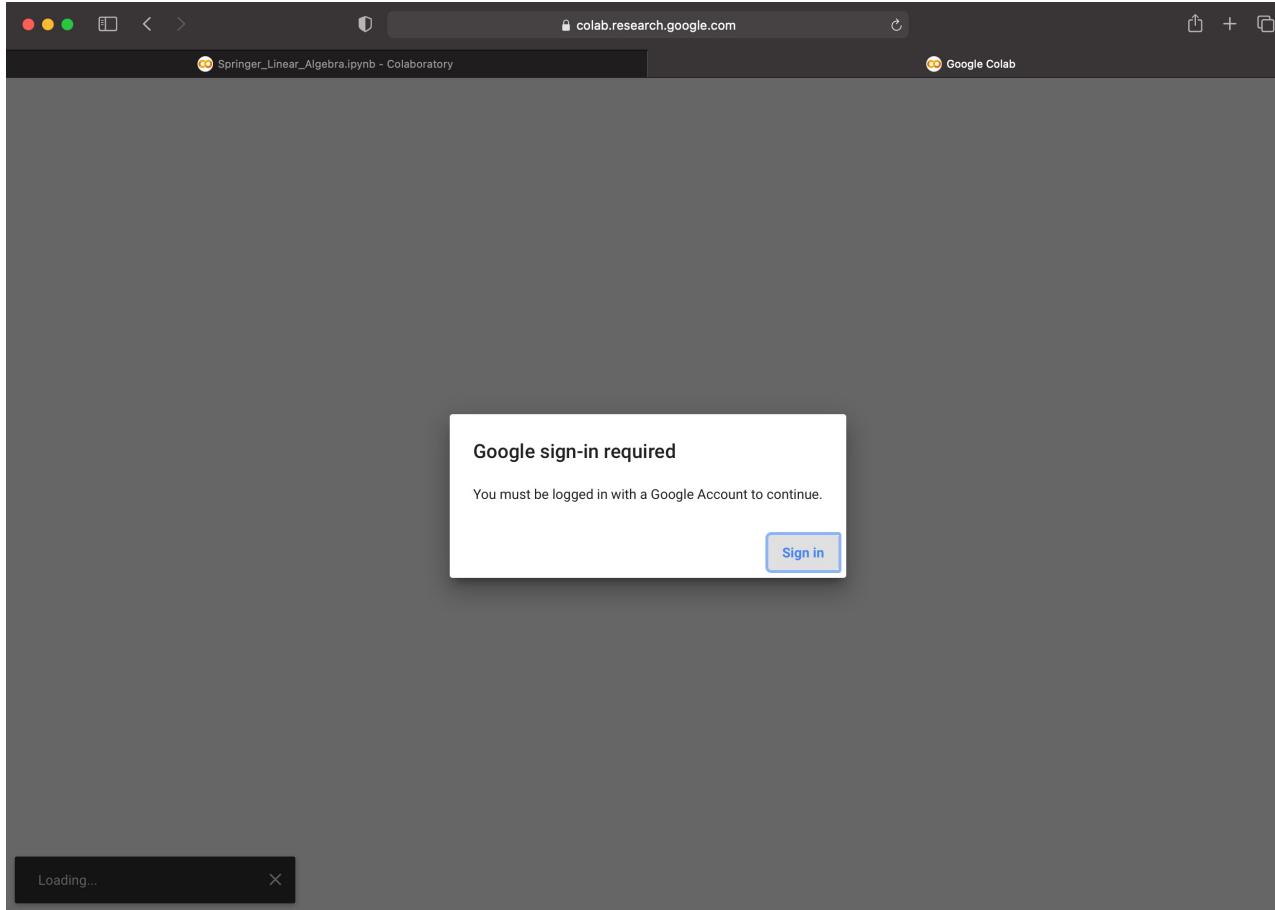
```
[ ] np.set_printoptions(precision=2)
```

2.2 Linear Algebra

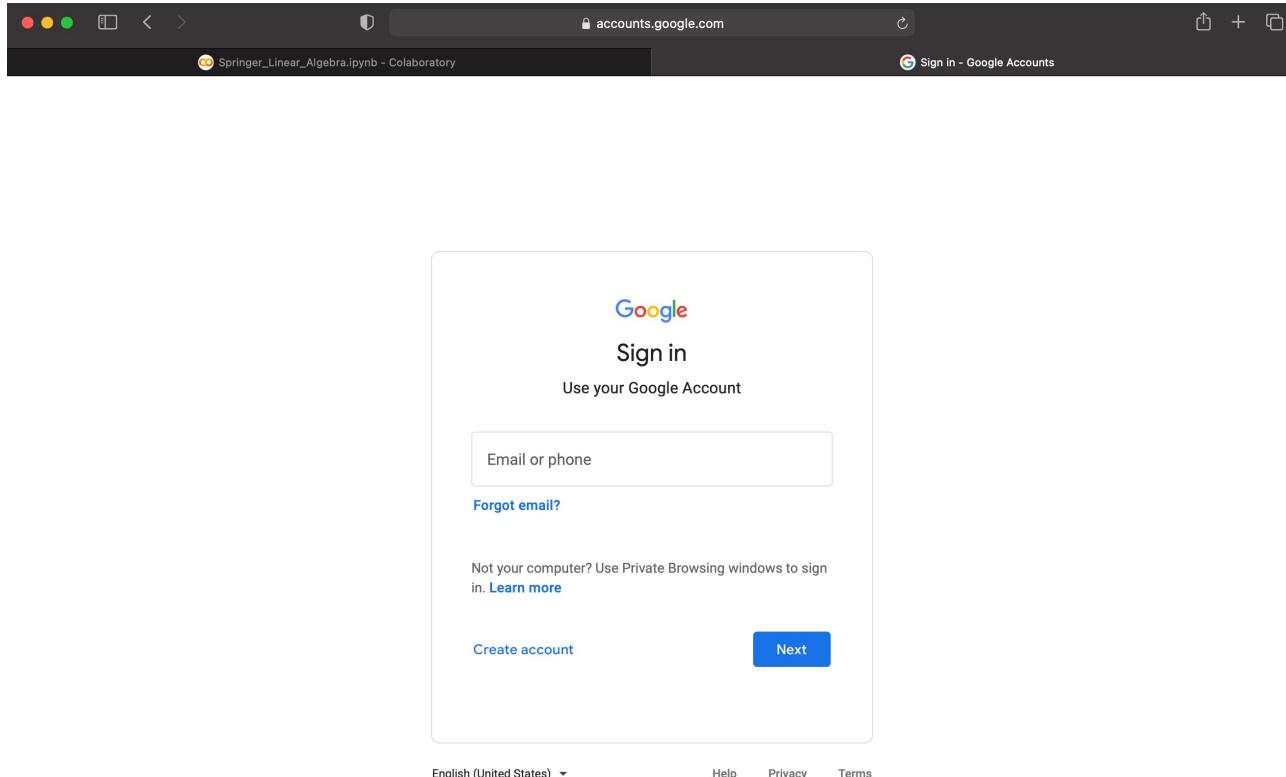
In this section, we will introduce important facts and theorems in linear algebra with Python codes. They will be used for analysis and design of multi-agent systems.

2.2.1 Finite-dimensional Vector Space

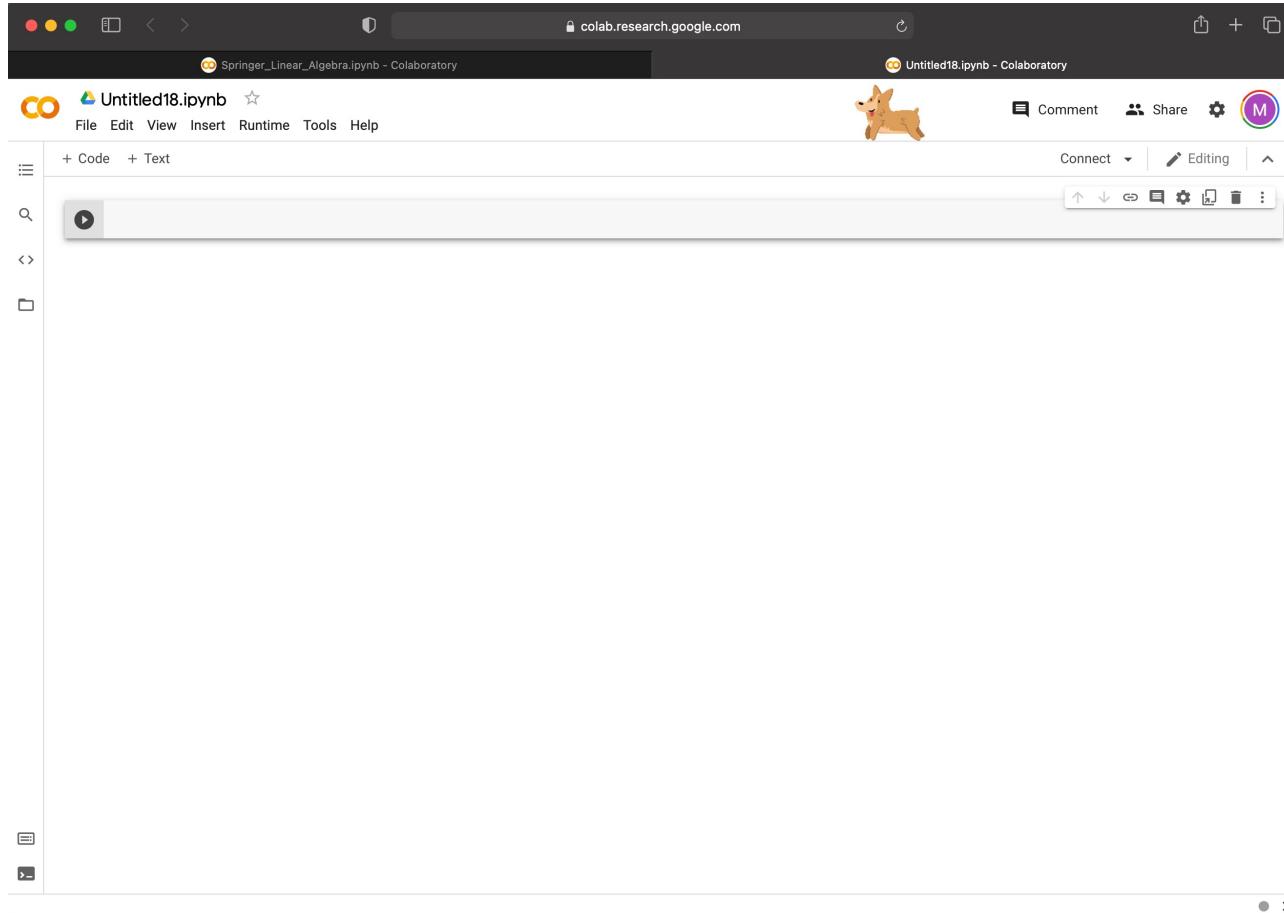
# Google sign-in will be required if you haven't logged in yet.



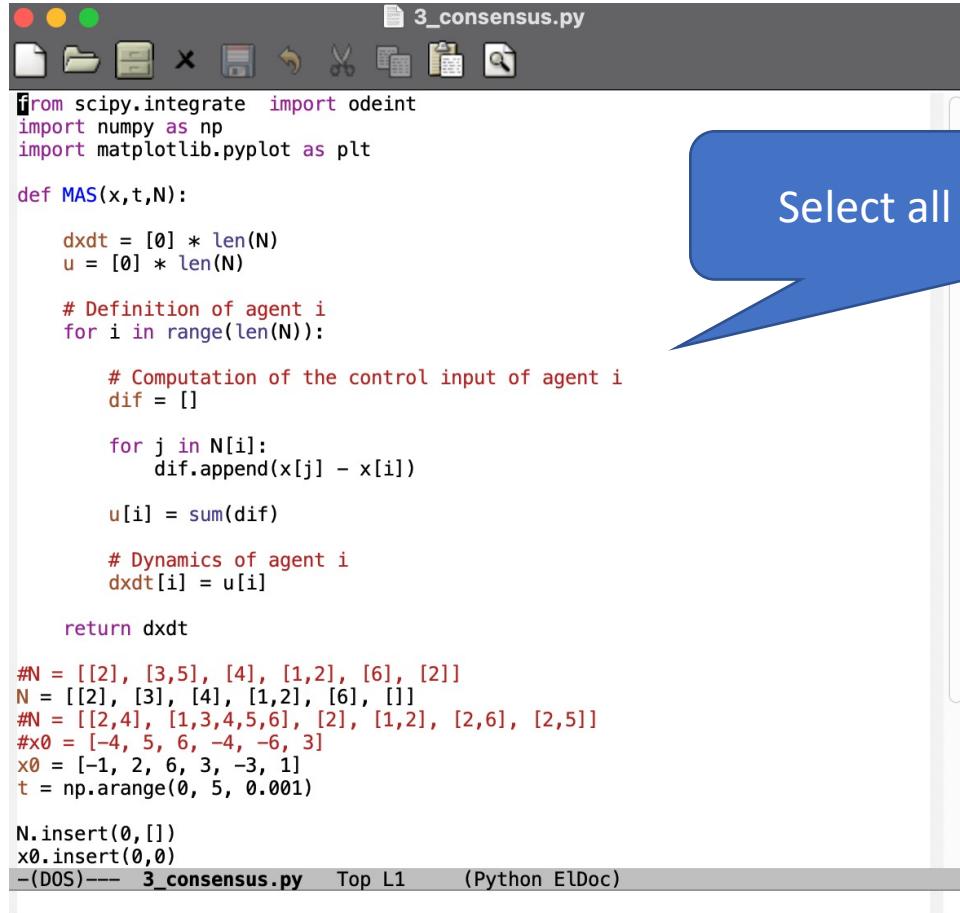
# Sign-in with your Google account



# Then a new notebook is opened



# Open “3\_consensus.py” with your text editor



```
3_consensus.py
from scipy.integrate import odeint
import numpy as np
import matplotlib.pyplot as plt

def MAS(x,t,N):
    dxdt = [0] * len(N)
    u = [0] * len(N)

    # Definition of agent i
    for i in range(len(N)):

        # Computation of the control input of agent i
        dif = []

        for j in N[i]:
            dif.append(x[j] - x[i])

        u[i] = sum(dif)

        # Dynamics of agent i
        dxdt[i] = u[i]

    return dxdt

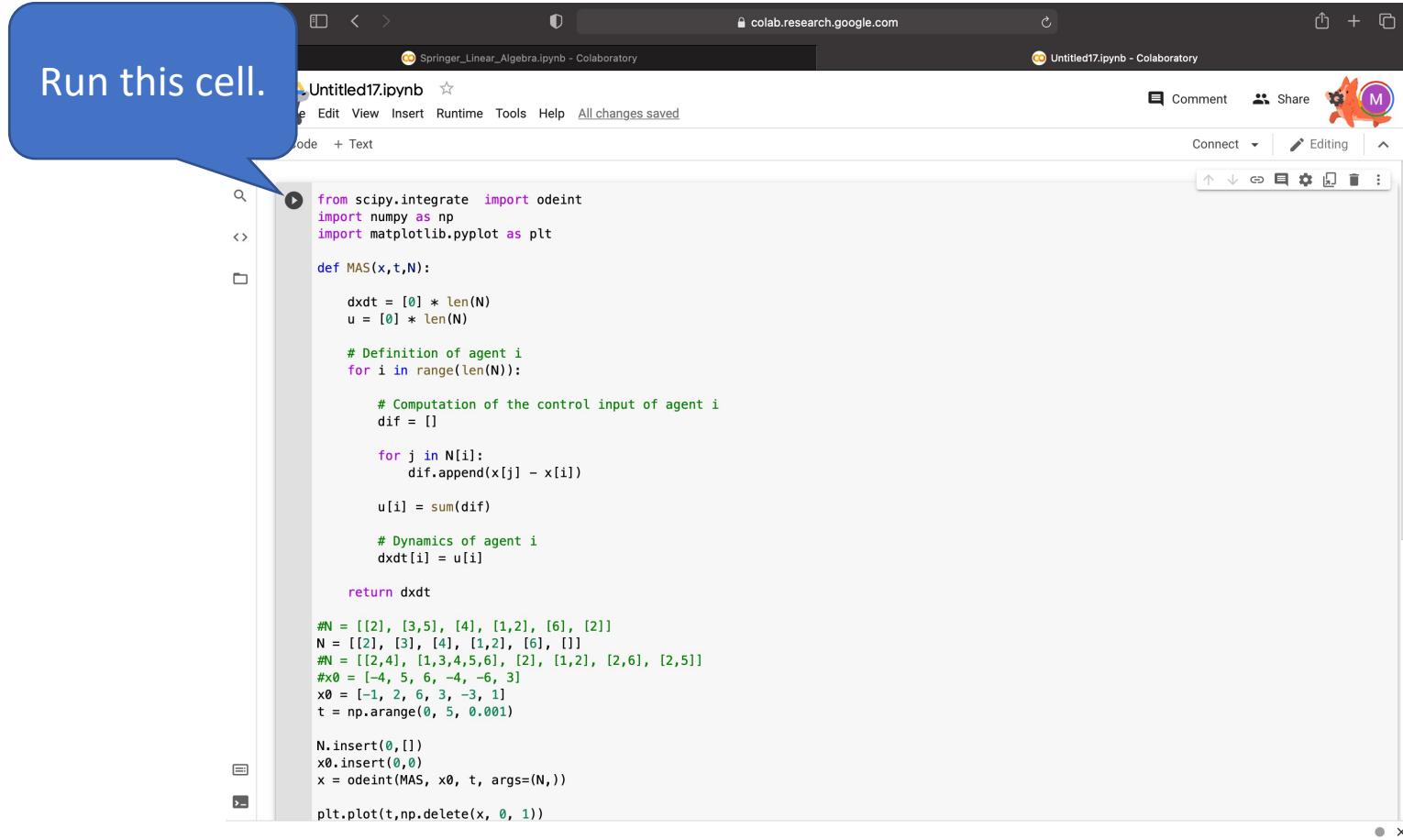
#N = [[2], [3,5], [4], [1,2], [6], [2]]
N = [[2], [3], [4], [1,2], [6], []]
#N = [[2,4], [1,3,4,5,6], [2], [1,2], [2,6], [2,5]]
#x0 = [-4, 5, 6, -4, -6, 3]
x0 = [-1, 2, 6, 3, -3, 1]
t = np.arange(0, 5, 0.001)

N.insert(0,[])
x0.insert(0,0)
-(DOS)--- 3_consensus.py  Top L1  (Python ElDoc)
```

Select all and copy them.

# Past them into the blank cell

Run this cell.



The screenshot shows a Google Colab interface with two code cells. The first cell contains the following Python code:

```
from scipy.integrate import odeint
import numpy as np
import matplotlib.pyplot as plt

def MAS(x,t,N):
    dxdt = [0] * len(N)
    u = [0] * len(N)

    # Definition of agent i
    for i in range(len(N)):

        # Computation of the control input of agent i
        dif = []

        for j in N[i]:
            dif.append(x[j] - x[i])

        u[i] = sum(dif)

        # Dynamics of agent i
        dxdt[i] = u[i]

    return dxdt

#N = [[2], [3,5], [4], [1,2], [6], [2]]
N = [[2], [3], [4], [1,2], [6], []]
#N = [[2,4], [1,3,4,5,6], [2], [1,2], [2,6], [2,5]]
#x0 = [-4, 5, 6, -4, -6, 3]
x0 = [-1, 2, 6, 3, -3, 1]
t = np.arange(0, 5, 0.001)

N.insert(0,[])
x0.insert(0,0)
x = odeint(MAS, x0, t, args=(N,))

plt.plot(t,np.delete(x, 0, 1))
```

# You will get the result!

The screenshot shows a Google Colab interface with two tabs open: "Springer\_Linear\_Algebra.ipynb - Colaboratory" and "Untitled17.ipynb - Colaboratory". The "Untitled17.ipynb" tab is active, displaying a Jupyter notebook cell. The cell contains the following Python code:

```
x0.insert(0,0)
x = odeint(MAS, x0, t, args=(N,))

plt.plot(t,np.delete(x, 0, 1))
plt.xlabel('t')
plt.ylabel('xi')
plt.grid()
plt.show()

plt.savefig("consensus.eps", dpi=200, bbox_inches="tight", pad_inches=0.1)
```

Below the code, a plot is displayed showing the evolution of variables over time  $t$  from 0 to 5. The x-axis is labeled "t" and the y-axis is labeled "xi". The plot shows several curves starting at different initial values and converging towards a steady state. The legend indicates the curves correspond to  $\xi_1, \xi_2, \dots, \xi_N$ .

<Figure size 432x288 with 0 Axes>

At the bottom of the screen, a status bar shows "1s completed at 10:28 AM".