5-day Hands-on Workshop on:

Python for Scientific Computing
and
TensorFlow for Artificial Intelligence

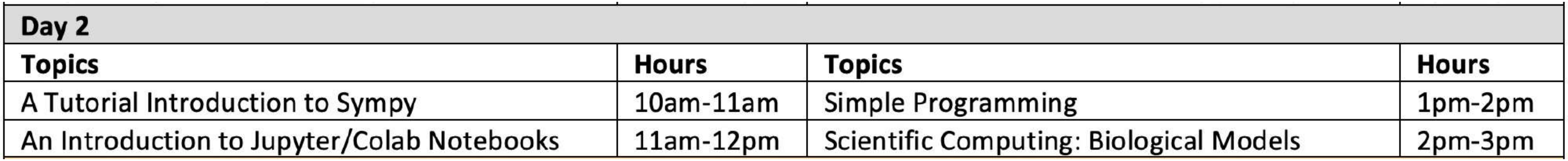
By Dr Stephen Lynch FIMA SFHEA

Holder of Two Patents
Author of PYTHON, MATLAB®, MAPLE™ AND MATHEMATICA® BOOKS
STEM Ambassador and Speaker for Schools

s.lynch@mmu.ac.uk

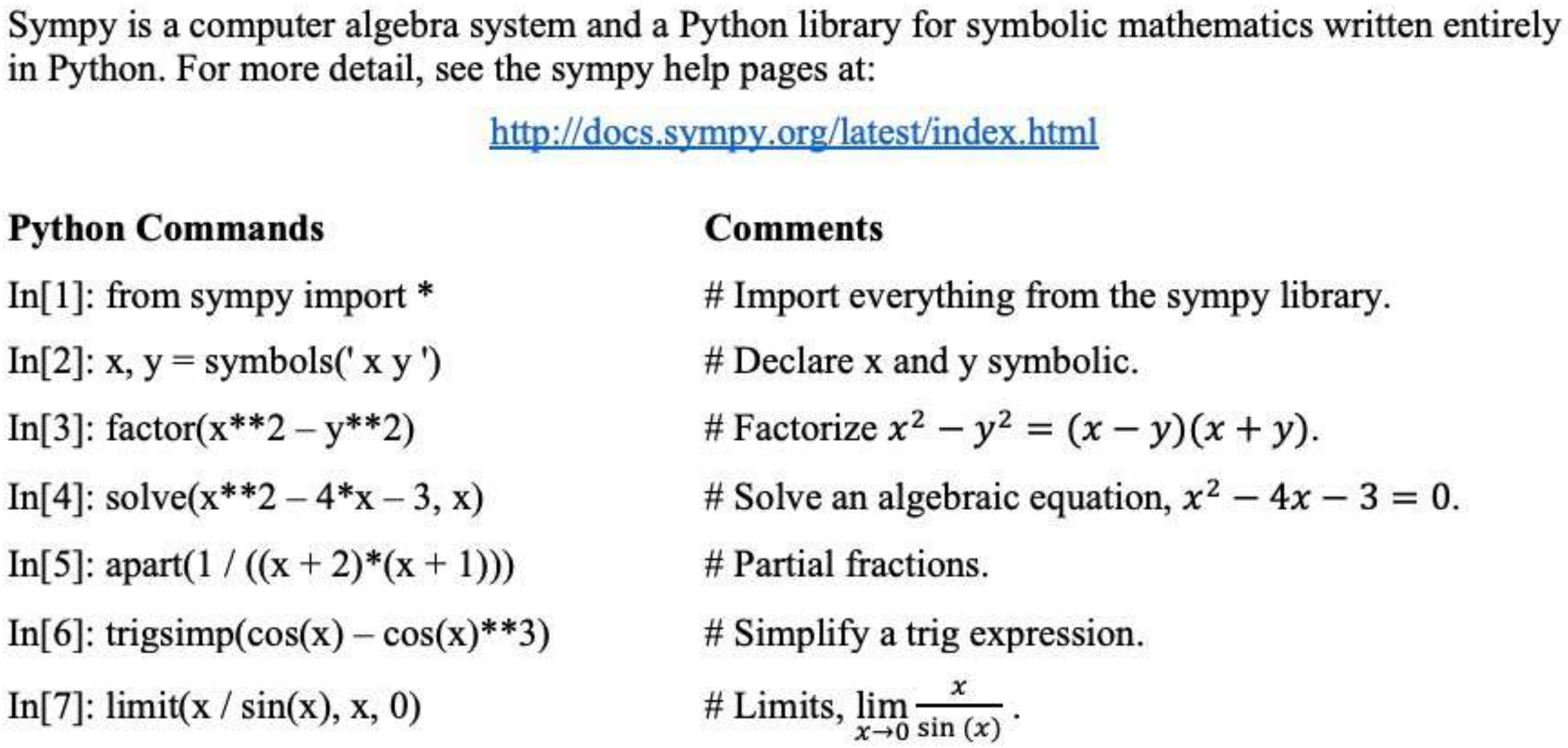
https://www2.mmu.ac.uk/scmdt/staff/profile/index.php?id=2443

Schedule (Day 2): Start Session 1



Day 2			
Topics	**Hours**	**Topics**	**Hours**
A Tutorial Introduction to Sympy	10am-11am	Simple Programming	1pm-2pm
An Introduction to Jupyter/Colab Notebooks	11am-12pm	Scientific Computing: Biological Models	2pm-3pm

A Tutorial Introduction to Sympy (SYMbolic PYthon)



Sympy is a computer algebra system and a Python library for symbolic mathematics written entirely in Python. For more detail, see the sympy help pages at:

http://docs.sympy.org/latest/index.html

Python Commands	**Comments**
In[1]: from sympy import *	# Import everything from the sympy library.
In[2]: x, y = symbols(' x y ')	# Declare x and y symbolic.
In[3]: factor(x**2 – y**2)	# Factorize $x^2 - y^2 = (x - y)(x + y)$.
In[4]: solve(x**2 – 4*x – 3, x)	# Solve an algebraic equation, $x^2 - 4x - 3 = 0$.
In[5]: apart(1 / ((x + 2)*(x + 1)))	# Partial fractions.
In[6]: trigsimp(cos(x) – cos(x)**3)	# Simplify a trig expression.
In[7]: limit(x / sin(x), x, 0)	# Limits, $\lim_{x \to 0} \frac{x}{\sin(x)}$.

A Tutorial Introduction to Sympy (SYMbolic PYthon)

`In[8]: diff(x**2 - 7*x + 8, x)` # Differentiate with respect to x.

`In[9]: diff(5*x**7, x, 3)` # Differentiate with respect to x three times.

`In[10]: (exp(x)*cos(x)).series(x, 0, 10)` # Taylor series expansion around $x = 0$.

`In[11]: integrate(x**2 - 7*x + 8, x)` # Indefinite integration, $\int x^2 - 7x + 8\,dx$

`In[12]: integrate(x**2 - 7*x + 8, (x, 1, 2))` # Definite integration, $\int_{x=1}^{2} x^2 - 7x + 8\,dx$.

`In[13]: summation(1 / x**2, (x, 1, oo))` # Infinite summation, $\sum_{x=1}^{\infty} \frac{1}{x^2}$.

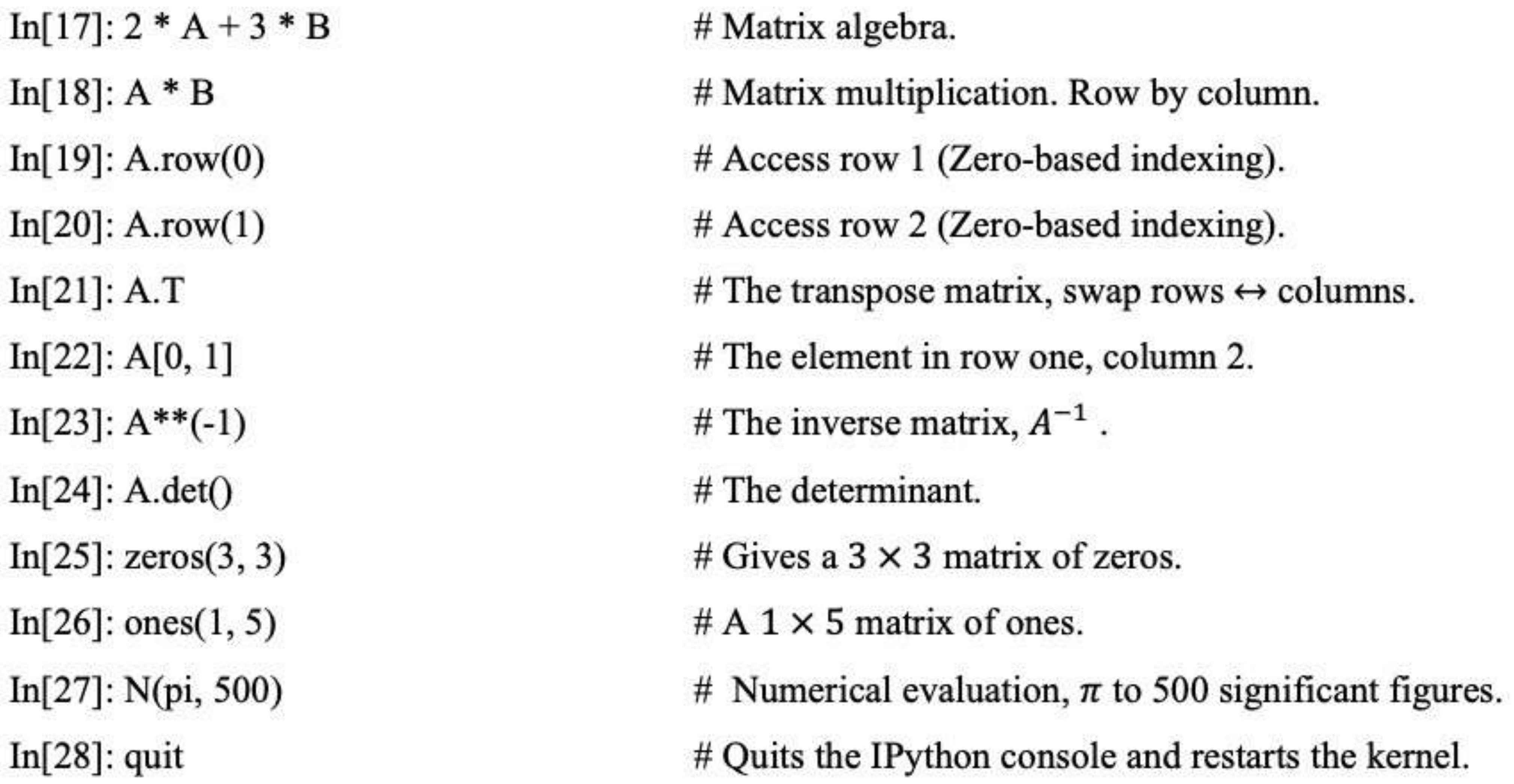
`In[14]: solve([x+5*y-2, -3*x+6*y-15],[x,y])` # Solve simultaneous equations.

`In[15]: A = Matrix([[1, -1], [2, 3]])` # The matrix, $A = \begin{pmatrix} 1 & -1 \\ 2 & 3 \end{pmatrix}$.

`In[16]: B = Matrix([[0, 2], [3, 3]])` # The matrix, $B = \begin{pmatrix} 0 & 2 \\ 3 & 3 \end{pmatrix}$.

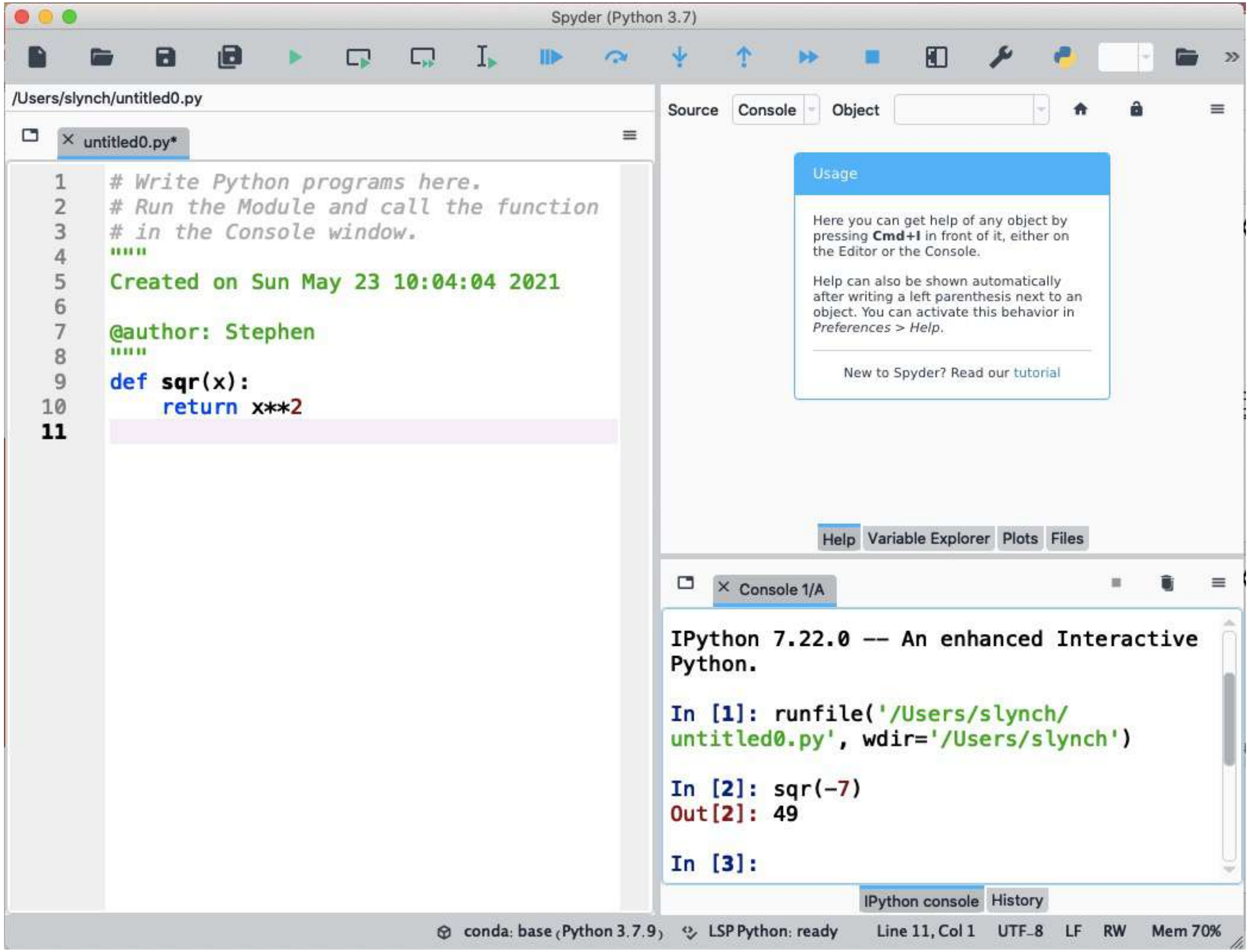
Manchester Metropolitan University

A Tutorial Introduction to Sympy (SYMbolic PYthon)



`In[17]: 2 * A + 3 * B`	# Matrix algebra.
`In[18]: A * B`	# Matrix multiplication. Row by column.
`In[19]: A.row(0)`	# Access row 1 (Zero-based indexing).
`In[20]: A.row(1)`	# Access row 2 (Zero-based indexing).
`In[21]: A.T`	# The transpose matrix, swap rows ↔ columns.
`In[22]: A[0, 1]`	# The element in row one, column 2.
`In[23]: A**(-1)`	# The inverse matrix, A^{-1} .
`In[24]: A.det()`	# The determinant.
`In[25]: zeros(3, 3)`	# Gives a 3 × 3 matrix of zeros.
`In[26]: ones(1, 5)`	# A 1 × 5 matrix of ones.
`In[27]: N(pi, 500)`	# Numerical evaluation, π to 500 significant figures.
`In[28]: quit`	# Quits the IPython console and restarts the kernel.

A Simple Program in Spyder



```
# Write Python programs here.
# Run the Module and call the function
# in the Console window.
"""
Created on Sun May 23 10:04:04 2021

@author: Stephen
"""
def sqr(x):
    return x**2

```

Usage

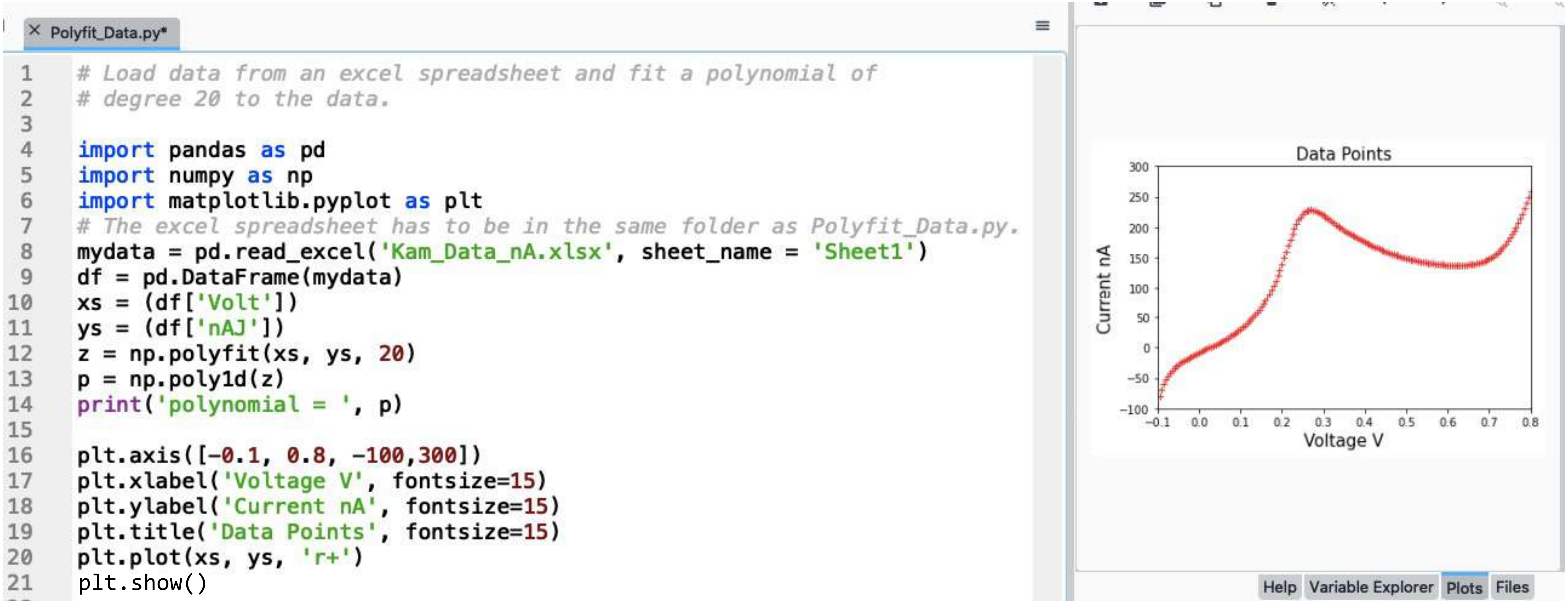
Here you can get help of any object by pressing **Cmd+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
IPython 7.22.0 -- An enhanced Interactive
Python.

In [1]: runfile('/Users/slynch/
untitled0.py', wdir='/Users/slynch')

In [2]: sqr(-7)
Out[2]: 49

In [3]:
```

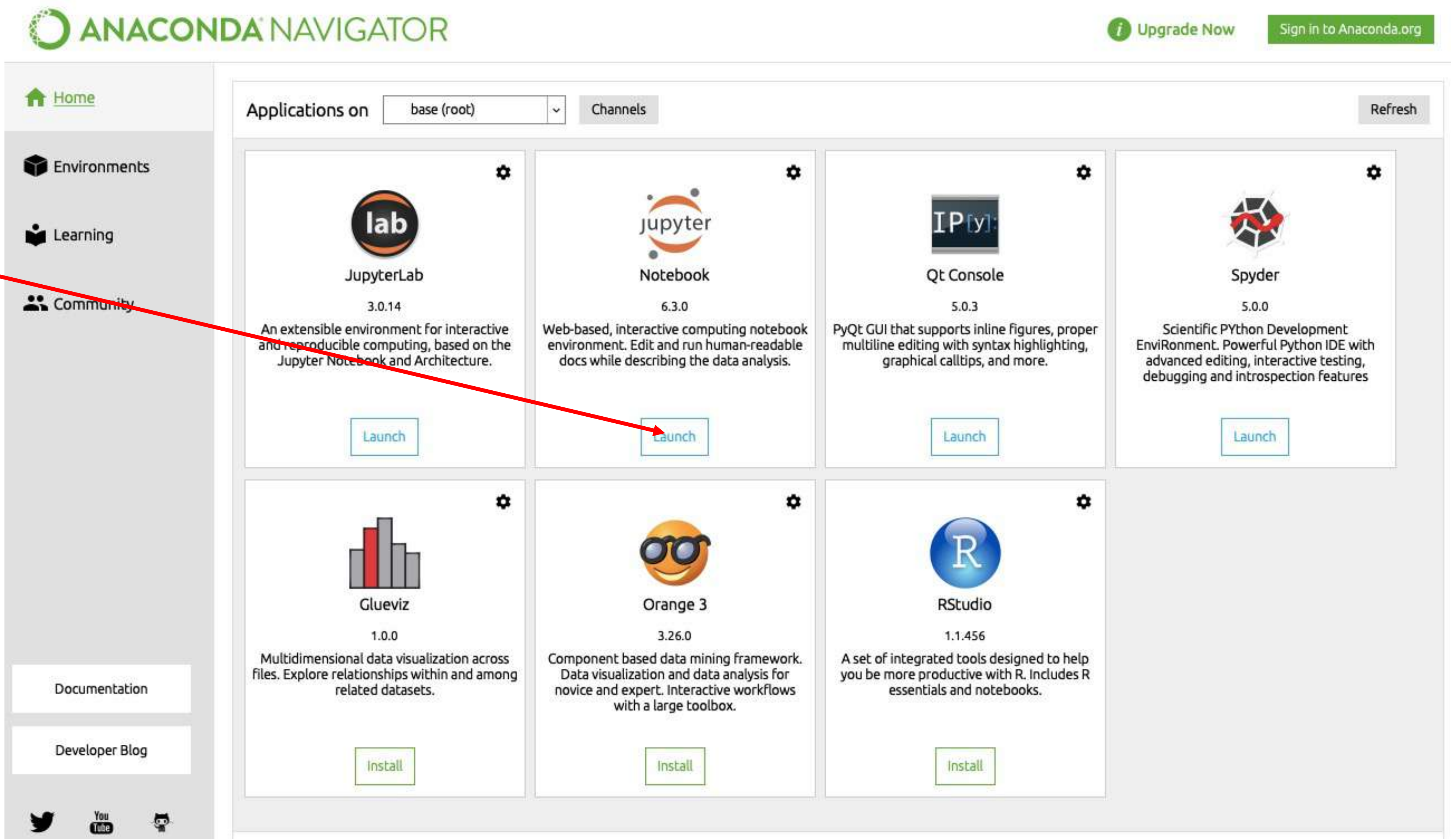
Load Data from an Excel Spreadsheet using Pandas: End Session 1



```python
# Load data from an excel spreadsheet and fit a polynomial of
# degree 20 to the data.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# The excel spreadsheet has to be in the same folder as Polyfit_Data.py.
mydata = pd.read_excel('Kam_Data_nA.xlsx', sheet_name = 'Sheet1')
df = pd.DataFrame(mydata)
xs = (df['Volt'])
ys = (df['nAJ'])
z = np.polyfit(xs, ys, 20)
p = np.poly1d(z)
print('polynomial = ', p)

plt.axis([-0.1, 0.8, -100,300])
plt.xlabel('Voltage V', fontsize=15)
plt.ylabel('Current nA', fontsize=15)
plt.title('Data Points', fontsize=15)
plt.plot(xs, ys, 'r+')
plt.show()
```

Anaconda: Launch a Jupyter Notebook: Start Session 2

Click here to open a new Python 3 ipynb notebook



JupyterLab
3.0.14
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
6.3.0
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Qt Console
5.0.3
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

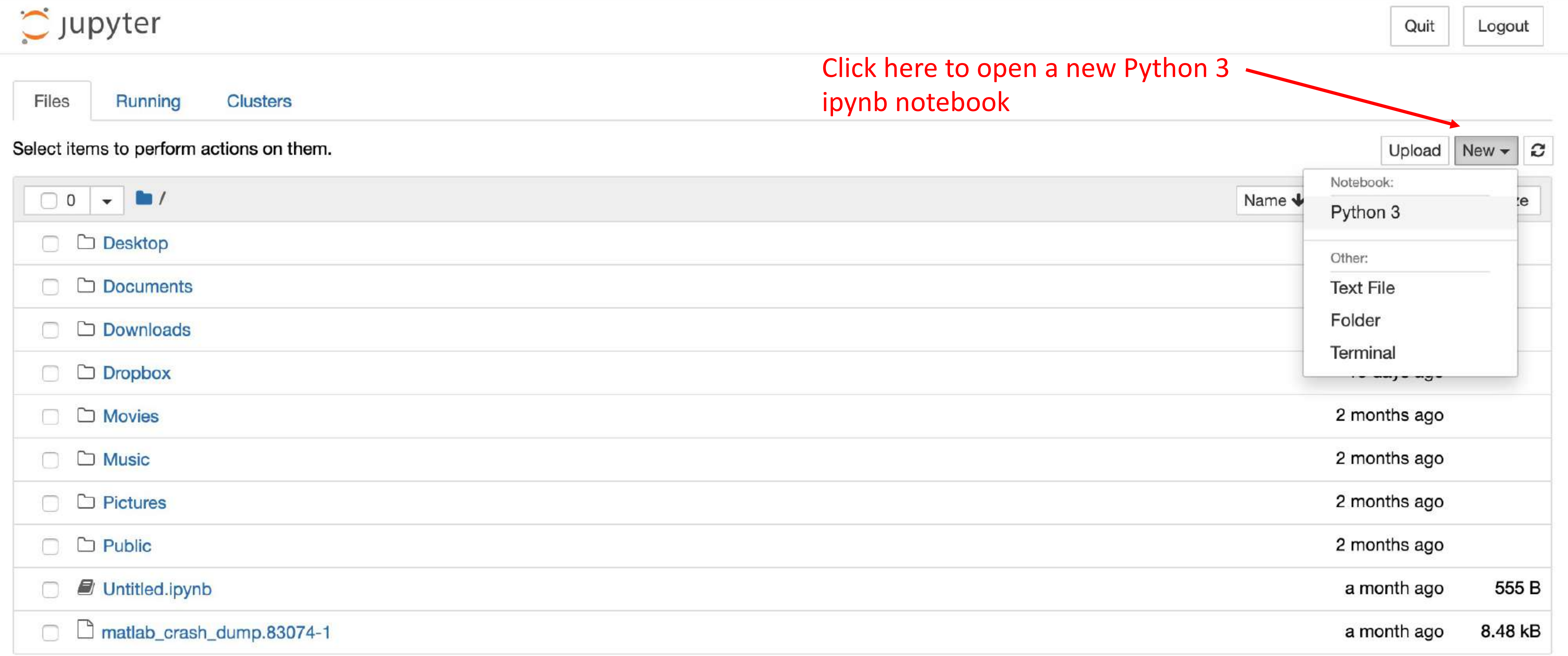
Spyder
5.0.0
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Glueviz
1.0.0
Multidimensional data visualization across files. Explore relationships within and among related datasets.

Orange 3
3.26.0
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

RStudio
1.1.456
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

Anaconda: Launch a Jupyter Notebook

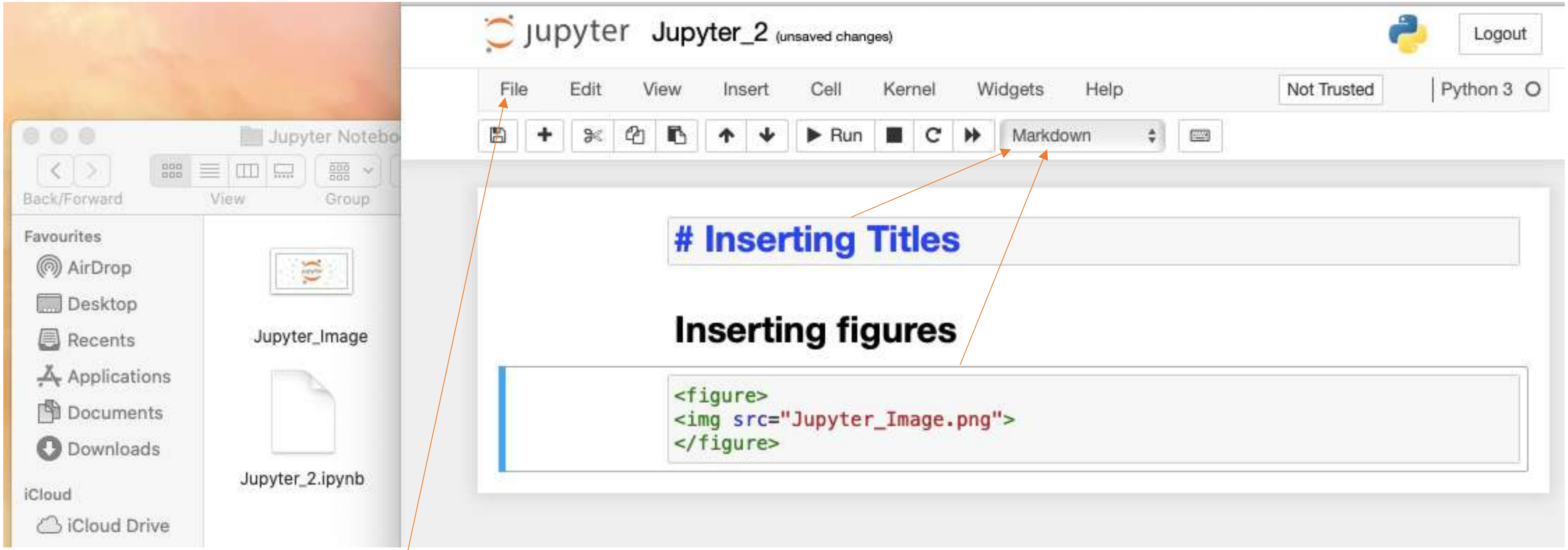


Click here to open a new Python 3 ipynb notebook

Select items to perform actions on them.

Desktop		
Documents		
Downloads		
Dropbox		
Movies	2 months ago	
Music	2 months ago	
Pictures	2 months ago	
Public	2 months ago	
Untitled.ipynb	a month ago	555 B
matlab_crash_dump.83074-1	a month ago	8.48 kB

Inserting Titles and Figures



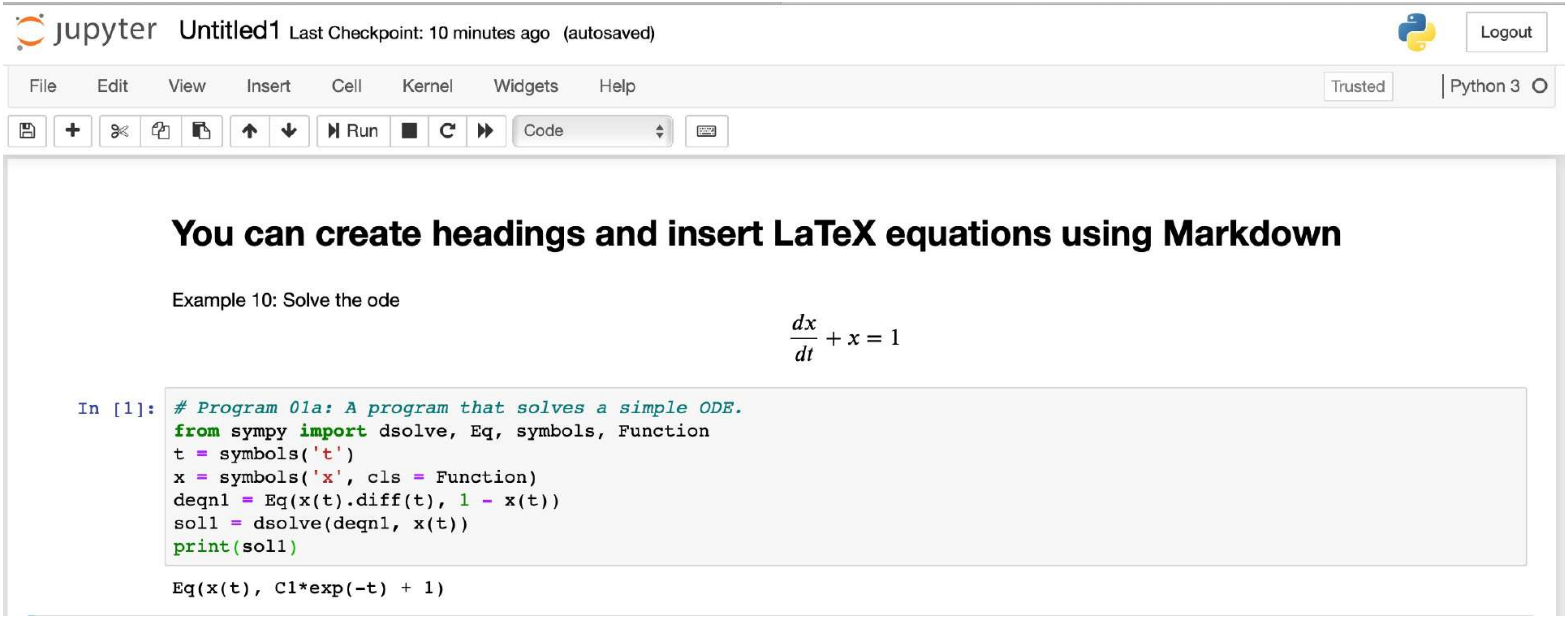
Inserting Titles

Inserting figures

```
<figure>
<img src="Jupyter_Image.png">
</figure>
```

You can **File** > **Download as** > Notebook (.pynb) or Webpage (.html)

Simple Programming with Jupyter Notebooks (Solving ODEs)



You can create headings and insert LaTeX equations using Markdown

Example 10: Solve the ode

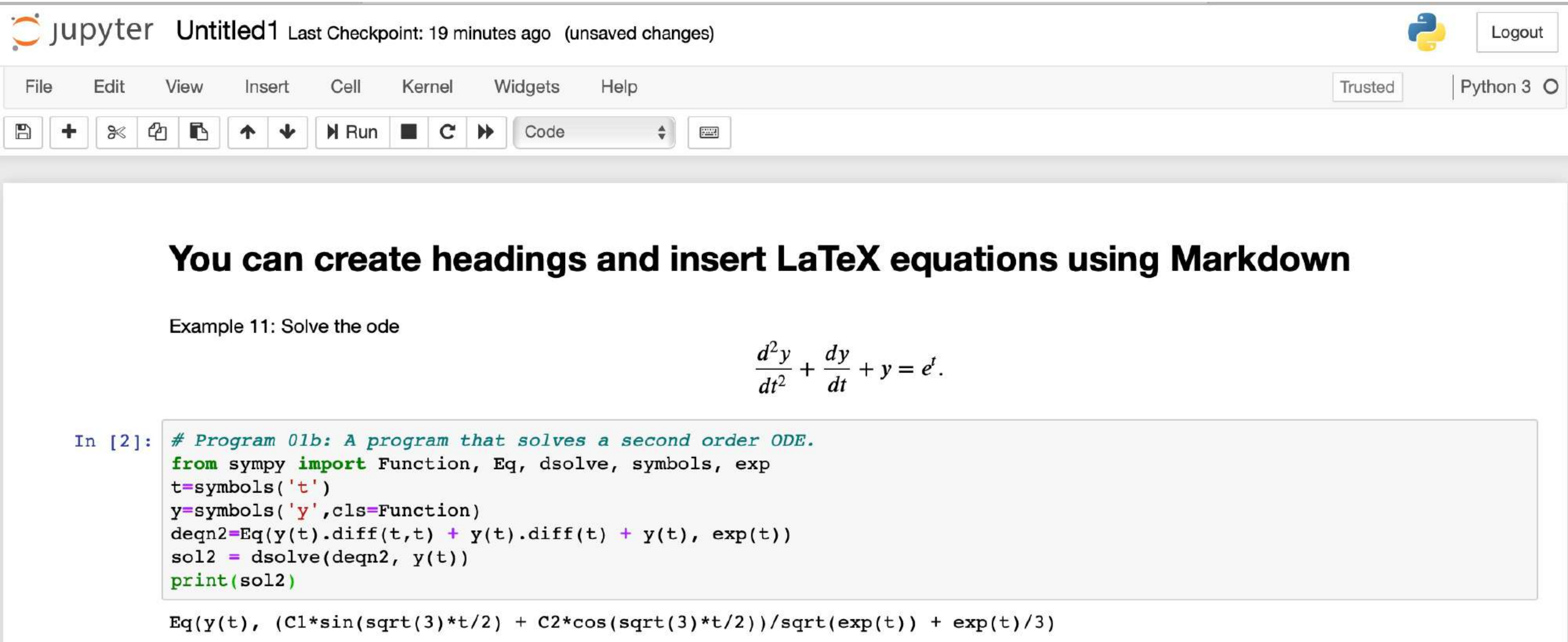
$$\frac{dx}{dt} + x = 1$$

```
# Program 01a: A program that solves a simple ODE.
from sympy import dsolve, Eq, symbols, Function
t = symbols('t')
x = symbols('x', cls = Function)
deqn1 = Eq(x(t).diff(t), 1 - x(t))
sol1 = dsolve(deqn1, x(t))
print(sol1)
```

```
Eq(x(t), C1*exp(-t) + 1)
```

https://oeis.org/wiki/List_of_LaTeX_mathematical_symbols

Simple Programming with Jupyter Notebooks



You can create headings and insert LaTeX equations using Markdown

Example 11: Solve the ode

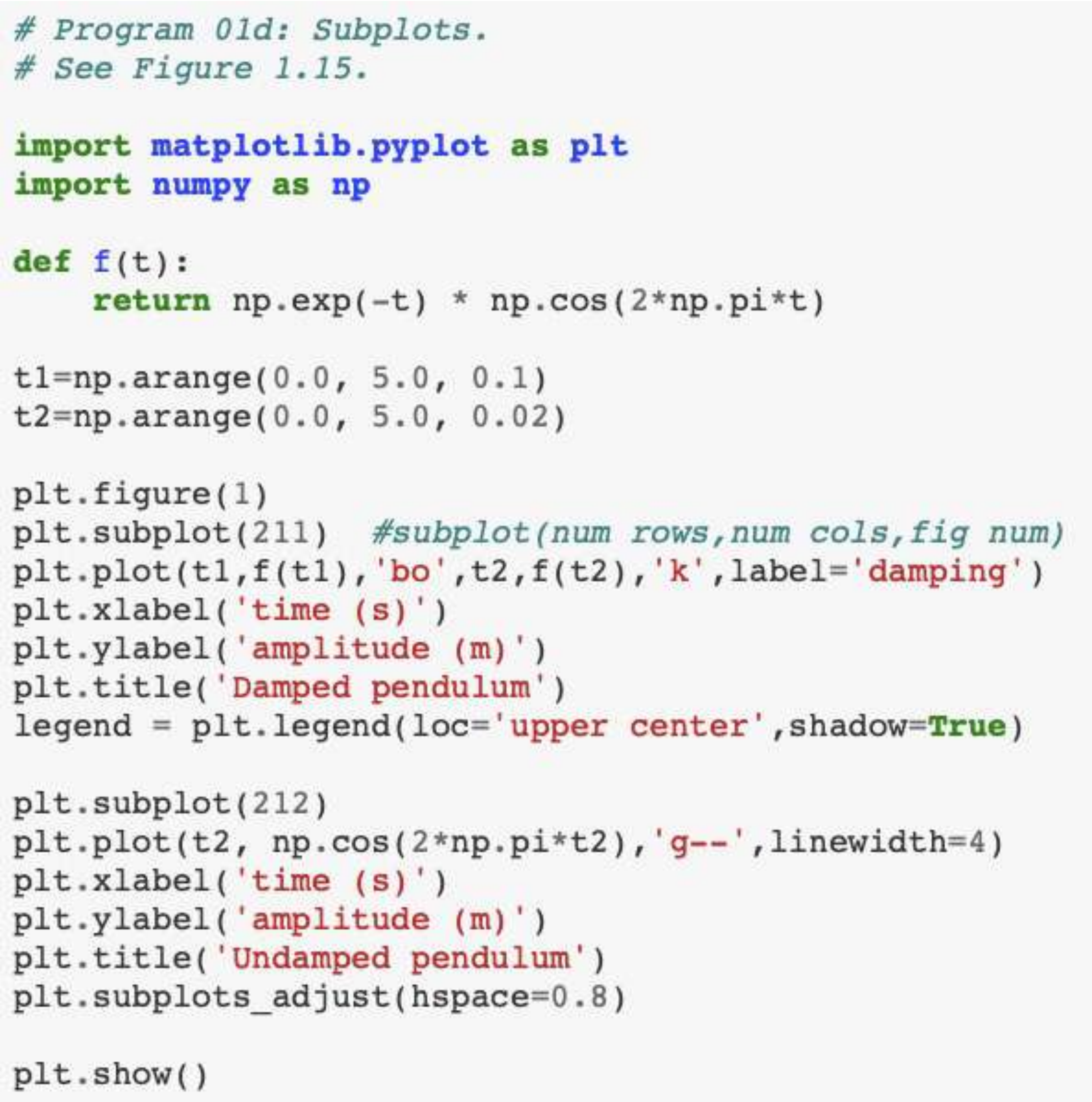
$$\frac{d^2y}{dt^2} + \frac{dy}{dt} + y = e^t.$$

In [2]:

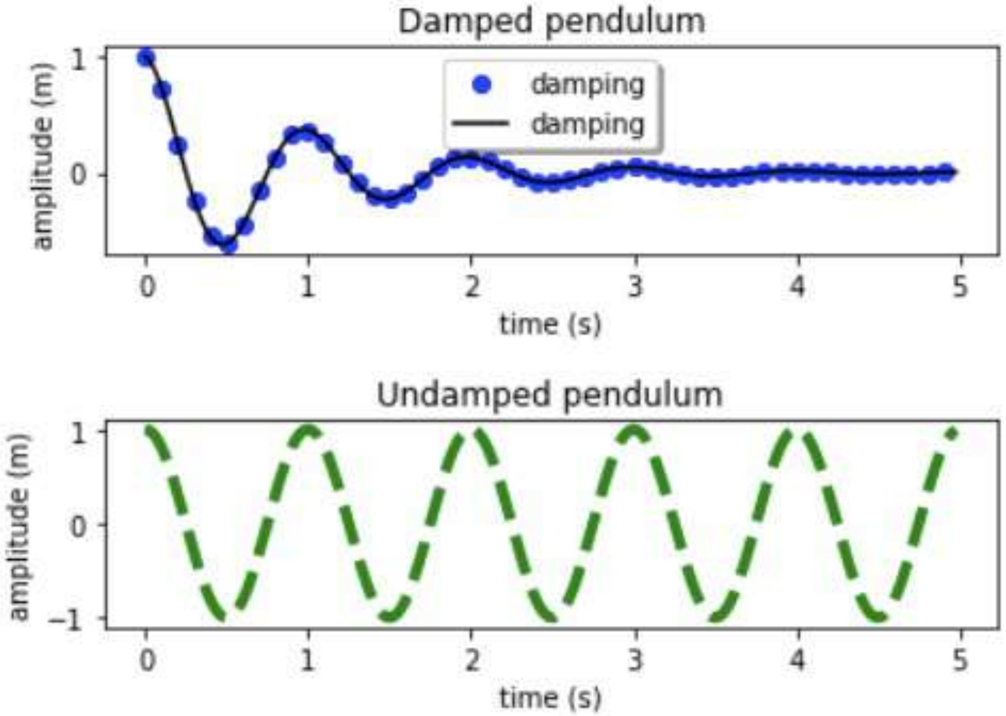
```python
# Program 01b: A program that solves a second order ODE.
from sympy import Function, Eq, dsolve, symbols, exp
t=symbols('t')
y=symbols('y',cls=Function)
deqn2=Eq(y(t).diff(t,t) + y(t).diff(t) + y(t), exp(t))
sol2 = dsolve(deqn2, y(t))
print(sol2)
```

```
Eq(y(t), (C1*sin(sqrt(3)*t/2) + C2*cos(sqrt(3)*t/2))/sqrt(exp(t)) + exp(t)/3)
```

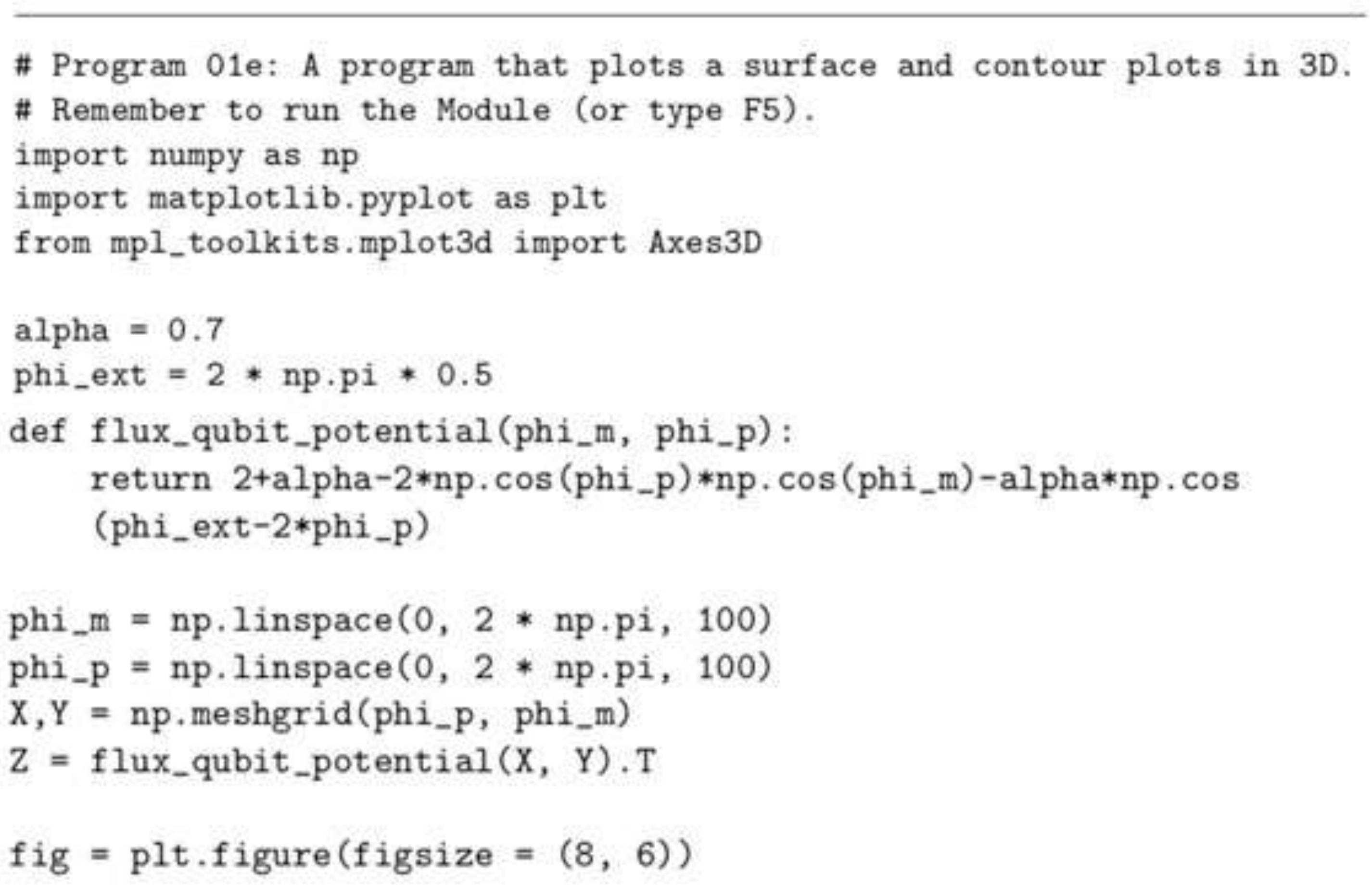
Simple Programming (Subplots)



```python
# Program 01d: Subplots.
# See Figure 1.15.

import matplotlib.pyplot as plt
import numpy as np

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1=np.arange(0.0, 5.0, 0.1)
t2=np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211)  #subplot(num rows,num cols,fig num)
plt.plot(t1,f(t1),'bo',t2,f(t2),'k',label='damping')
plt.xlabel('time (s)')
plt.ylabel('amplitude (m)')
plt.title('Damped pendulum')
legend = plt.legend(loc='upper center',shadow=True)

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2),'g--',linewidth=4)
plt.xlabel('time (s)')
plt.ylabel('amplitude (m)')
plt.title('Undamped pendulum')
plt.subplots_adjust(hspace=0.8)

plt.show()
```

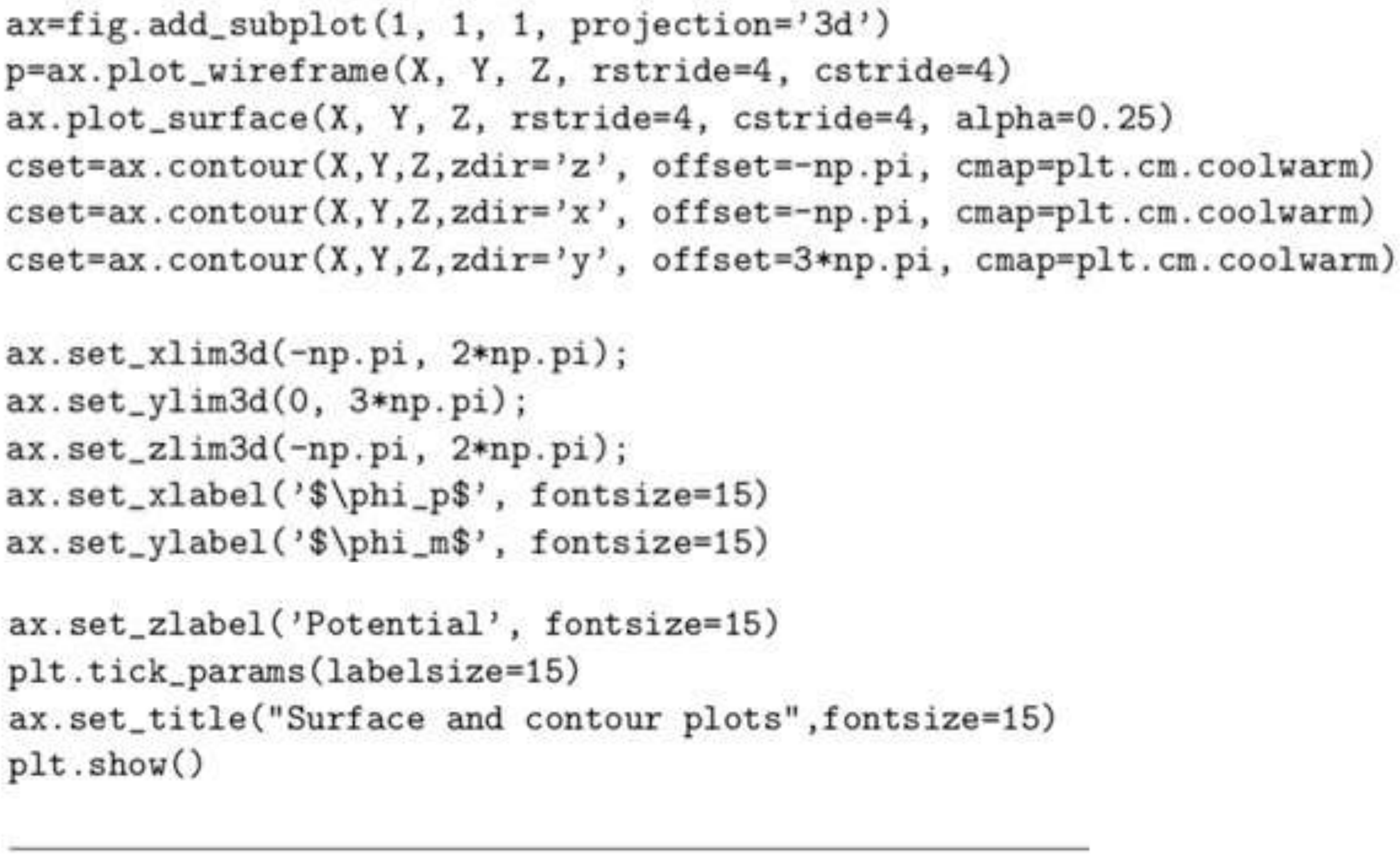


Simple Programming (Surface and Contour Plot)



```
# Program 01e: A program that plots a surface and contour plots in 3D.
# Remember to run the Module (or type F5).
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

alpha = 0.7
phi_ext = 2 * np.pi * 0.5
def flux_qubit_potential(phi_m, phi_p):
    return 2+alpha-2*np.cos(phi_p)*np.cos(phi_m)-alpha*np.cos
    (phi_ext-2*phi_p)

phi_m = np.linspace(0, 2 * np.pi, 100)
phi_p = np.linspace(0, 2 * np.pi, 100)
X,Y = np.meshgrid(phi_p, phi_m)
Z = flux_qubit_potential(X, Y).T

fig = plt.figure(figsize = (8, 6))
```

Simple Programming (Surface and Contour Plot)



```
ax=fig.add_subplot(1, 1, 1, projection=’3d’)
p=ax.plot_wireframe(X, Y, Z, rstride=4, cstride=4)
ax.plot_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.25)
cset=ax.contour(X,Y,Z,zdir=’z’, offset=-np.pi, cmap=plt.cm.coolwarm)
cset=ax.contour(X,Y,Z,zdir=’x’, offset=-np.pi, cmap=plt.cm.coolwarm)
cset=ax.contour(X,Y,Z,zdir=’y’, offset=3*np.pi, cmap=plt.cm.coolwarm)

ax.set_xlim3d(-np.pi, 2*np.pi);
ax.set_ylim3d(0, 3*np.pi);
ax.set_zlim3d(-np.pi, 2*np.pi);
ax.set_xlabel(’$\phi_p$’, fontsize=15)
ax.set_ylabel(’$\phi_m$’, fontsize=15)

ax.set_zlabel(’Potential’, fontsize=15)
plt.tick_params(labelsize=15)
ax.set_title("Surface and contour plots",fontsize=15)
plt.show()
```

Simple Programming (Surface and Contour Plot)

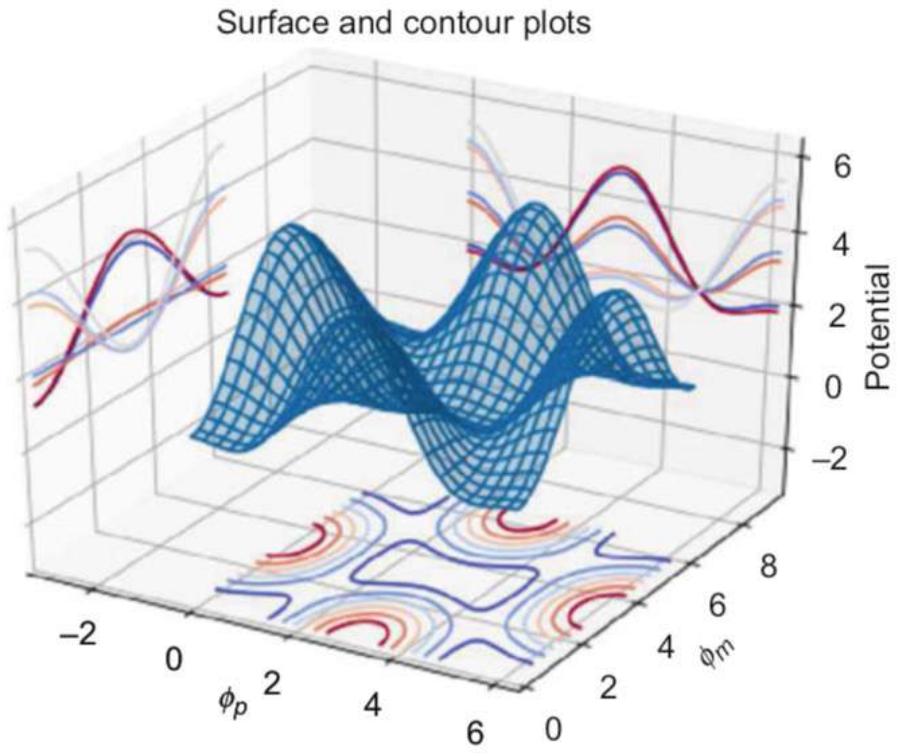


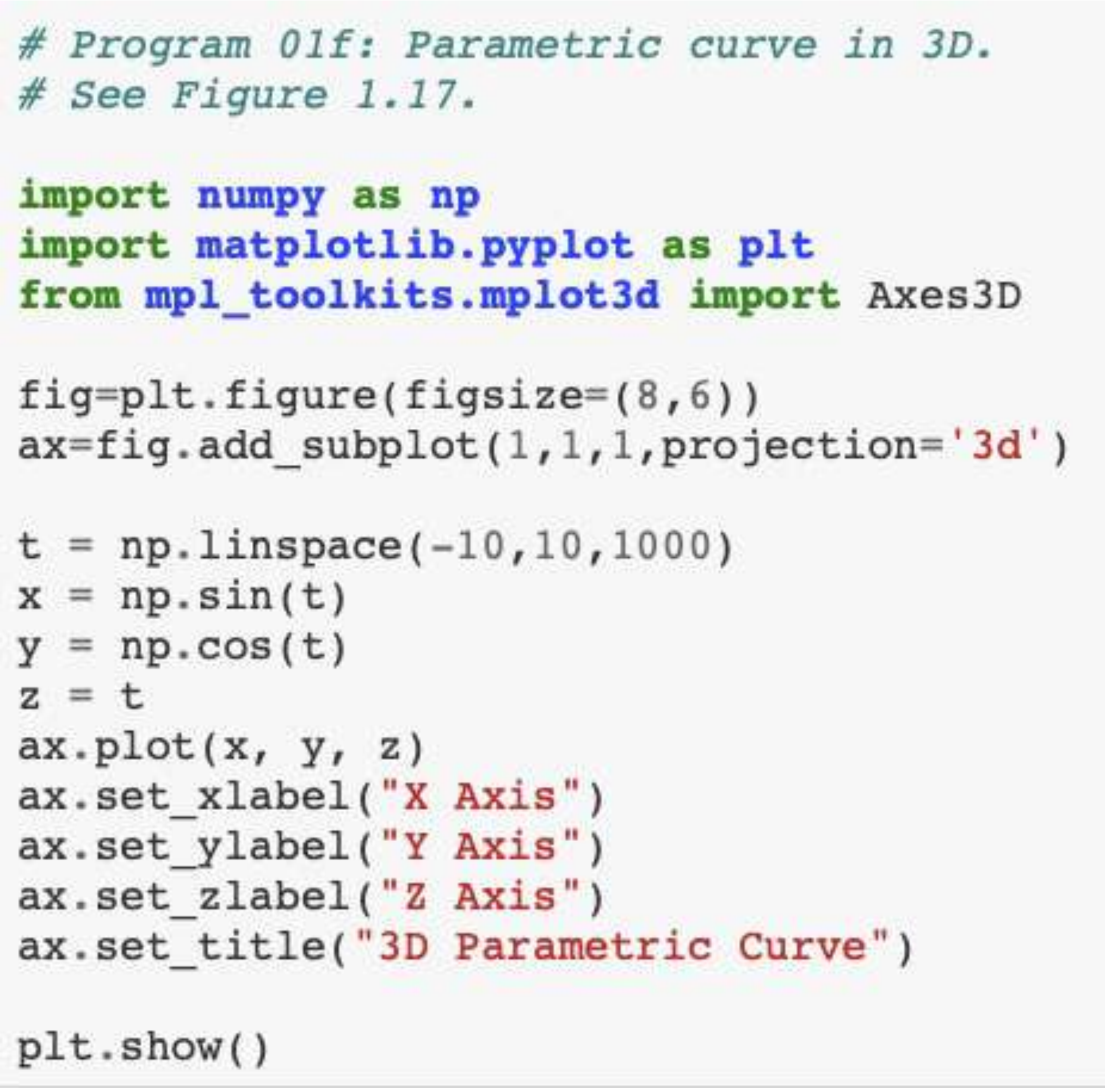
Figure 1.16: [Python] A surface and contour plot. Note that the font size of ticks and axis labels have also been set. In this case the axis labels are generated with LaTeX code.

You can rotate the figure in Spyder.

In the Console window type:

In[1]: %matplotlib qt5

A 3D Parametric Curve: End Session 2



```
# Program 01f: Parametric curve in 3D.
# See Figure 1.17.

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig=plt.figure(figsize=(8,6))
ax=fig.add_subplot(1,1,1,projection='3d')

t = np.linspace(-10,10,1000)
x = np.sin(t)
y = np.cos(t)
z = t
ax.plot(x, y, z)
ax.set_xlabel("X Axis")
ax.set_ylabel("Y Axis")
ax.set_zlabel("Z Axis")
ax.set_title("3D Parametric Curve")

plt.show()
```

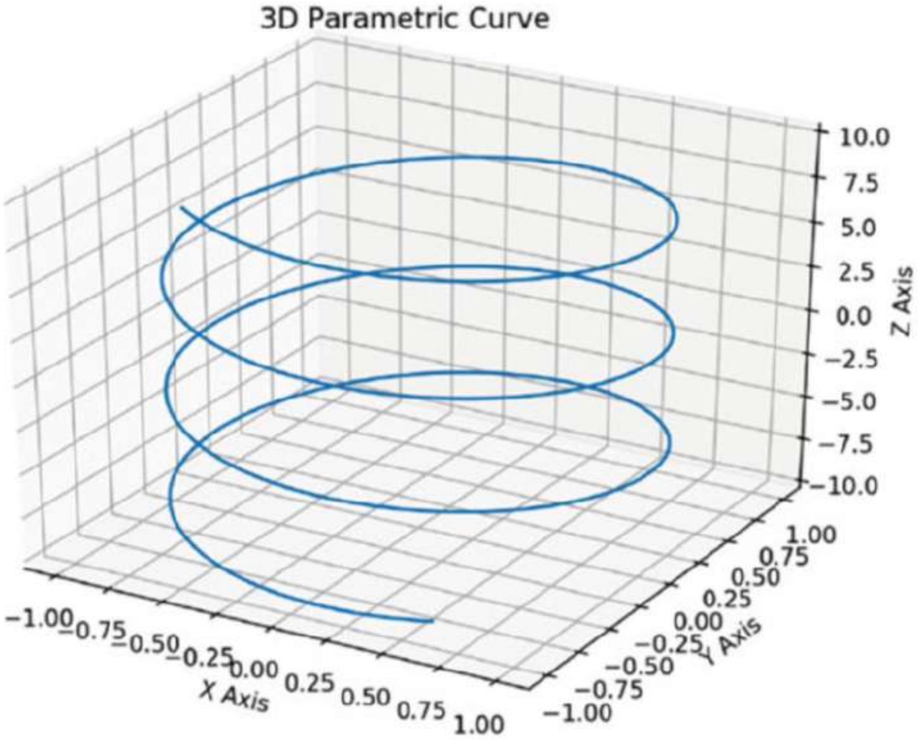
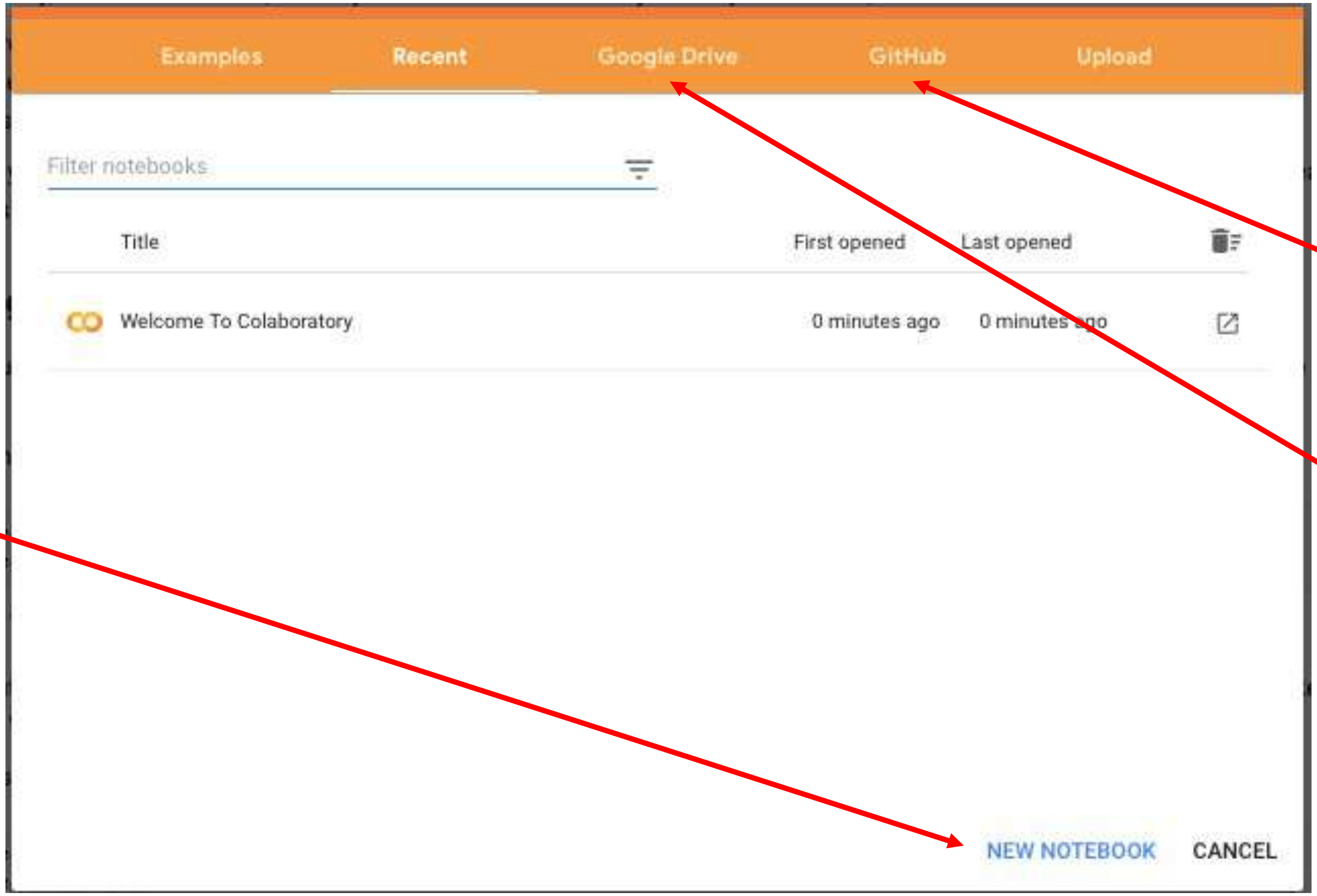


Figure 1.17: [Python] A parametric plot in 3D.

Google Colab: Start Session 3



Title	First opened	Last opened
Welcome To Colaboratory	0 minutes ago	0 minutes ago

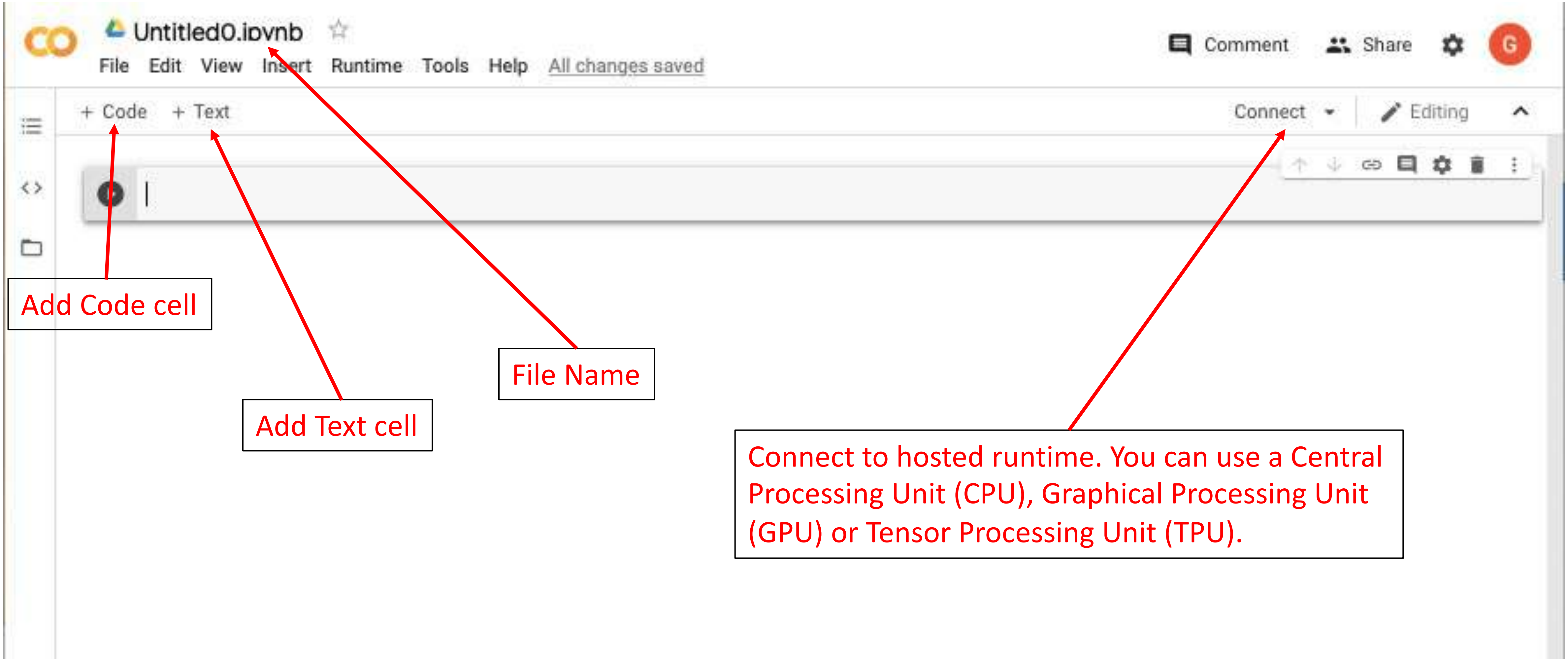
GitHub

Google Drive

Click here to open a new Python 3 ipynb notebook

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

Google Colab: Untitled0.ipynb Notebook



Add Code cell

Add Text cell

File Name

Connect to hosted runtime. You can use a Central Processing Unit (CPU), Graphical Processing Unit (GPU) or Tensor Processing Unit (TPU).

Google Colab: Untitled0.ipynb Notebook

Save the file to Google Drive or straight to GitHub

```
[1] def f_mu(mu, x):
      return mu * x * (1 - x)
```

You can insert Math text here using LaTeX: $f_\mu(x) = \mu x(1 - x)$.

```
f_mu(2, 0.8)
```

```
0.31999999999999995
```

Connect to hosted runtime.
You can use CPU, GPU or TPU

Google Colab: Loading Images from the Web and Computer

Loading a figure from the Web:

```
![picture](http://algotrading101.com/learn/wp-content/uploads/2021/05/Google-Colab-Guide-1024x683.jpg)
```

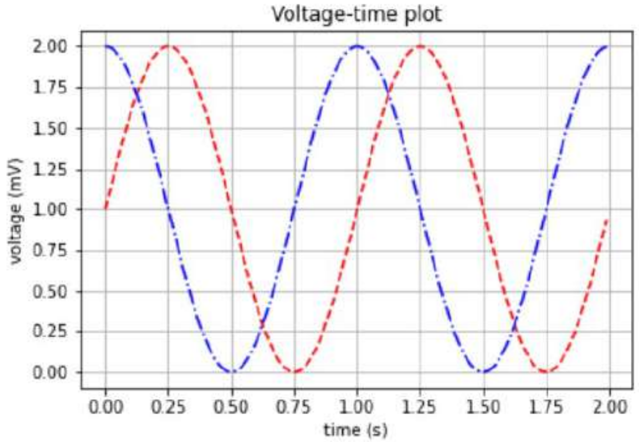
Copy Image Address from internet

Loading a figure from your computer:

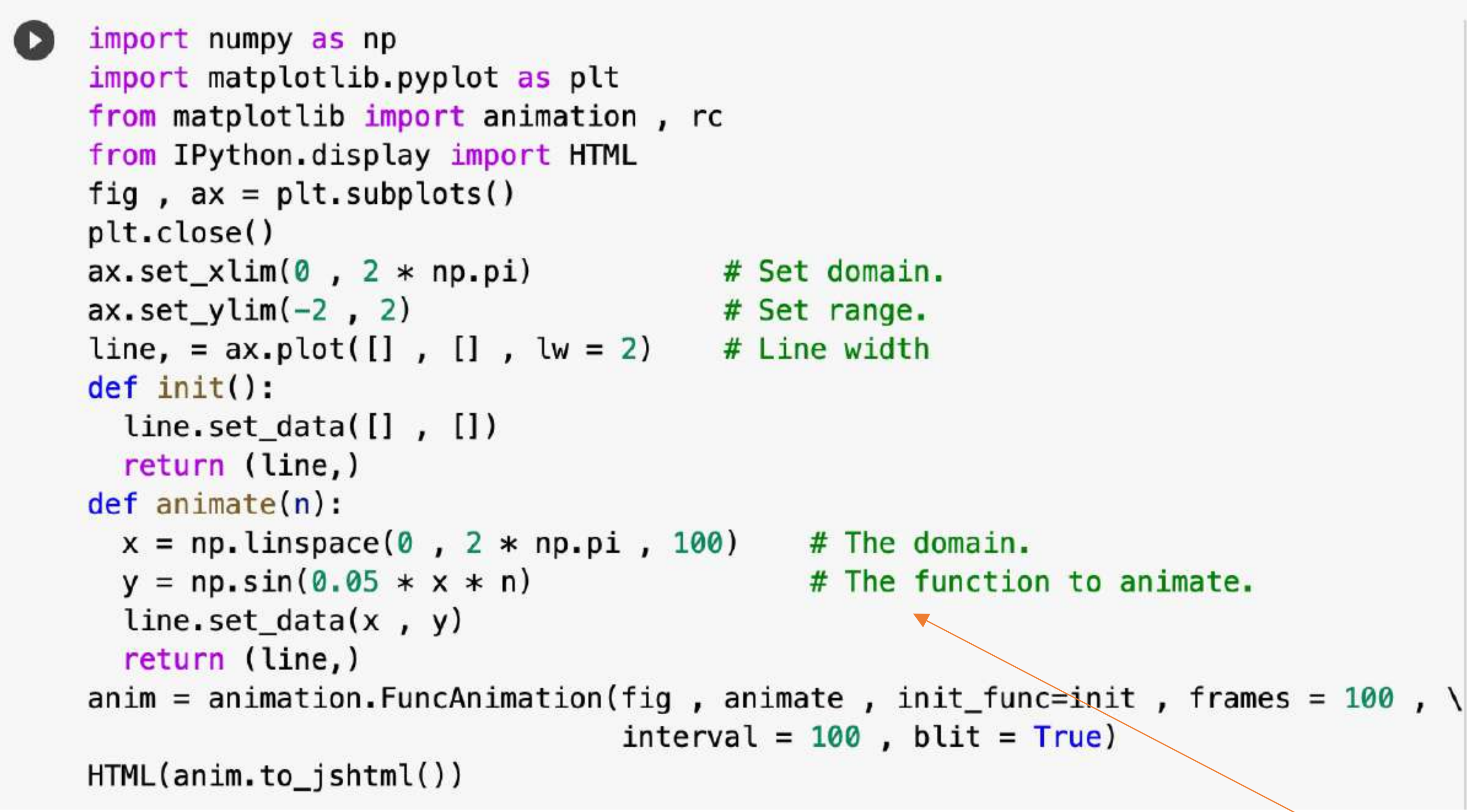
```
[1] from google.colab import files
    from IPython.display import Image
```

```
uploaded = files.upload()
```

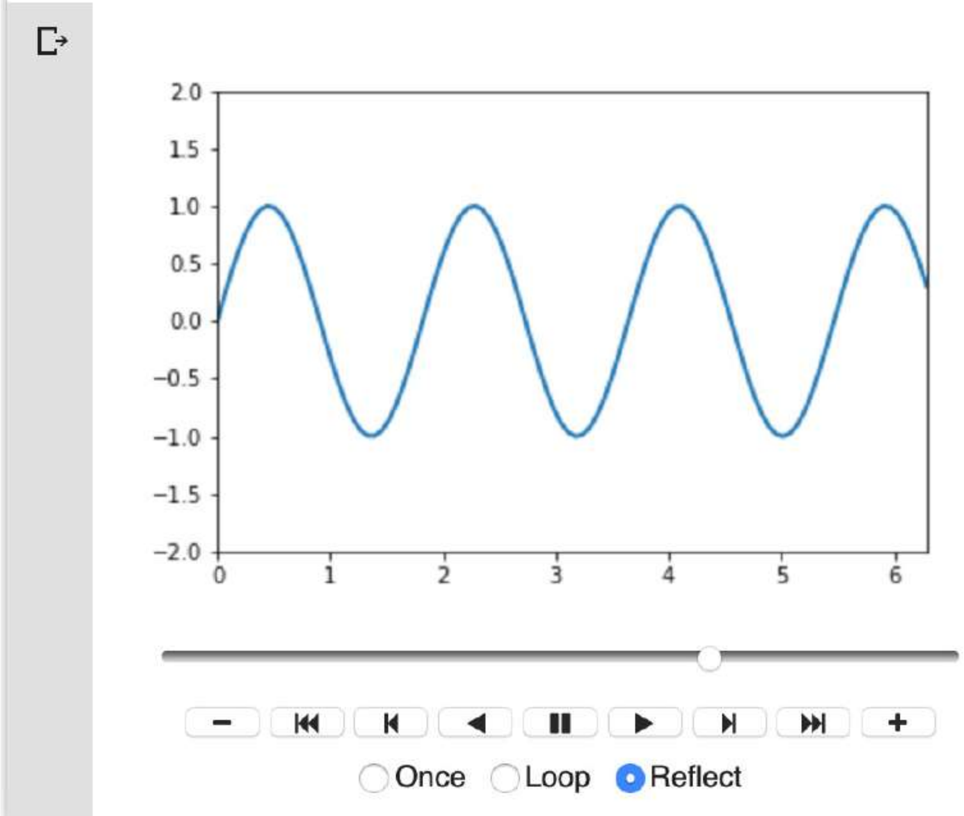
```
Image('Voltage-Time Plot.png', width = 300)
```



Animation in Google Colab

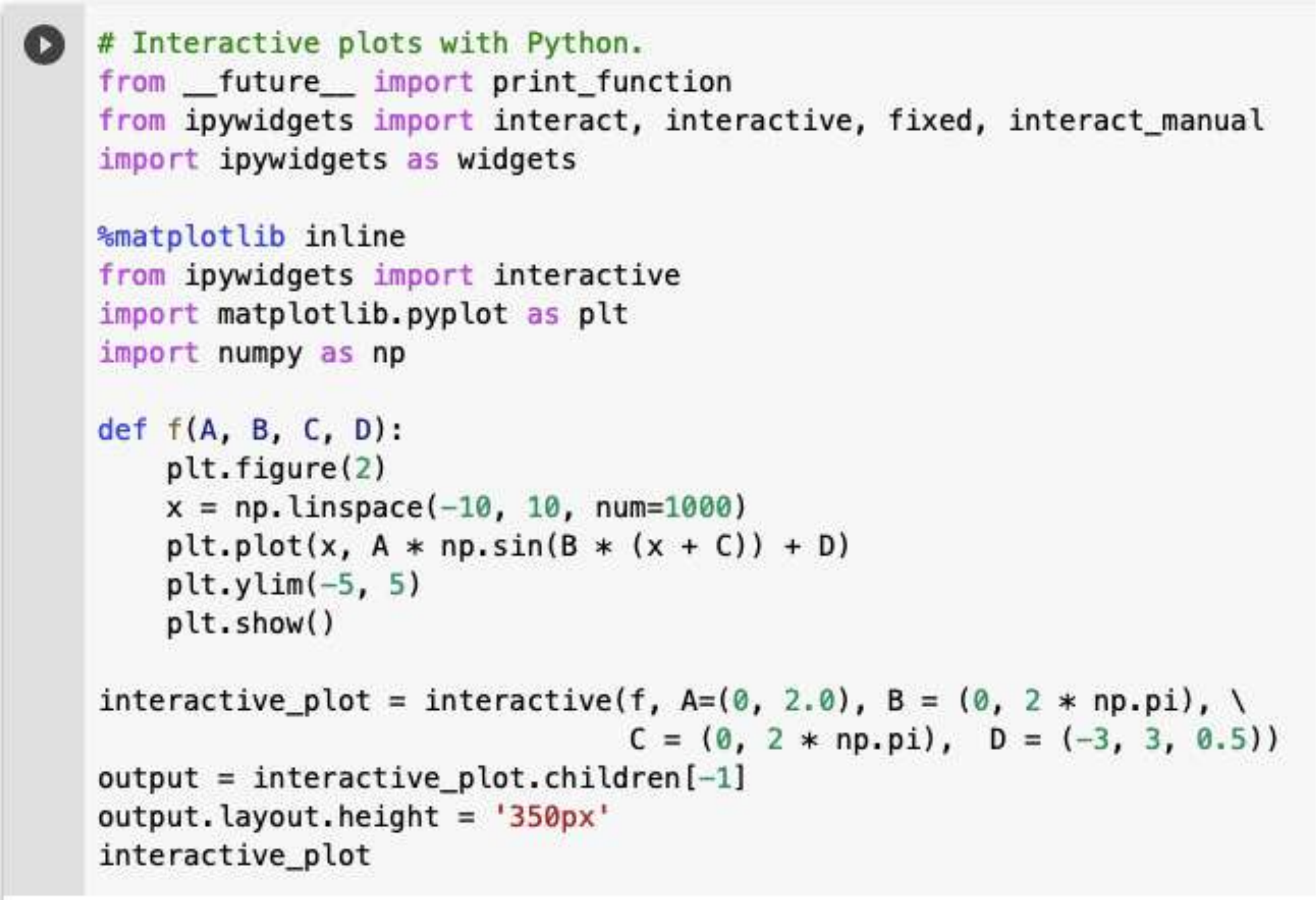


```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation , rc
from IPython.display import HTML
fig , ax = plt.subplots()
plt.close()
ax.set_xlim(0 , 2 * np.pi)          # Set domain.
ax.set_ylim(-2 , 2)                 # Set range.
line, = ax.plot([] , [] , lw = 2)   # Line width
def init():
  line.set_data([] , [])
  return (line,)
def animate(n):
  x = np.linspace(0 , 2 * np.pi , 100)    # The domain.
  y = np.sin(0.05 * x * n)                # The function to animate.
  line.set_data(x , y)
  return (line,)
anim = animation.FuncAnimation(fig , animate , init_func=init , frames = 100 , \
                               interval = 100 , blit = True)
HTML(anim.to_jshtml())
```



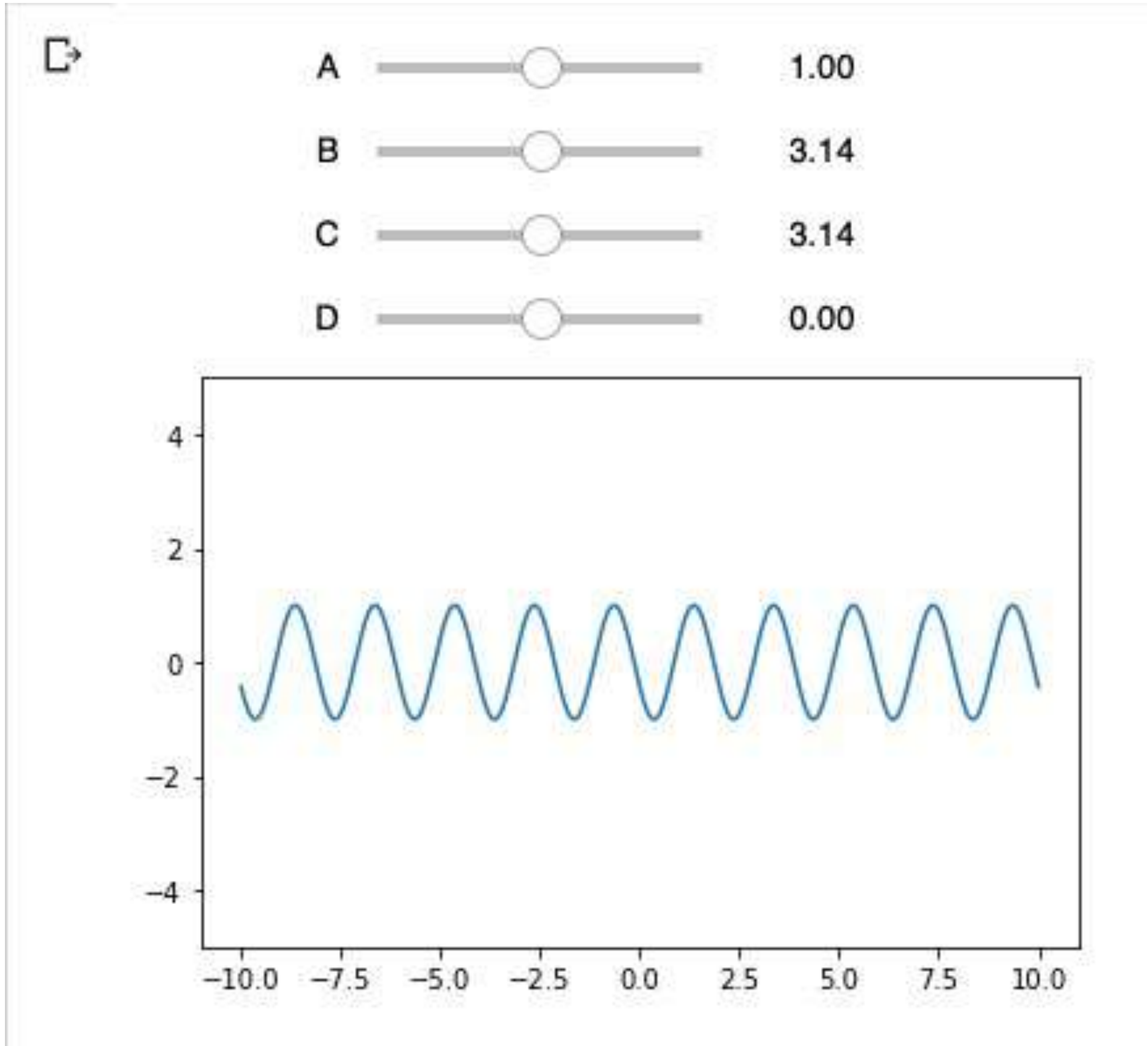
Edit to animate $y = e^{-0.01at}\sin(t)$, for $0 \le a \le 50$?

Interactive Plots in Google Colab



```python
# Interactive plots with Python.
from __future__ import print_function
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets

%matplotlib inline
from ipywidgets import interactive
import matplotlib.pyplot as plt
import numpy as np

def f(A, B, C, D):
    plt.figure(2)
    x = np.linspace(-10, 10, num=1000)
    plt.plot(x, A * np.sin(B * (x + C)) + D)
    plt.ylim(-5, 5)
    plt.show()

interactive_plot = interactive(f, A=(0, 2.0), B = (0, 2 * np.pi), \
                               C = (0, 2 * np.pi),  D = (-3, 3, 0.5))
output = interactive_plot.children[-1]
output.layout.height = '350px'
interactive_plot
```

Interactive Plots in Google Colab: End Session 3

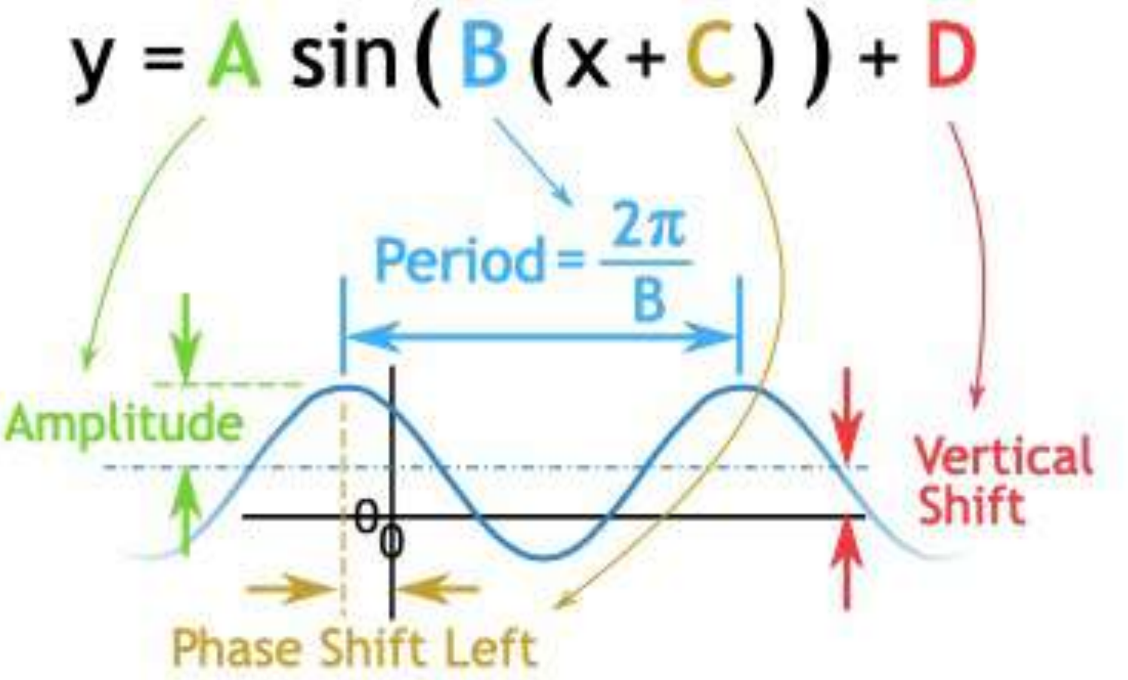


A 1.00

B 3.14

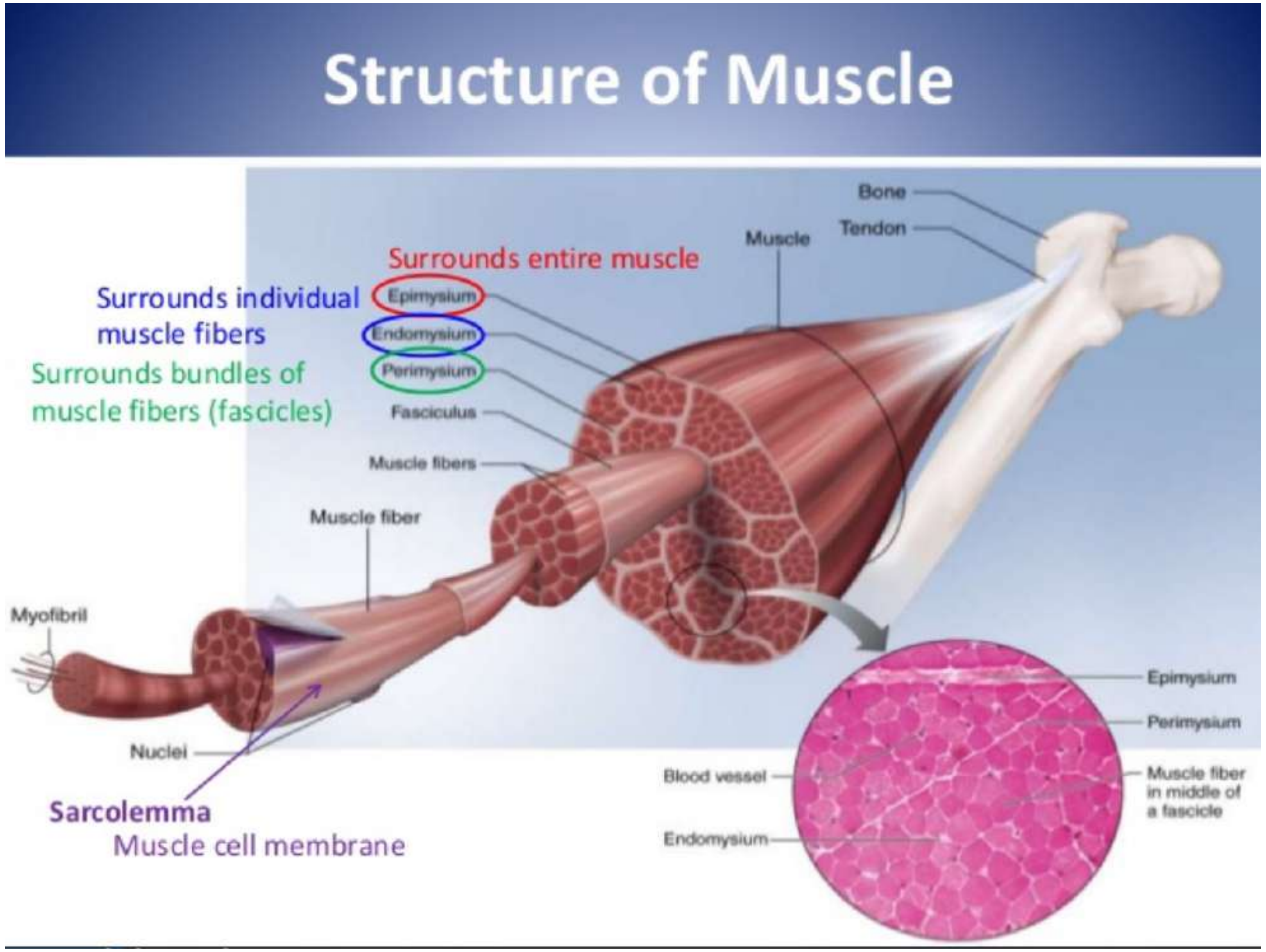
C 3.14

D 0.00



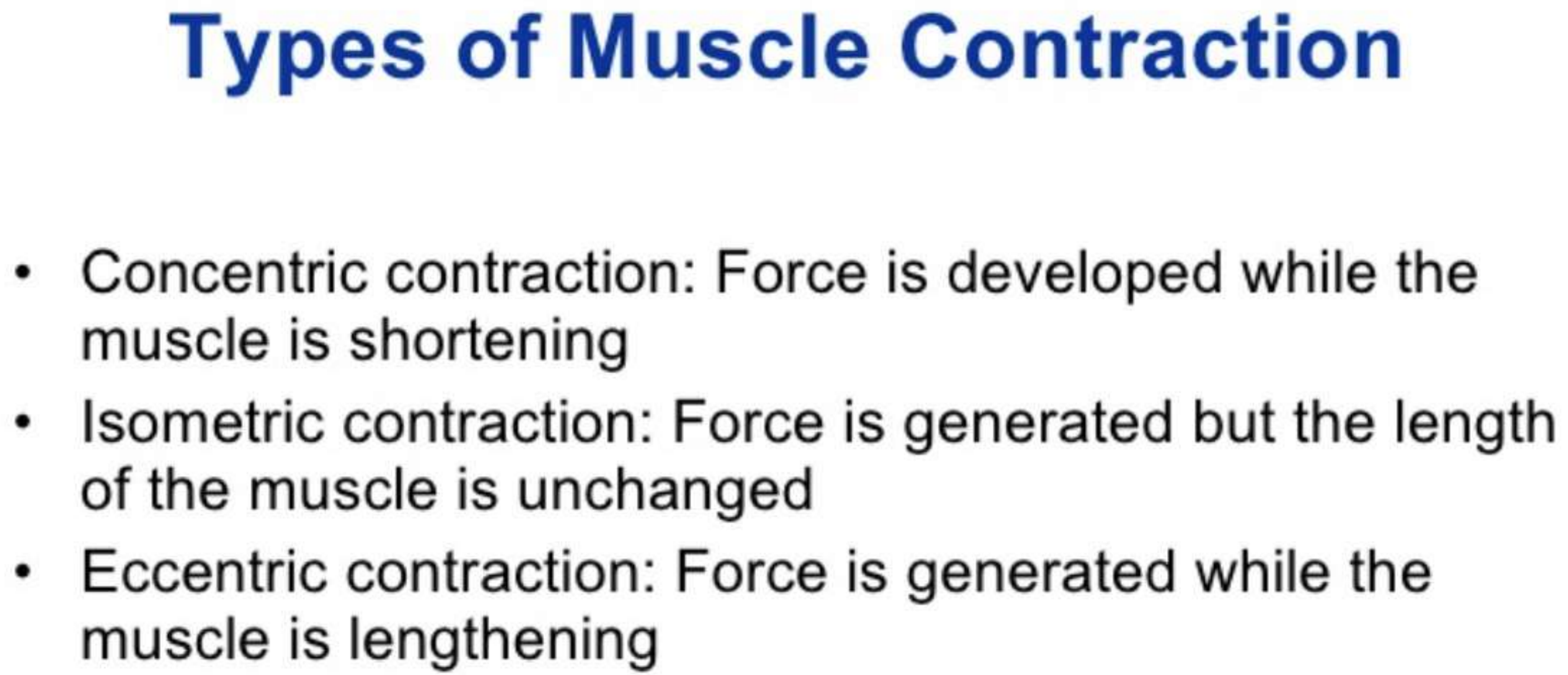
$$y = A \sin(B(x + C)) + D$$

Biological Models: Muscle Model: Start Session 4



Structure of Muscle

Muscle Model



Types of Muscle Contraction

- Concentric contraction: Force is developed while the muscle is shortening
- Isometric contraction: Force is generated but the length of the muscle is unchanged
- Eccentric contraction: Force is generated while the muscle is lengthening

Muscle Model (Modelling with Springs and Dampers)

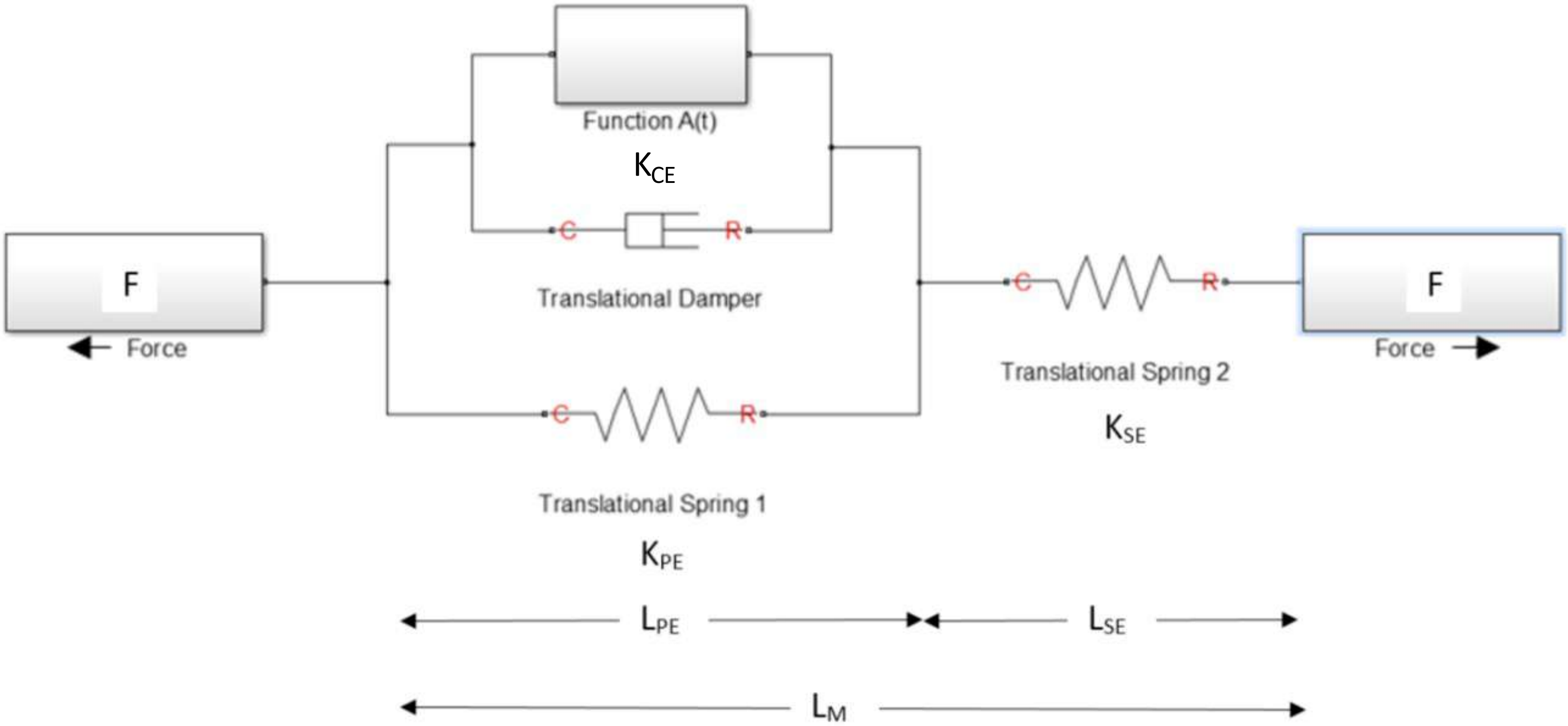
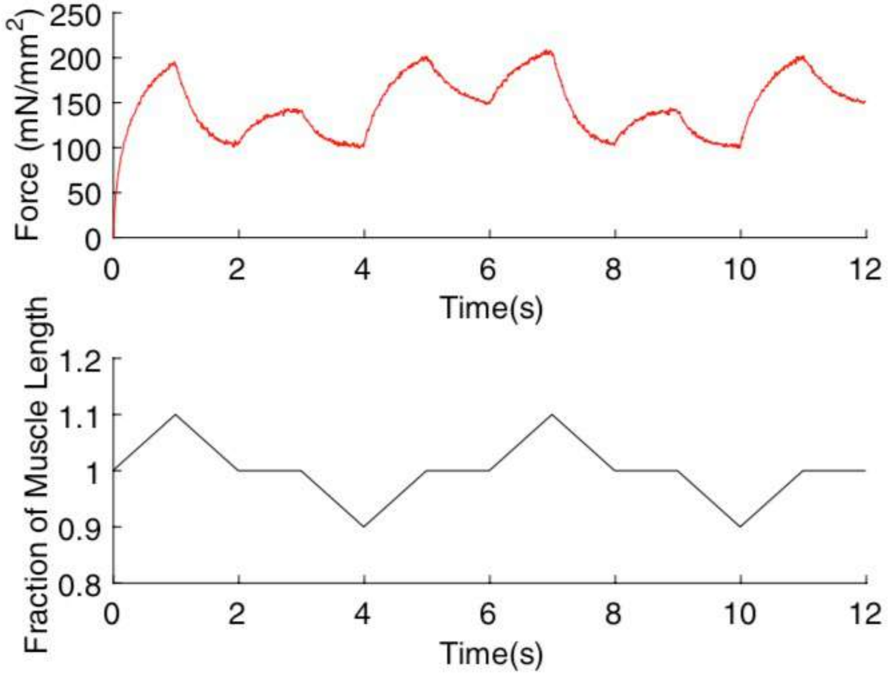


Fig. 18. A Simscape simulation of the Hill muscle model comprising of a series element (SE), a contractile element (CE) and a parallel element (PE).

Muscle Model (Python Program of Hill Model)

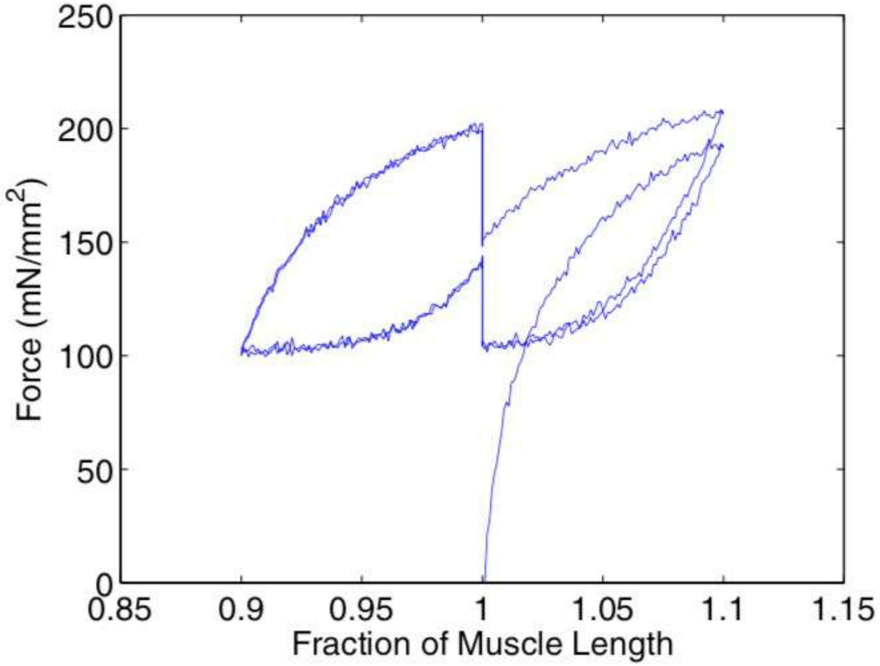
```python
# Muscle Hill model.
import numpy as np
import matplotlib.pyplot as plt

# From Hill's paper.
Length, a, b = 1200, 380 * 0.098, 0.325
P0 = a / 0.257
vm = P0 * b / a
alpha = P0 / 0.1
LSE0 = 0.3
k = a / 25

t = [0 + 0.01 * i for i in range(1201)]      # Time

# Stretching, holding and contracting muscle.
A = [1.001 + 0.001 * i for i in range(100)]  # Length A.
B = [1.099 - 0.001 * i for i in range(100)]  # Length B.
C = np.ones(100).tolist()                    # Length C.
D = [0.999 - 0.001 * i for i in range(100)]
E = [0.901 + 0.001 * i for i in range(100)]
F = np.ones(100).tolist()
G = [1.001 + 0.001 * i for i in range(100)]
H = [1.099 - 0.001 * i for i in range(100)]
HH = np.ones(100).tolist()
J = [0.999 - 0.001 * i for i in range(100)]
K = [0.901 + 0.001 * i for i in range(100)]
KK = np.ones(101).tolist()
L = A+B+C+D+E+F+G+H+HH+J+K+KK


# Muscle model continued...
LSE = np.zeros(1200).tolist()
LCE = np.zeros(1200).tolist()
P = np.zeros(1201).tolist()

# Hill's differential equations.
for i in range(1200):
    LSE[i] = 0.3 * P[i] / alpha
    LCE[i] = L[i] - LSE[i]
    dt = t[i + 1] - t[i]
    dL = L[i + 1] - L[i]
    dP = alpha * ((dL/dt) + b * ((P0 - P[i]) / (a + P[i]))) * dt
    P[i + 1] =P[i] + dP

P = np.array(P)
PP = (P0 / 100) * np.random.randn(1201)   # Add some noise.
P = P + PP
P = P.tolist()

plt.figure()
plt.plot(L, P)  # Plot length v Force.
plt.xlabel('Fraction of Muscle Length mm', fontsize = 15)
plt.ylabel('Force ($mN / mm^2$)', fontsize = 15)
plt.tick_params(labelsize = 15)
```

Muscle Model (Lengthening and Shortening)



(a)

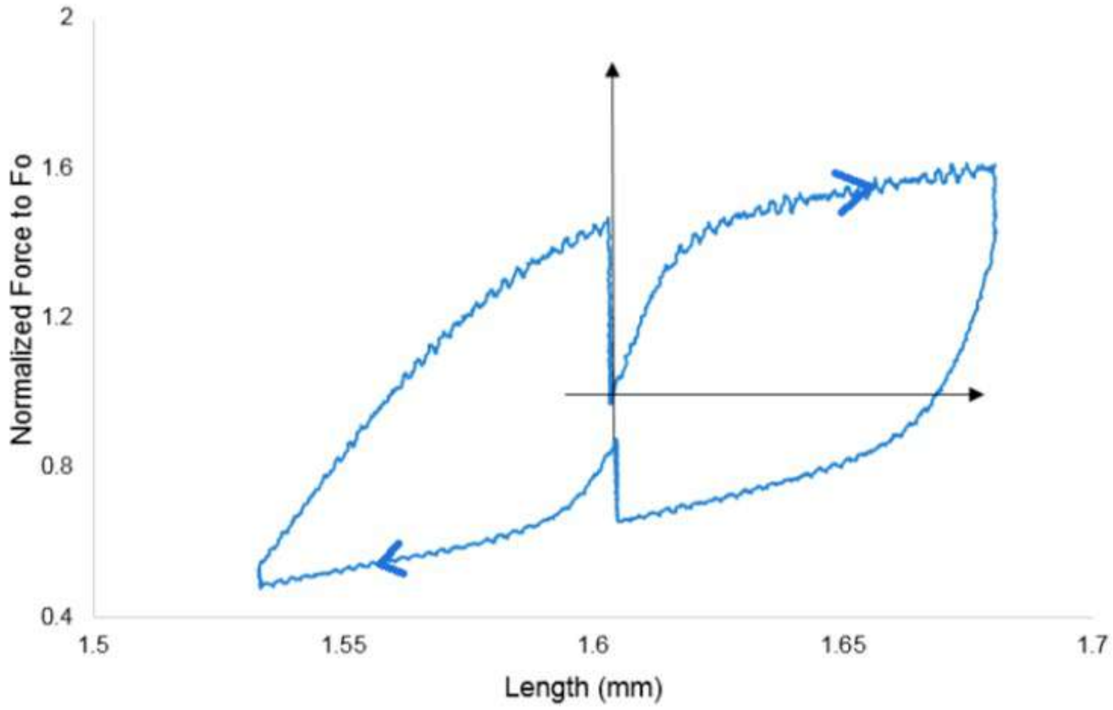
Lengthening and shortening with rests.

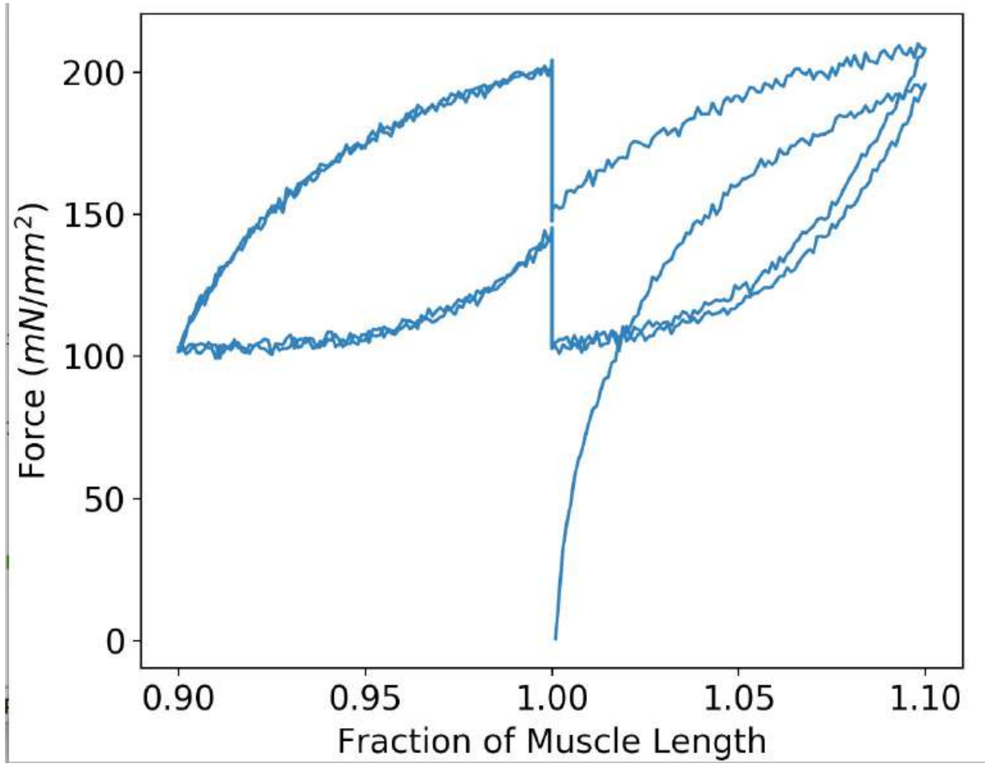


(b)

Hysteresis curves from modelling.

Muscle Model (Experimental and Mathematical Modelling Results)





Hysteresis curves from experiment.

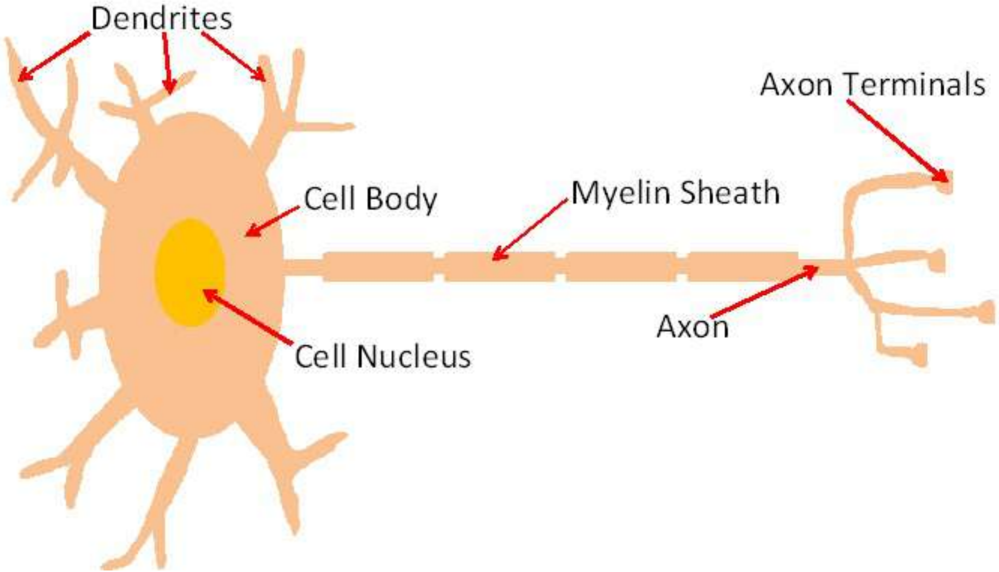
Hysteresis curves from modelling.

Ramos J, **Lynch S**, Jones DA & Degens H (2017) Hysteresis in muscle (Feature Article), International Journal of Bifurcation and Chaos 27, 1730003, 1-16.

A Simple Neuron Model: Fitzhugh-Nagumo Limit Cycle

$$\dot{u} = -u(u-\theta)(u-1) - v + \omega, \quad \dot{v} = \epsilon(u - \gamma v),$$

where u is a voltage, v is the recovery of voltage, θ is a threshold, γ is a shunting variable, and ω is a constant voltage.



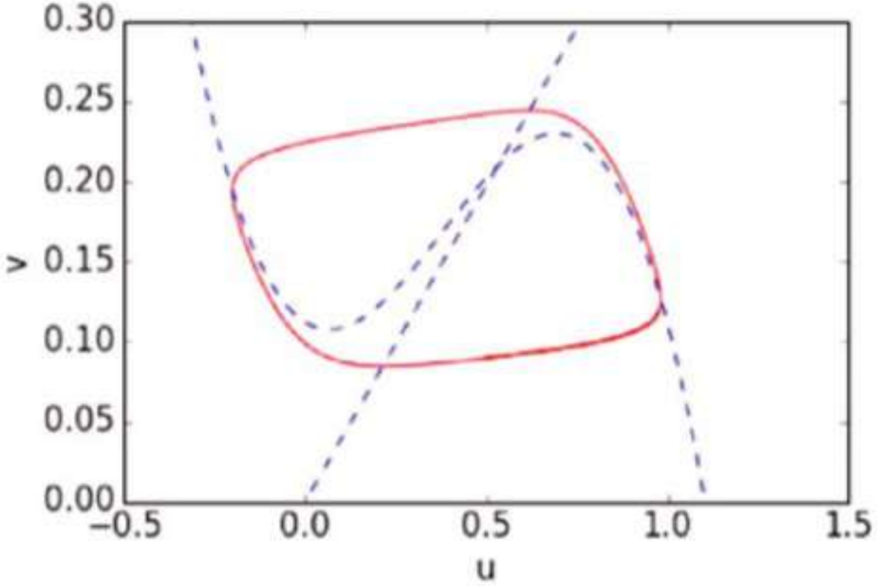
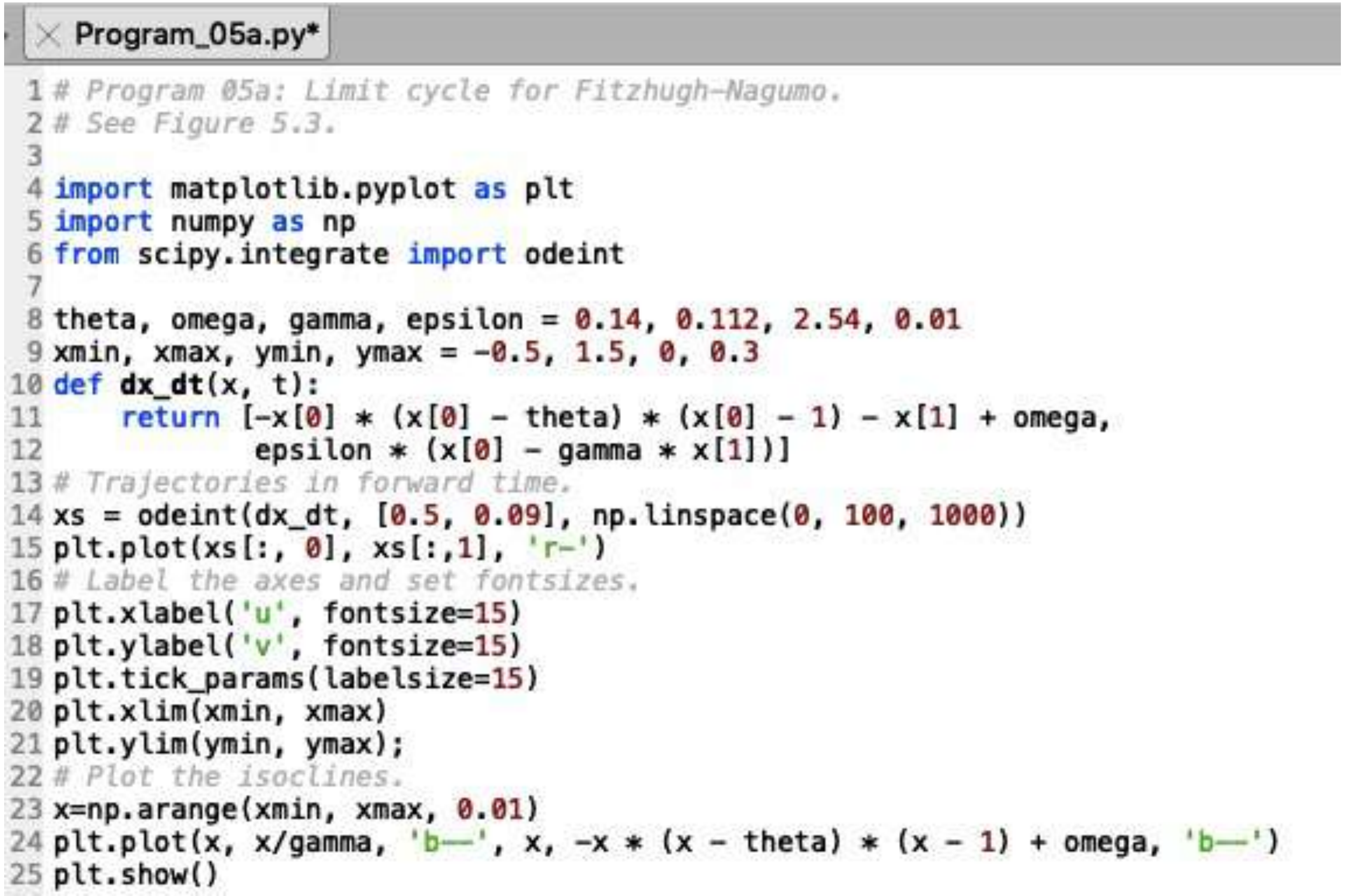


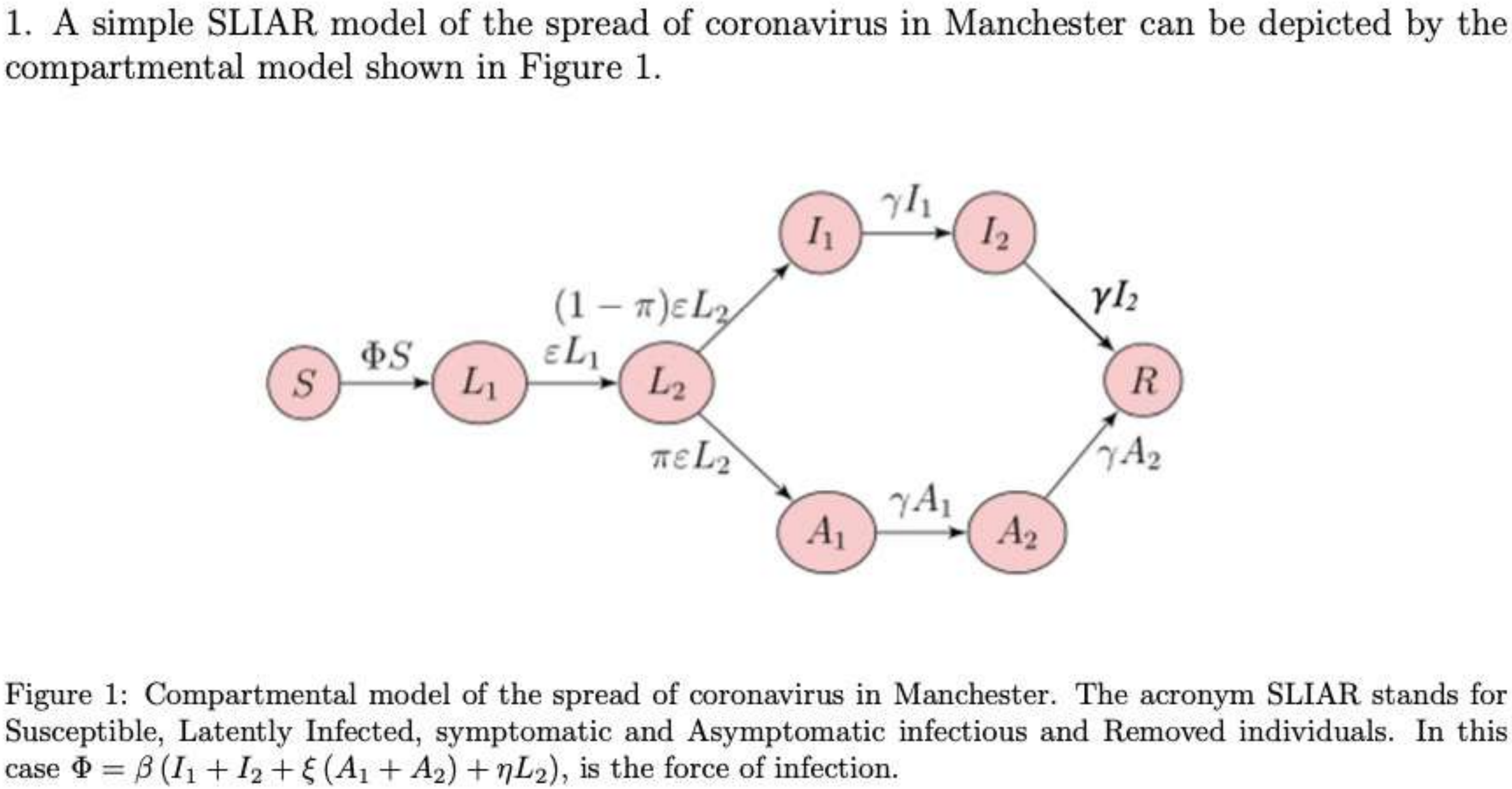
Figure 5.3: [Python] A limit cycle for the Fitzhugh-Nagumo oscillator. In this case, $\gamma = 2.54$, $\theta = 0.14$, $\omega = 0.112$, and $\epsilon = 0.01$. The blue dashed curves are the nullclines, where the trajectories cross horizontally and vertically.

A Simple Neuron Model: Fitzhugh-Nagumo Limit Cycle



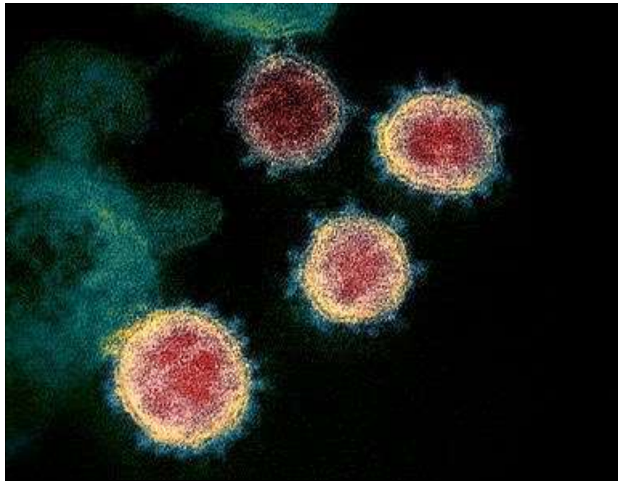
```python
# Program 05a: Limit cycle for Fitzhugh-Nagumo.
# See Figure 5.3.

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

theta, omega, gamma, epsilon = 0.14, 0.112, 2.54, 0.01
xmin, xmax, ymin, ymax = -0.5, 1.5, 0, 0.3
def dx_dt(x, t):
    return [-x[0] * (x[0] - theta) * (x[0] - 1) - x[1] + omega,
            epsilon * (x[0] - gamma * x[1])]
# Trajectories in forward time.
xs = odeint(dx_dt, [0.5, 0.09], np.linspace(0, 100, 1000))
plt.plot(xs[:, 0], xs[:,1], 'r-')
# Label the axes and set fontsizes.
plt.xlabel('u', fontsize=15)
plt.ylabel('v', fontsize=15)
plt.tick_params(labelsize=15)
plt.xlim(xmin, xmax)
plt.ylim(ymin, ymax);
# Plot the isoclines.
x=np.arange(xmin, xmax, 0.01)
plt.plot(x, x/gamma, 'b--', x, -x * (x - theta) * (x - 1) + omega, 'b--')
plt.show()
```

Compartmental Model Epidemiology: Coronavirus



1. A simple SLIAR model of the spread of coronavirus in Manchester can be depicted by the compartmental model shown in Figure 1.

Figure 1: Compartmental model of the spread of coronavirus in Manchester. The acronym SLIAR stands for Susceptible, Latently Infected, symptomatic and Asymptomatic infectious and Removed individuals. In this case $\Phi = \beta\left(I_1 + I_2 + \xi\left(A_1 + A_2\right) + \eta L_2\right)$, is the force of infection.



An electron microscope image showing the new coronavirus SARS-CoV-2.

For this model, β is the transmission coefficient, η and ξ are the attenuation factors for transmission by incubating and asymptomatic cases, respectively, π denotes the split between infectious and asymptomatic infectious individuals, and ϵ and γ describe the rates at which incubation and infectiousness end, respectively.

Compartmental Model Epidemiology

$$\frac{dS}{dt} = -\beta S(I_1 + I_2 + \xi(A_1 + A_2) + \eta L_2)$$

$$\frac{dL_1}{dt} = \beta S(I_1 + I_2 + \xi(A_1 + A_2) + \eta L_2) - \varepsilon L_1$$

$$\frac{dL_2}{dt} = \varepsilon(L_1 - L_2)$$

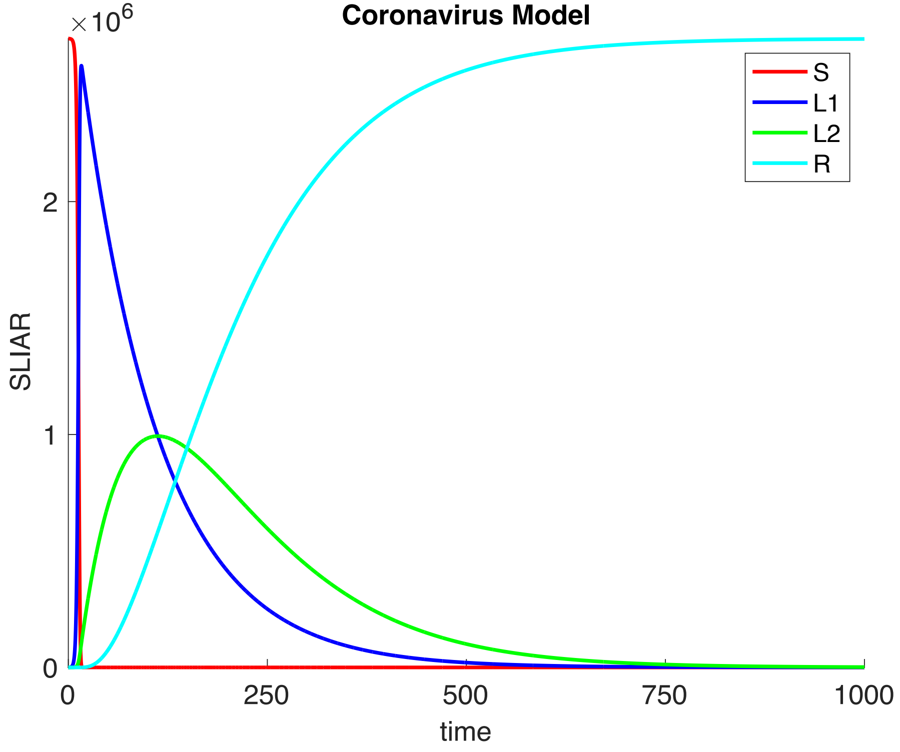
$$\frac{dI_1}{dt} = (1 - \pi)\varepsilon L_2 - \gamma I_1$$

$$\frac{dI_2}{dt} = \gamma(I_1 - I_2)$$

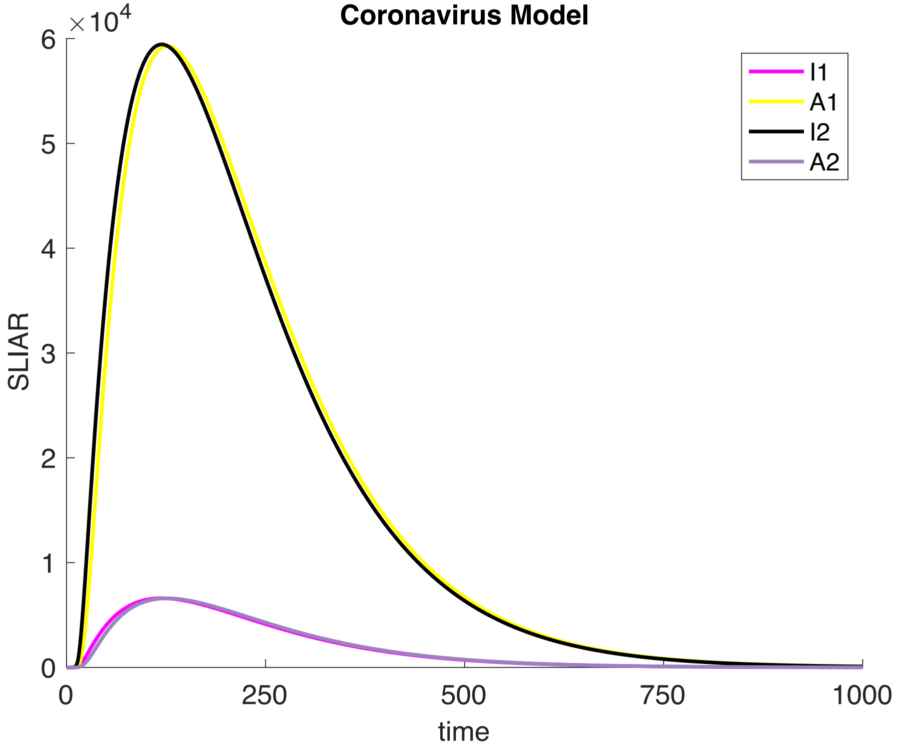
$$\frac{dA_1}{dt} = \pi\varepsilon L_2 - \gamma A_1$$

$$\frac{dA_2}{dt} = \gamma(A_1 - A_2)$$

$$\frac{dR}{dt} = \gamma(I_2 + A_2)$$

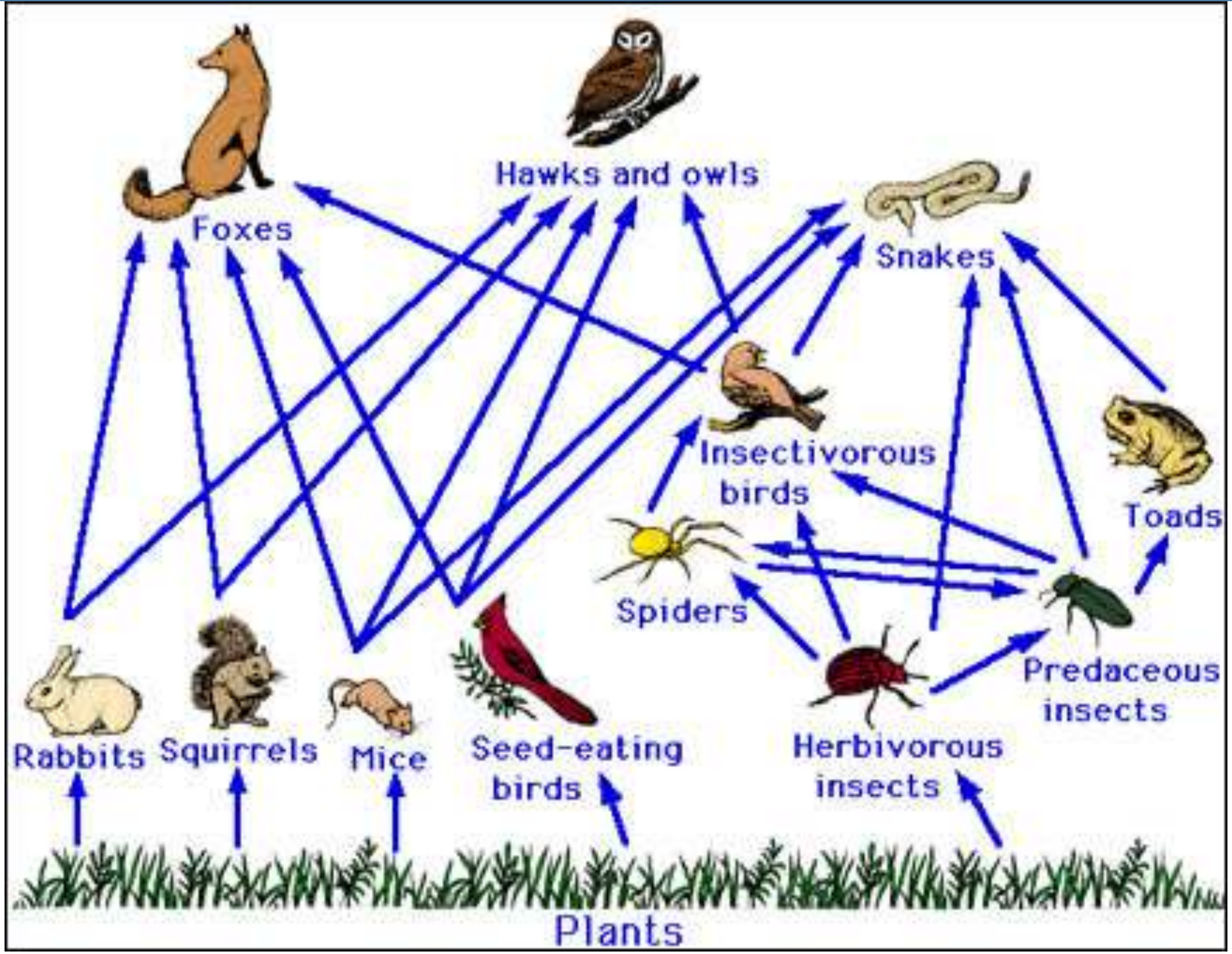


Coronavirus Model



Coronavirus Model

Chapter 4: Interacting Species in the UK



The Holling-Tanner Model

Example 3. Consider the specific Holling–Tanner model

$$\dot{x} = x\left(1 - \frac{x}{7}\right) - \frac{6xy}{(7+7x)}, \quad \dot{y} = 0.2y\left(1 - \frac{Ny}{x}\right)$$

Fig. Predator-Prey: Lynx and snowshoe hare.

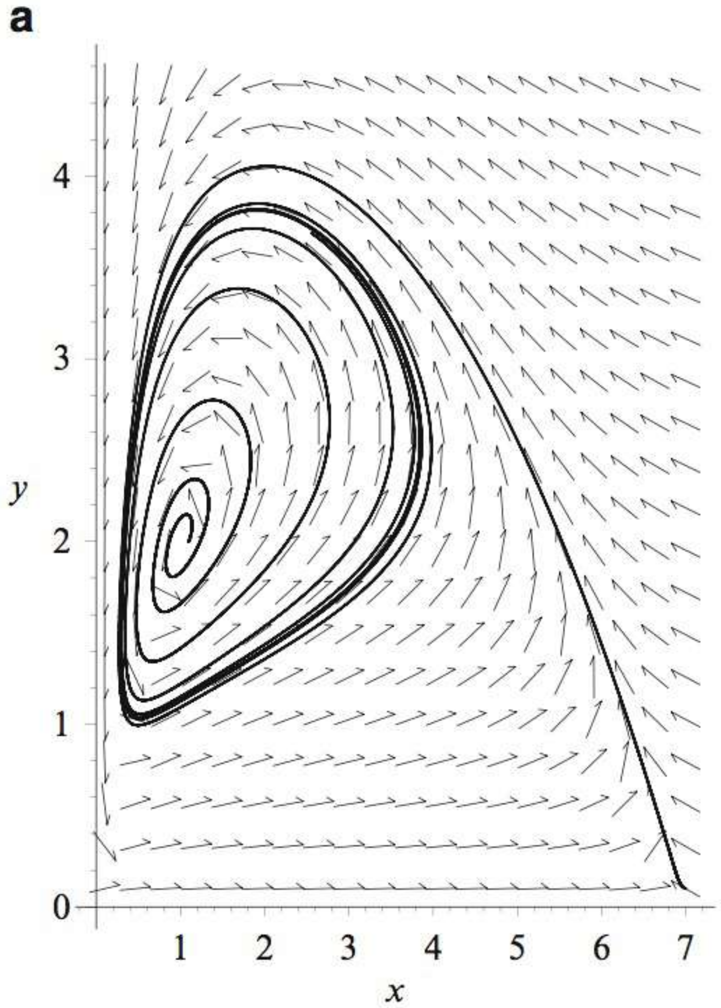
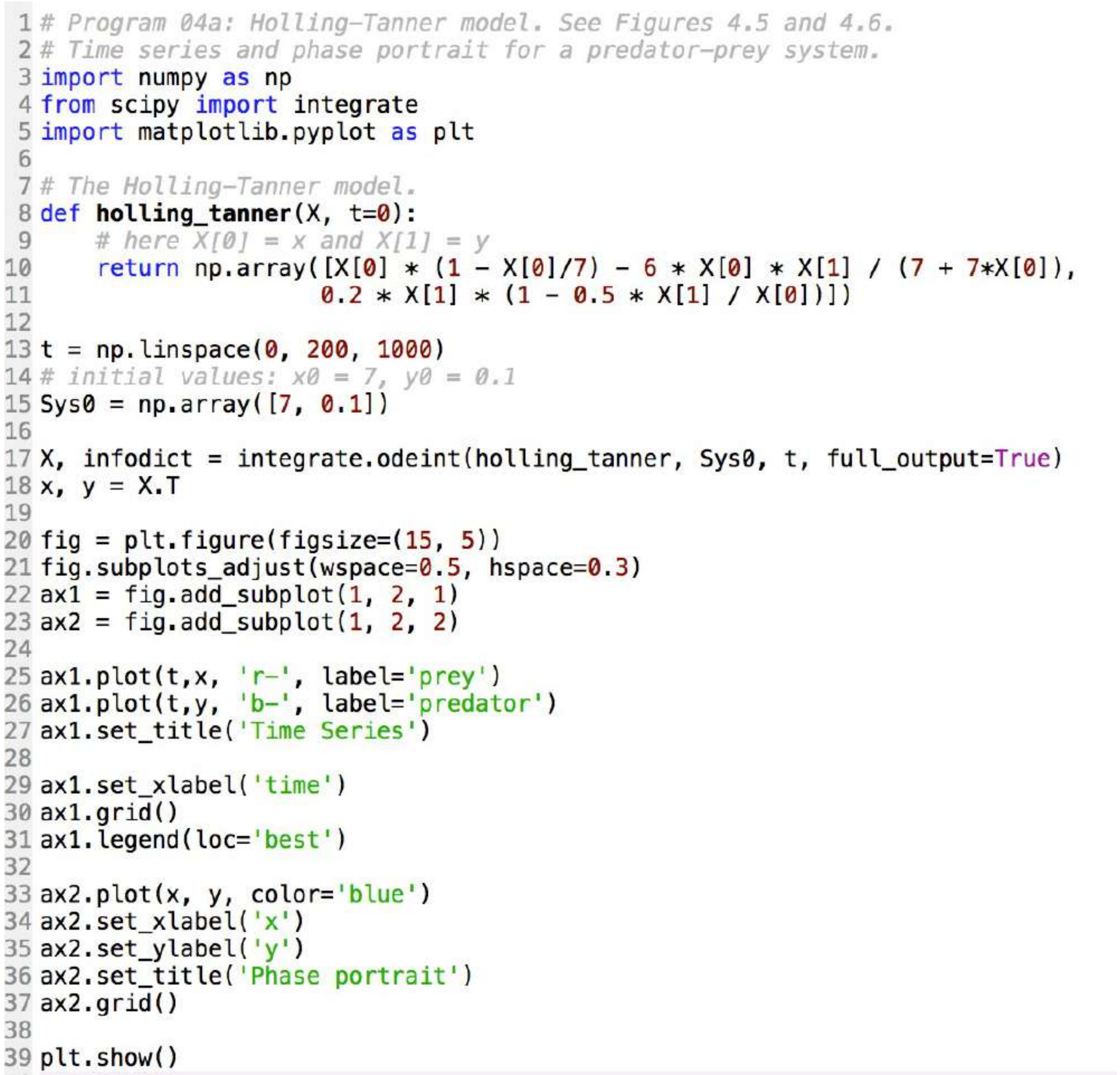
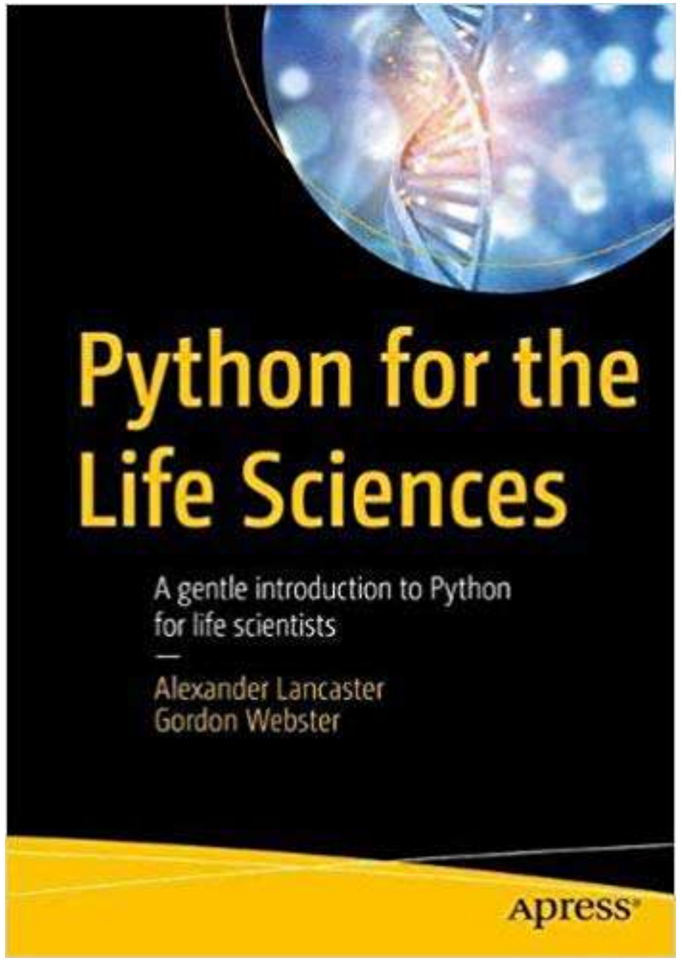


Fig. 4.5 A limit cycle of the Holling-Tanner Model

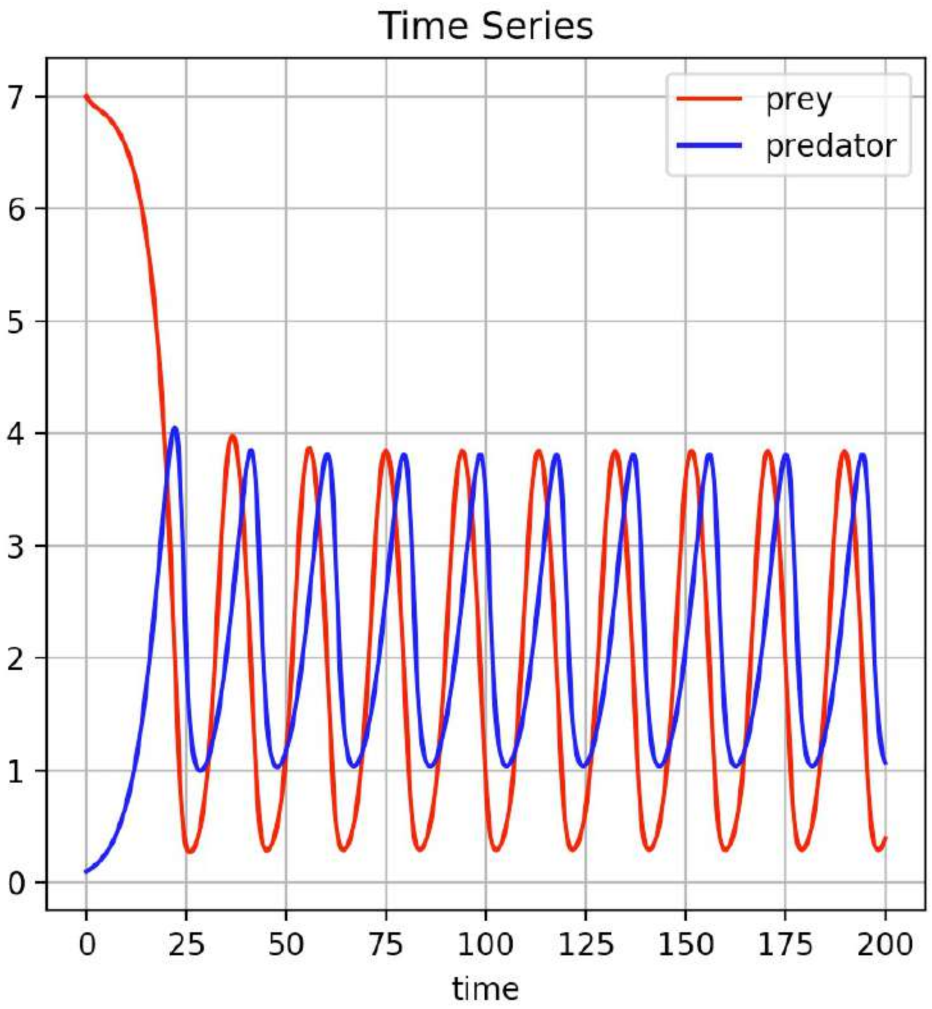
The Holling-Tanner Model (Python Program)

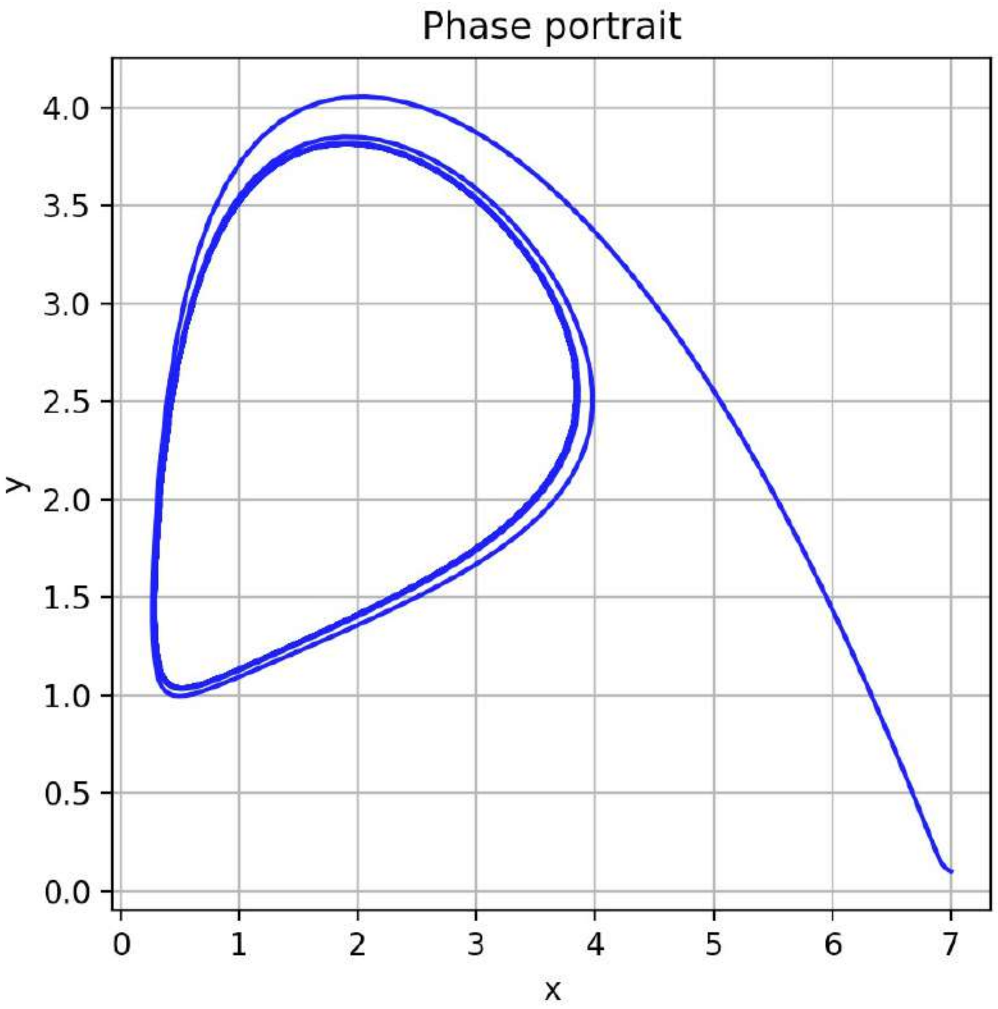


```
# Program 04a: Holling-Tanner model. See Figures 4.5 and 4.6.
# Time series and phase portrait for a predator-prey system.
import numpy as np
from scipy import integrate
import matplotlib.pyplot as plt

# The Holling-Tanner model.
def holling_tanner(X, t=0):
    # here X[0] = x and X[1] = y
    return np.array([X[0] * (1 - X[0]/7) - 6 * X[0] * X[1] / (7 + 7*X[0]),
                     0.2 * X[1] * (1 - 0.5 * X[1] / X[0])])

t = np.linspace(0, 200, 1000)
# initial values: x0 = 7, y0 = 0.1
Sys0 = np.array([7, 0.1])

X, infodict = integrate.odeint(holling_tanner, Sys0, t, full_output=True)
x, y = X.T

fig = plt.figure(figsize=(15, 5))
fig.subplots_adjust(wspace=0.5, hspace=0.3)
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)

ax1.plot(t,x, 'r-', label='prey')
ax1.plot(t,y, 'b-', label='predator')
ax1.set_title('Time Series')

ax1.set_xlabel('time')
ax1.grid()
ax1.legend(loc='best')

ax2.plot(x, y, color='blue')
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.set_title('Phase portrait')
ax2.grid()

plt.show()
```

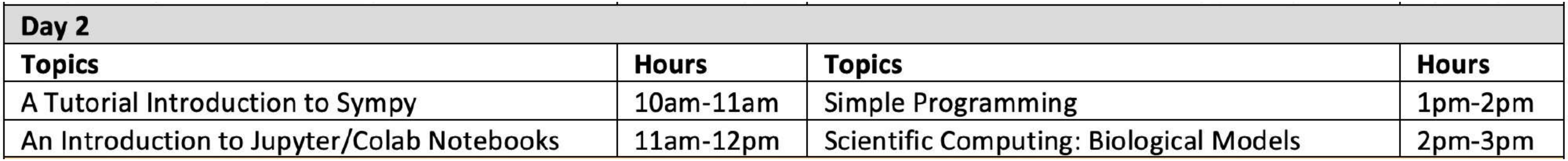


The Holling-Tanner Model





End Day 2 Summary



Day 2			
Topics	**Hours**	**Topics**	**Hours**
A Tutorial Introduction to Sympy	10am-11am	Simple Programming	1pm-2pm
An Introduction to Jupyter/Colab Notebooks	11am-12pm	Scientific Computing: Biological Models	2pm-3pm

You may also find the Jupyter notebook for A-level Mathematics useful:

http://www.doc.mmu.ac.uk/STAFF/S.Lynch/Python_for_A_Level_Mathematics_and_Beyond.html

Python for A-Level Mathematics, undergraduate Mathematics and employability:

https://www.mathscareers.org.uk/python-for-a-level-maths-undergraduate-maths-and-employability/

Manchester
Metropolitan
University