

5-day Hands-on Workshop on:

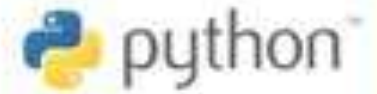
Python for Scientific Computing and TensorFlow for Artificial Intelligence

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

In[1]: `from sympy import *`

In[2]: `x, y = symbols(' x y ')`

In[3]: `factor(x**2 - y**2)`

In[4]: `solve(x**2 - 4*x - 3, x)`

In[5]: `apart(1 / ((x + 2)*(x + 1)))`

In[6]: `trigsimp(cos(x) - cos(x)**3)`

In[7]: `limit(x / sin(x), x, 0)`

Comments

Import everything from the sympy library.

Declare x and y symbolic.

Factorize $x^2 - y^2 = (x - y)(x + y)$.

Solve an algebraic equation, $x^2 - 4x - 3 = 0$.

Partial fractions.

Simplify a trig expression.

Limits, $\lim_{x \rightarrow 0} \frac{x}{\sin(x)}$.

A Tutorial Introduction to Sympy (SYMbolic PYthon)

In[8]: <code>diff(x**2 - 7*x + 8, x)</code>	# Differentiate with respect to x .
In[9]: <code>diff(5*x**7, x, 3)</code>	# Differentiate with respect to x three times.
In[10]: <code>(exp(x)*cos(x)).series(x, 0, 10)</code>	# Taylor series expansion around $x = 0$.
In[11]: <code>integrate(x**2 - 7*x + 8, x)</code>	# Indefinite integration, $\int x^2 - 7x + 8 \, dx$
In[12]: <code>integrate(x**2 - 7*x + 8, (x, 1, 2))</code>	# Definite integration, $\int_{x=1}^2 x^2 - 7x + 8 \, dx$.
In[13]: <code>summation(1 / x**2, (x, 1, oo))</code>	# Infinite summation, $\sum_{x=1}^{\infty} \frac{1}{x^2}$.
In[14]: <code>solve([x+5*y-2, -3*x+6*y-15],[x,y])</code>	# Solve simultaneous equations.
In[15]: <code>A = Matrix([[1, -1], [2, 3]])</code>	# The matrix, $A = \begin{pmatrix} 1 & -1 \\ 2 & 3 \end{pmatrix}$.
In[16]: <code>B = Matrix([[0, 2], [3, 3]])</code>	# The matrix, $B = \begin{pmatrix} 0 & 2 \\ 3 & 3 \end{pmatrix}$.

A Tutorial Introduction to Sympy (SYMbolic PYthon)

```
In[17]: 2 * A + 3 * B
```

```
In[18]: A * B
```

```
In[19]: A.row(0)
```

```
In[20]: A.row(1)
```

```
In[21]: A.T
```

```
In[22]: A[0, 1]
```

```
In[23]: A**(-1)
```

```
In[24]: A.det()
```

```
In[25]: zeros(3, 3)
```

```
In[26]: ones(1, 5)
```

```
In[27]: N(pi, 500)
```

```
In[28]: quit
```

```
# Matrix algebra.
```

```
# Matrix multiplication. Row by column.
```

```
# Access row 1 (Zero-based indexing).
```

```
# Access row 2 (Zero-based indexing).
```

```
# The transpose matrix, swap rows  $\leftrightarrow$  columns.
```

```
# The element in row one, column 2.
```

```
# The inverse matrix,  $A^{-1}$ .
```

```
# The determinant.
```

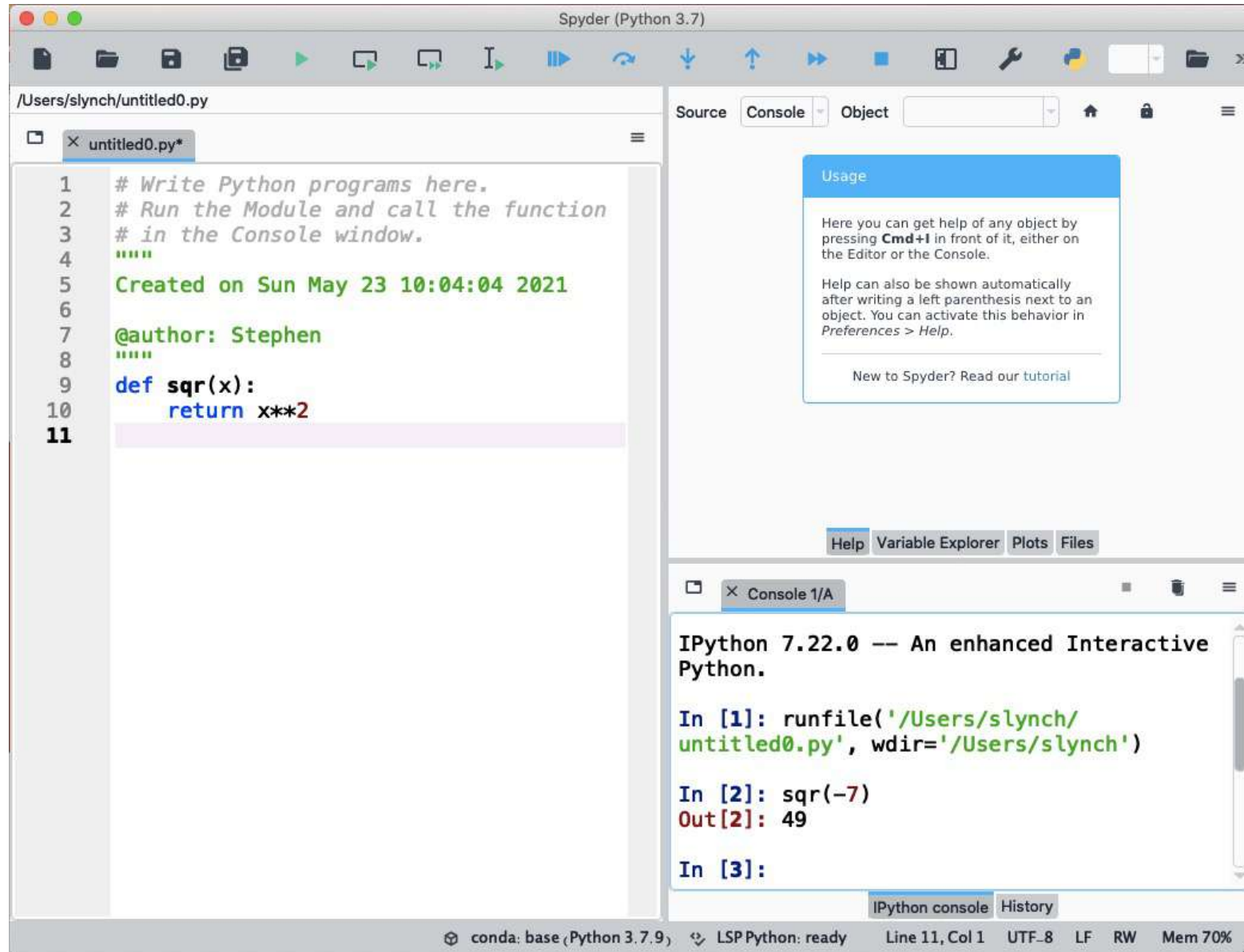
```
# Gives a  $3 \times 3$  matrix of zeros.
```

```
# A  $1 \times 5$  matrix of ones.
```

```
# Numerical evaluation,  $\pi$  to 500 significant figures.
```

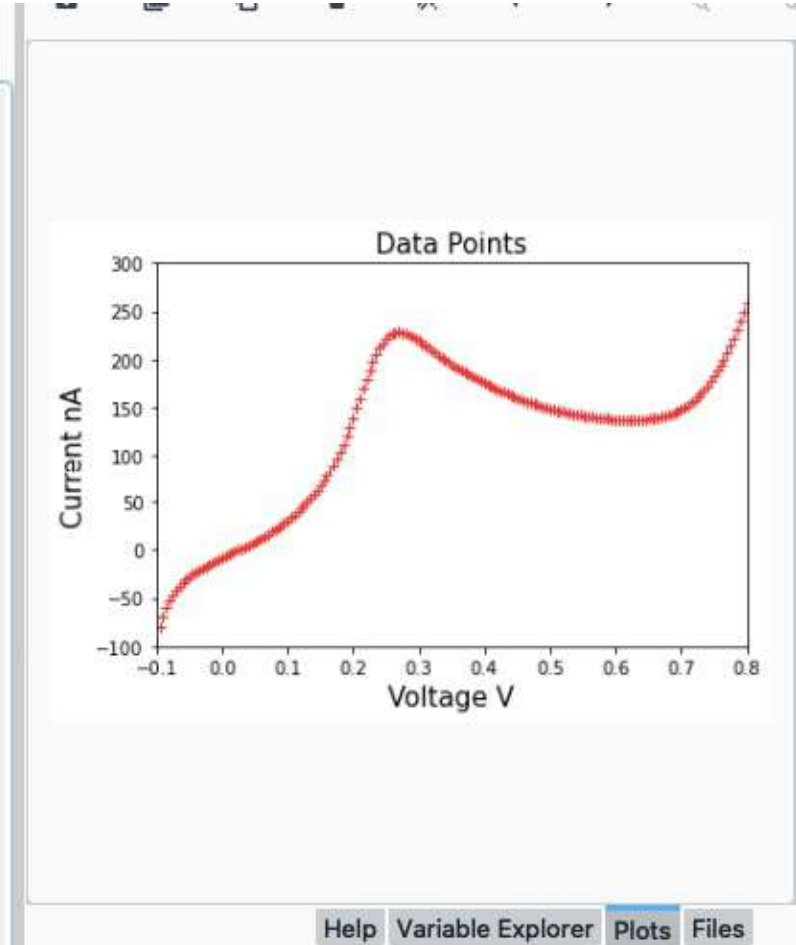
```
# Quits the IPython console and restarts the kernel.
```

A Simple Program in Spyder



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

```
Polyfit_Data.py*
1  # Load data from an excel spreadsheet and fit a polynomial of
2  # degree 20 to the data.
3
4  import pandas as pd
5  import numpy as np
6  import matplotlib.pyplot as plt
7  # The excel spreadsheet has to be in the same folder as Polyfit_Data.py.
8  mydata = pd.read_excel('Kam_Data_nA.xlsx', sheet_name = 'Sheet1')
9  df = pd.DataFrame(mydata)
10 xs = (df['Volt'])
11 ys = (df['nAJ'])
12 z = np.polyfit(xs, ys, 20)
13 p = np.poly1d(z)
14 print('polynomial = ', p)
15
16 plt.axis([-0.1, 0.8, -100, 300])
17 plt.xlabel('Voltage V', fontsize=15)
18 plt.ylabel('Current nA', fontsize=15)
19 plt.title('Data Points', fontsize=15)
20 plt.plot(xs, ys, 'r+')
21 plt.show()
```



Anaconda: Launch a Jupyter Notebook: Start Session 2

The screenshot displays the Anaconda Navigator desktop application. At the top, the 'ANACONDA NAVIGATOR' logo is on the left, and 'Upgrade Now' and 'Sign in to Anaconda.org' buttons are on the right. A left-hand sidebar contains navigation links: 'Home' (with a house icon), 'Environments' (with a cube icon), 'Learning' (with a book icon), and 'Community' (with a group of people icon). Below these are links for 'Documentation' and 'Developer Blog', and social media icons for Twitter, YouTube, and GitHub at the bottom. The main area is titled 'Applications on' followed by a dropdown menu set to 'base (root)' and a 'Channels' button. A 'Refresh' button is in the top right of the main area. The applications are arranged in a grid of eight cards. The top row includes JupyterLab (3.0.14), Jupyter Notebook (6.3.0), Qt Console (5.0.3), and Spyder (5.0.0). The bottom row includes Glueviz (1.0.0), Orange 3 (3.26.0), and RStudio (1.1.456). Each card features an icon, the application name, version, a brief description, and a button to either 'Launch' or 'Install'. A red arrow originates from the text 'Click here to open a new Python 3 ipynb notebook' and points directly to the 'Launch' button of the Jupyter Notebook application card.

ANACONDA NAVIGATOR

Upgrade Now Sign in to Anaconda.org

Home Environments Learning Community

Documentation Developer Blog


Twitter YouTube GitHub

Applications on base (root) Channels Refresh

Application	Version	Description	Action
JupyterLab	3.0.14	An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.	Launch
Jupyter Notebook	6.3.0	Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.	Launch
Qt Console	5.0.3	PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.	Launch
Spyder	5.0.0	Scientific PYTHON Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features	Launch
Glueviz	1.0.0	Multidimensional data visualization across files. Explore relationships within and among related datasets.	Install
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.	Install
RStudio	1.1.456	A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.	Install

Click here to open a new Python 3 ipynb notebook

Anaconda: Launch a Jupyter Notebook

 jupyter

Quit

Logout

Files

Running

Clusters

Click here to open a new Python 3 ipynb notebook

Upload

New

Notebook:

Python 3

Other:

Text File

Folder

Terminal

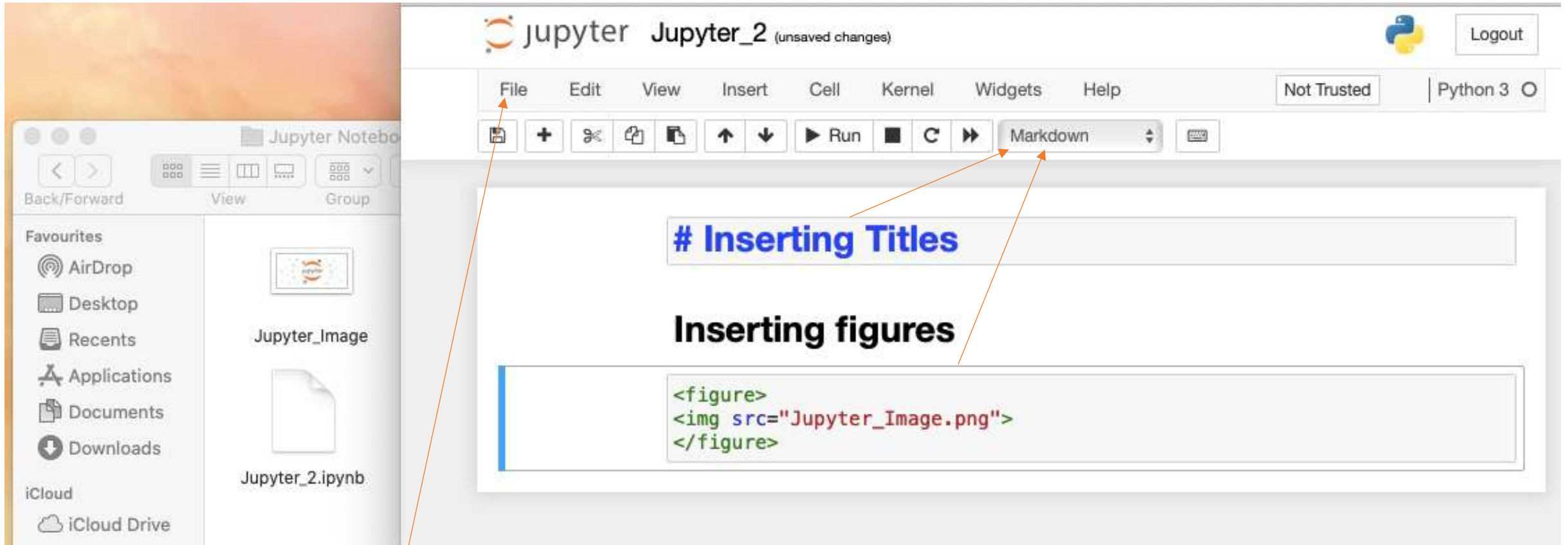
Select items to perform actions on them.

0

/

<input type="checkbox"/>	Desktop		
<input type="checkbox"/>	Documents		
<input type="checkbox"/>	Downloads		
<input type="checkbox"/>	Dropbox		
<input type="checkbox"/>	Movies		2 months ago
<input type="checkbox"/>	Music		2 months ago
<input type="checkbox"/>	Pictures		2 months ago
<input type="checkbox"/>	Public		2 months ago
<input type="checkbox"/>	Untitled.ipynb	a month ago	555 B
<input type="checkbox"/>	matlab_crash_dump.83074-1	a month ago	8.48 kB

Inserting Titles and Figures



The screenshot displays a Jupyter Notebook interface. On the left is a sidebar with a file explorer showing 'Jupyter_Image' and 'Jupyter_2.ipynb'. The main area shows a notebook titled 'Jupyter_2 (unsaved changes)'. The top menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. Below the menu is a toolbar with icons for file operations, navigation, and execution. The notebook content consists of two cells: the first is a Markdown cell with the heading '# Inserting Titles', and the second is a code cell containing the HTML code for inserting an image:

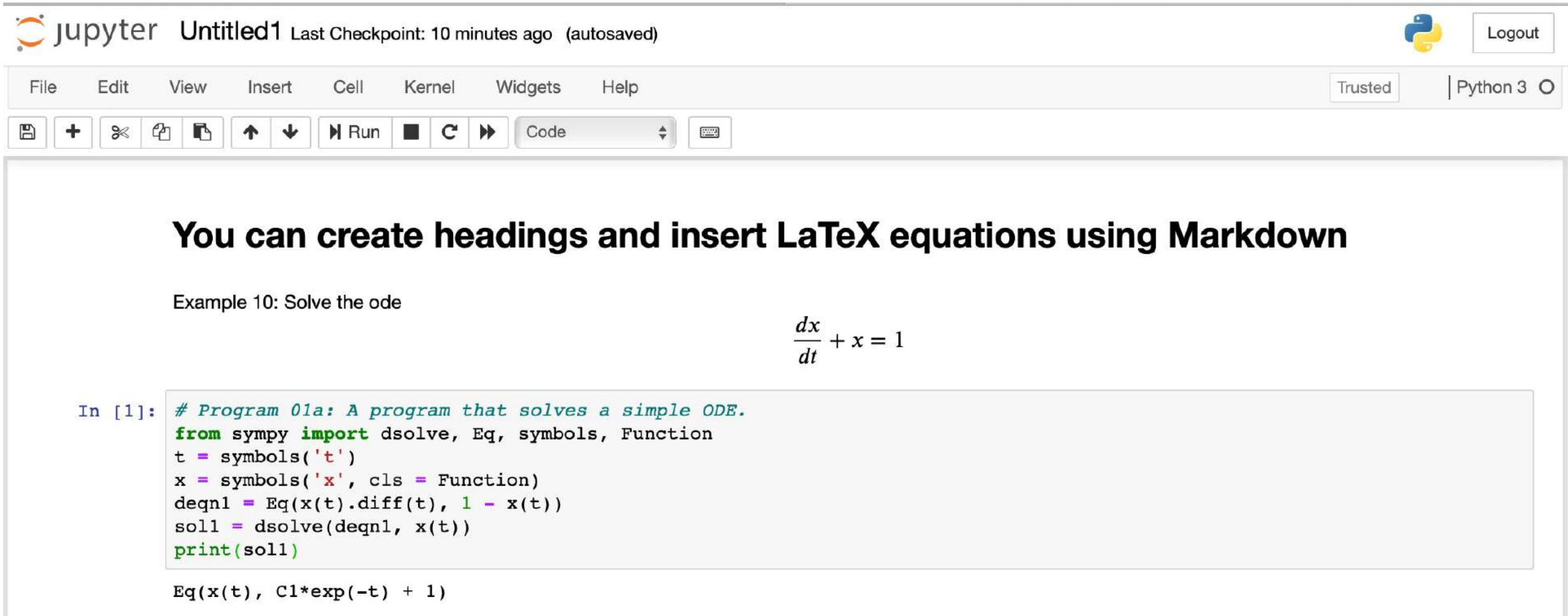
```
<figure>

</figure>
```

 Three orange arrows point from the 'File' menu, the 'Inserting figures' text, and the code cell to a text box at the bottom.

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

Simple Programming with Jupyter Notebooks (Solving ODEs)



The screenshot shows a Jupyter Notebook interface. At the top, the title bar says "jupyter Untitled1 Last Checkpoint: 10 minutes ago (autosaved)". The top right has a "Logout" button. Below the title bar is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". To the right of the menu bar are "Trusted" and "Python 3" buttons. Below the menu bar is a toolbar with icons for saving, adding, deleting, copying, pasting, undo, redo, and running code. The main area of the notebook contains a heading "You can create headings and insert LaTeX equations using Markdown". Below the heading is the text "Example 10: Solve the ode". To the right of this text is the differential equation $\frac{dx}{dt} + x = 1$. Below the text and equation is a code cell with the following code:

```
In [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

 jupyter

Untitled1 Last Checkpoint: 19 minutes ago (unsaved changes)

 Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

           Code 

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]: # 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)

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

import matplotlib.pyplot as plt
import numpy as np

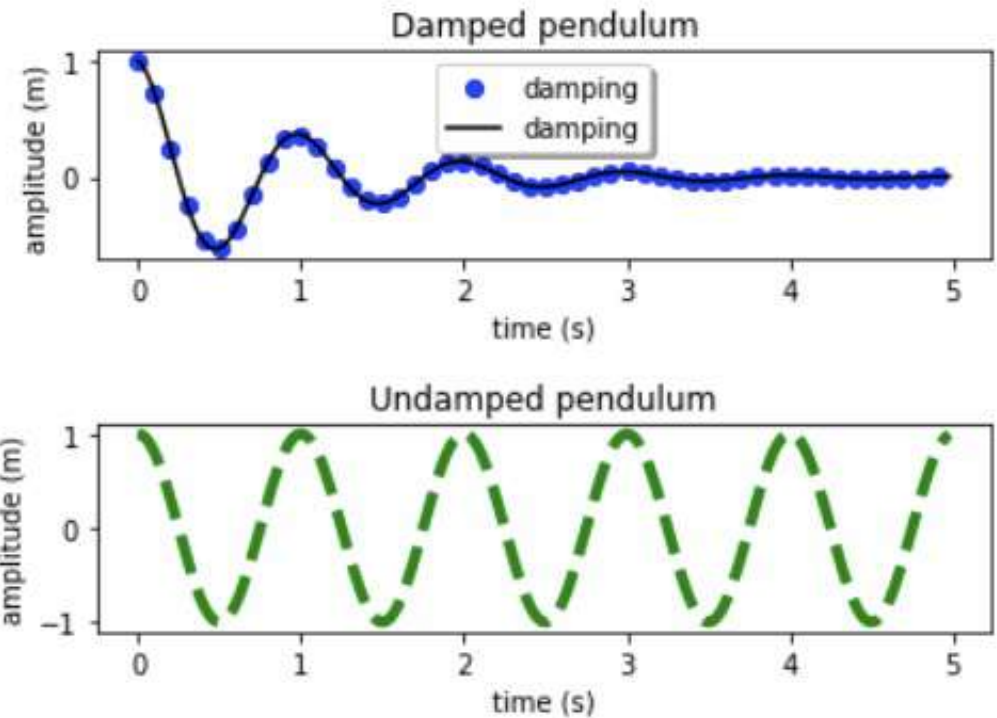
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

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))
```

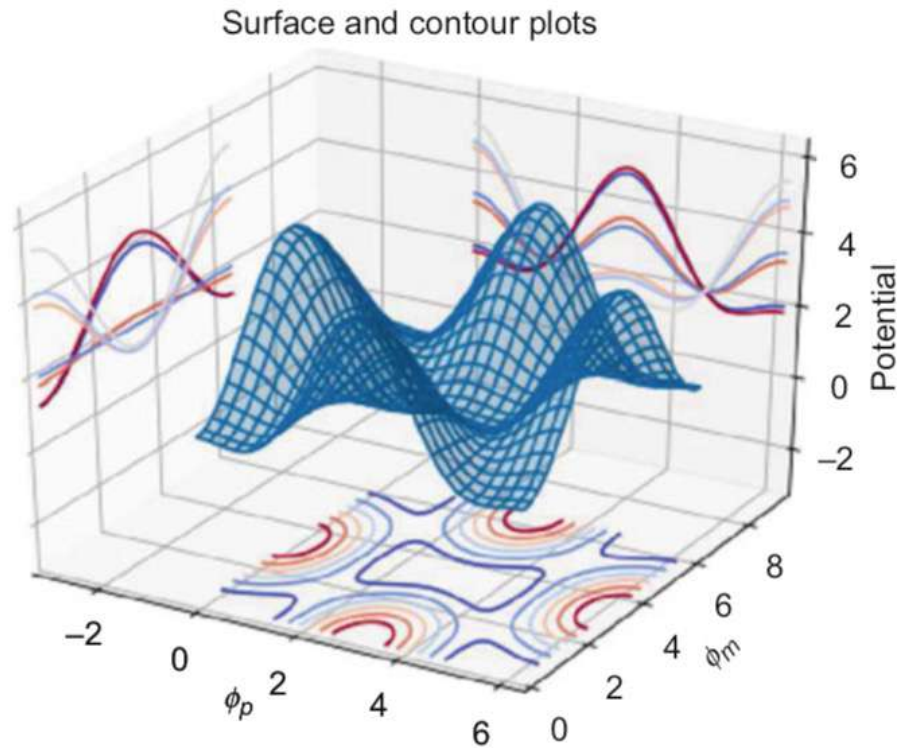
Simple Programming (Surface and Contour Plot)

```
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)



You can rotate the figure in Spyder.

In the Console window type:

```
In[1]: %matplotlib qt5
```

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.

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()
```

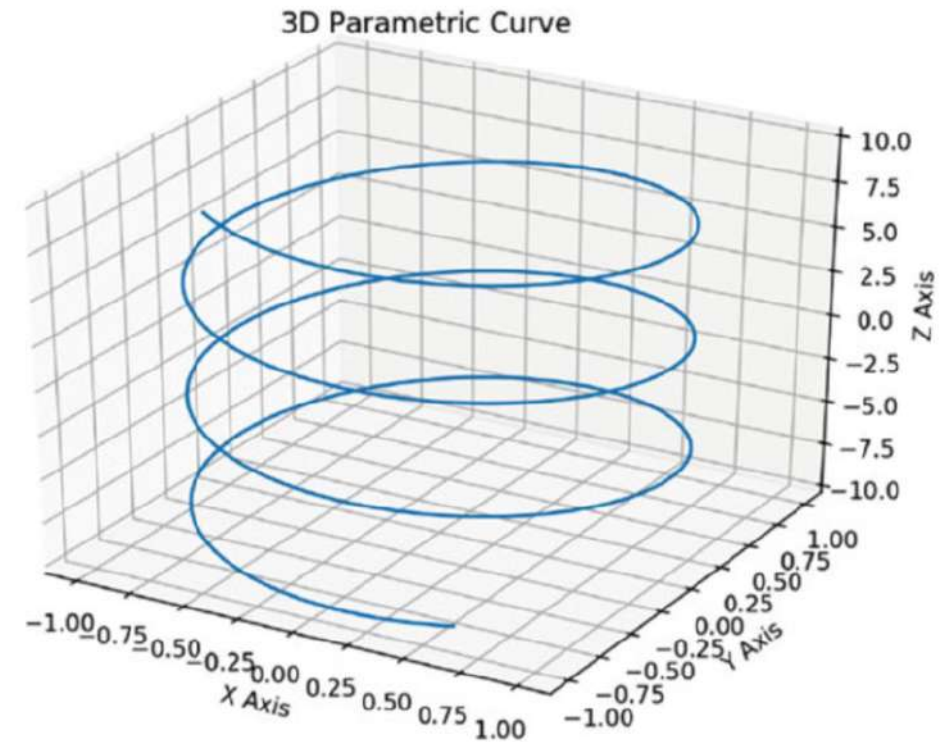
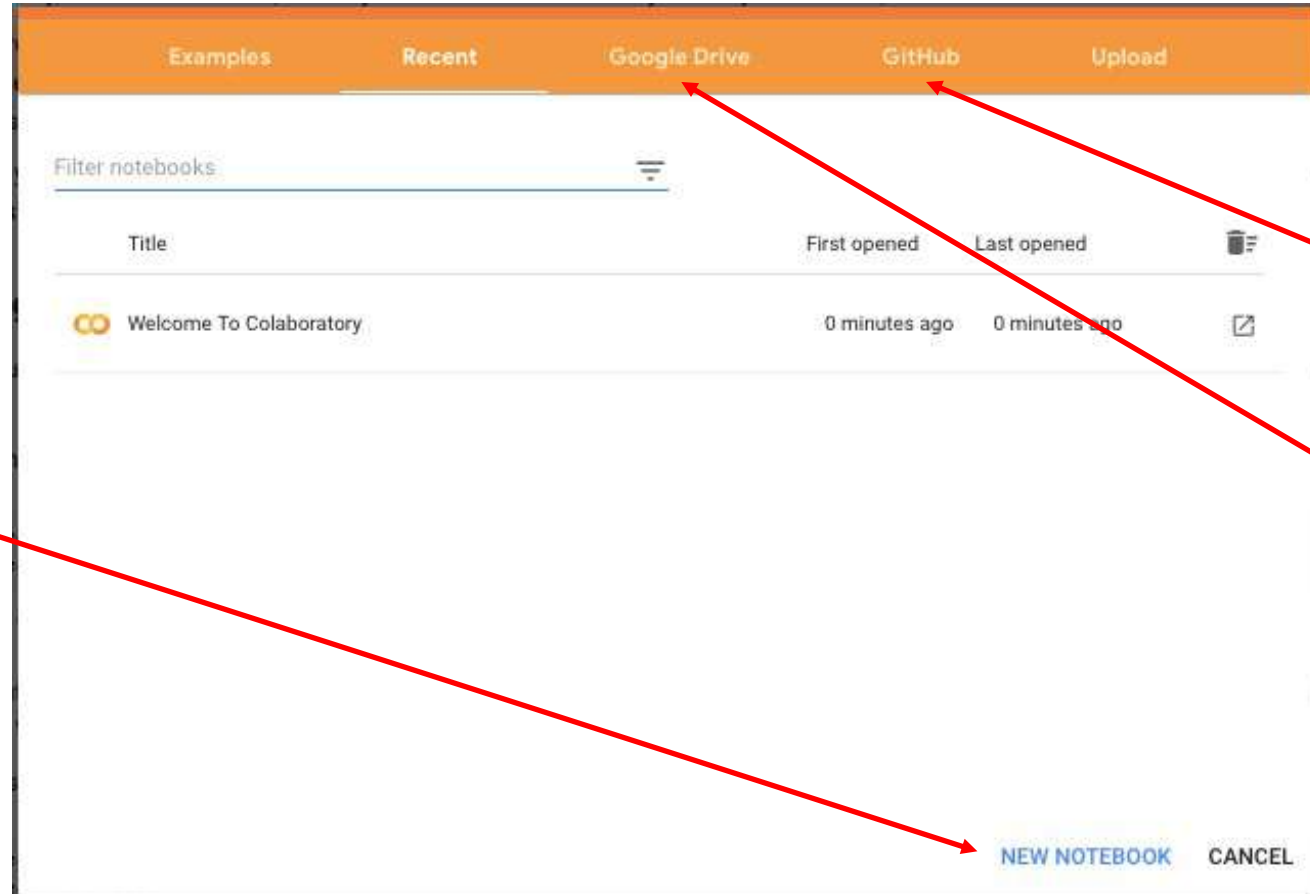


Figure 1.17: [Python] A parametric plot in 3D.

Google Colab: Start Session 3



Click here to open a new Python 3 ipynb notebook

GitHub

Google Drive

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

Google Colab: Untitled0.ipynb Notebook

The screenshot shows the Google Colab interface for a notebook titled 'Untitled0.ipynb'. The top menu bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help', with a status indicator 'All changes saved'. On the right, there are buttons for 'Comment', 'Share', and a settings gear, along with a Google account icon. Below the menu, a toolbar contains '+ Code' and '+ Text' buttons, a 'Connect' dropdown menu, and an 'Editing' mode selector. A code editor area is visible with a cursor. Red arrows point from callout boxes to these elements: one to the '+ Code' button, one to the '+ Text' button, one to the 'Untitled0.ipynb' title, and one to the 'Connect' dropdown. A fifth callout box provides a detailed explanation of the 'Connect' option.

+ Code + Text

Connect Editing

Comment Share

File Edit View Insert Runtime Tools Help All changes saved

Untitled0.ipynb

Add Code cell

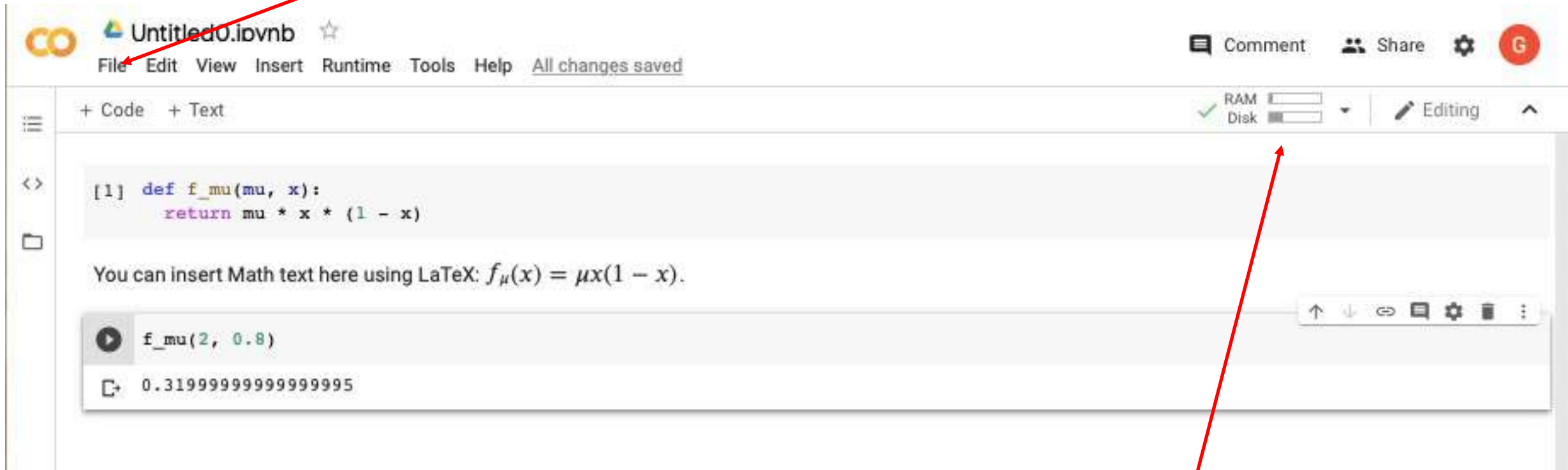
Add Text cell

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



The screenshot displays the Google Colab interface for a notebook titled 'Untitled0.ipynb'. The top menu bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help', with a status indicator 'All changes saved'. On the right, there are buttons for 'Comment', 'Share', and a settings gear. Below the menu, a toolbar shows '+ Code' and '+ Text' options. The main editor area contains a code cell with the following Python code:

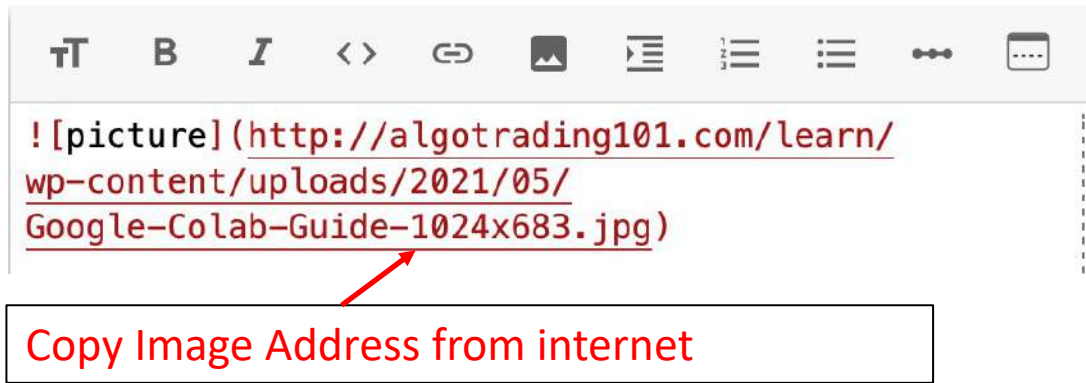
```
[1] def f_mu(mu, x):  
    return mu * x * (1 - x)
```

Below the code cell, a text prompt reads: 'You can insert Math text here using LaTeX: $f_{\mu}(x) = \mu x(1 - x)$.' Below this is a runtime cell showing the execution of `f_mu(2, 0.8)` with the output `0.31999999999999995`. On the right side of the interface, there are sliders for 'RAM' and 'Disk' usage, and a status indicator 'Editing'. A red arrow points from the 'File' menu to the 'Save the file to Google Drive or straight to GitHub' text box. Another red arrow points from the 'Connect to hosted runtime. You can use CPU, GPU or TPU' text box to the 'Runtime' menu.

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:



Loading a figure from your computer:

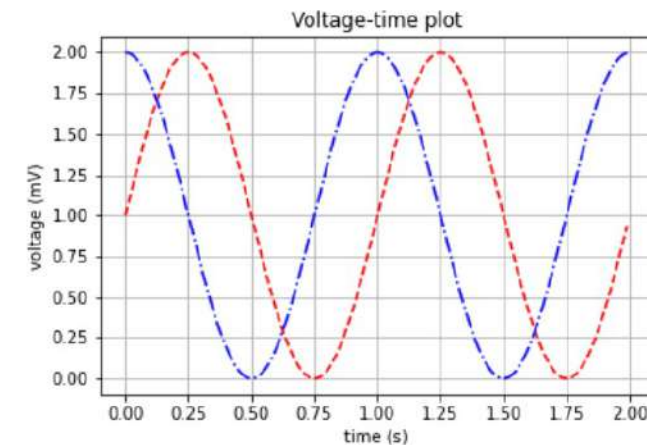
```
[1] from google.colab import files
    from IPython.display import Image
```

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

Choose Files no files selected

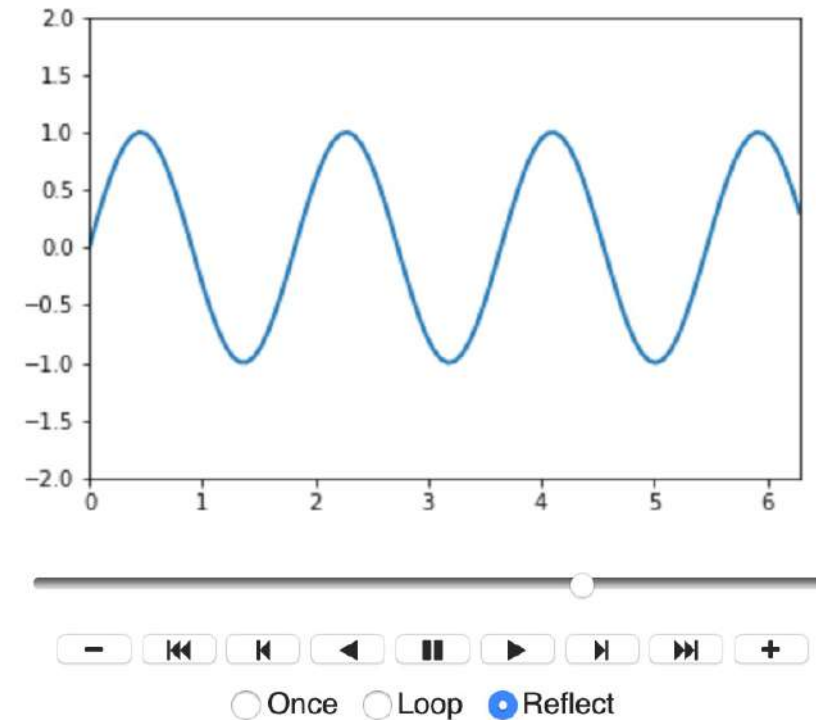
Cancel upload

```
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 $0 \leq a \leq 50$?

Interactive Plots in Google Colab

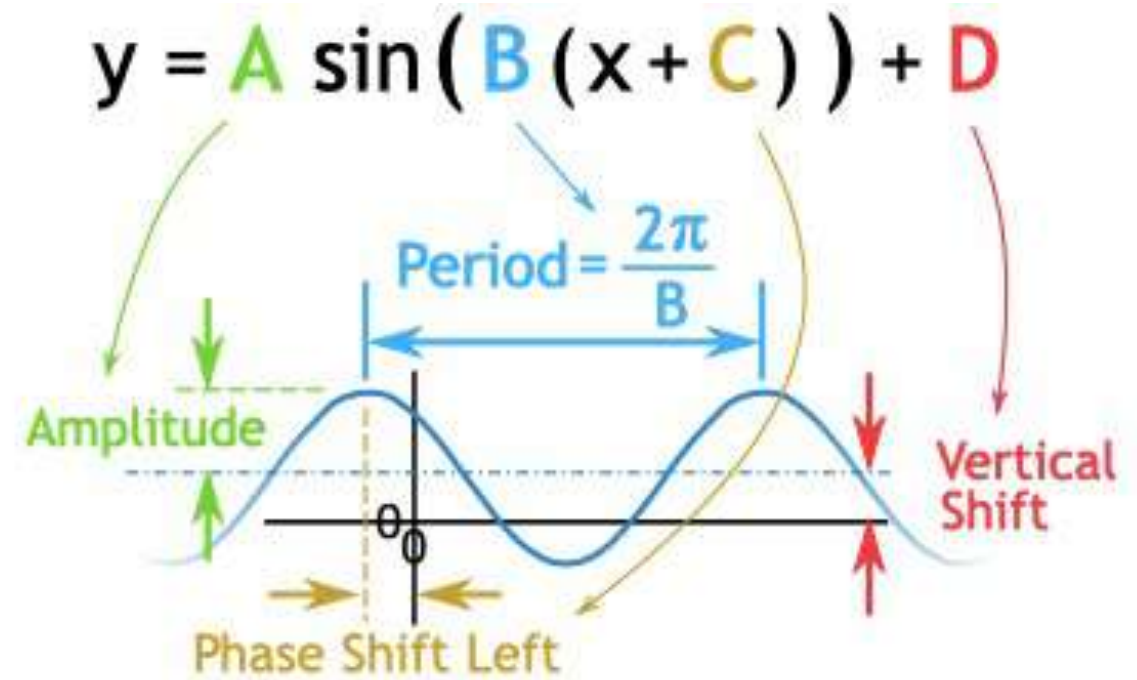
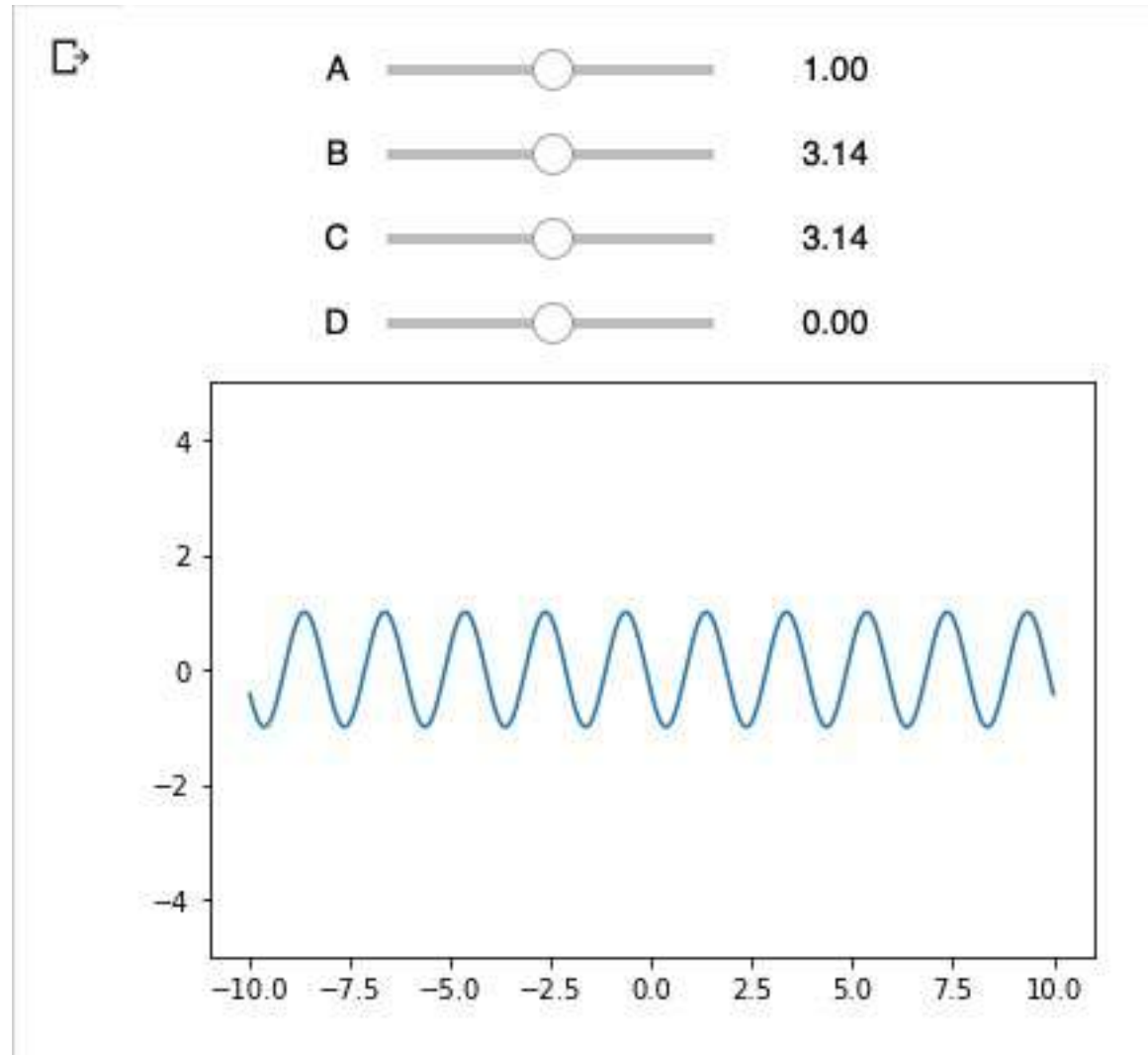
```
# Interactive plots with Python.
from __future__ import print_function
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets

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

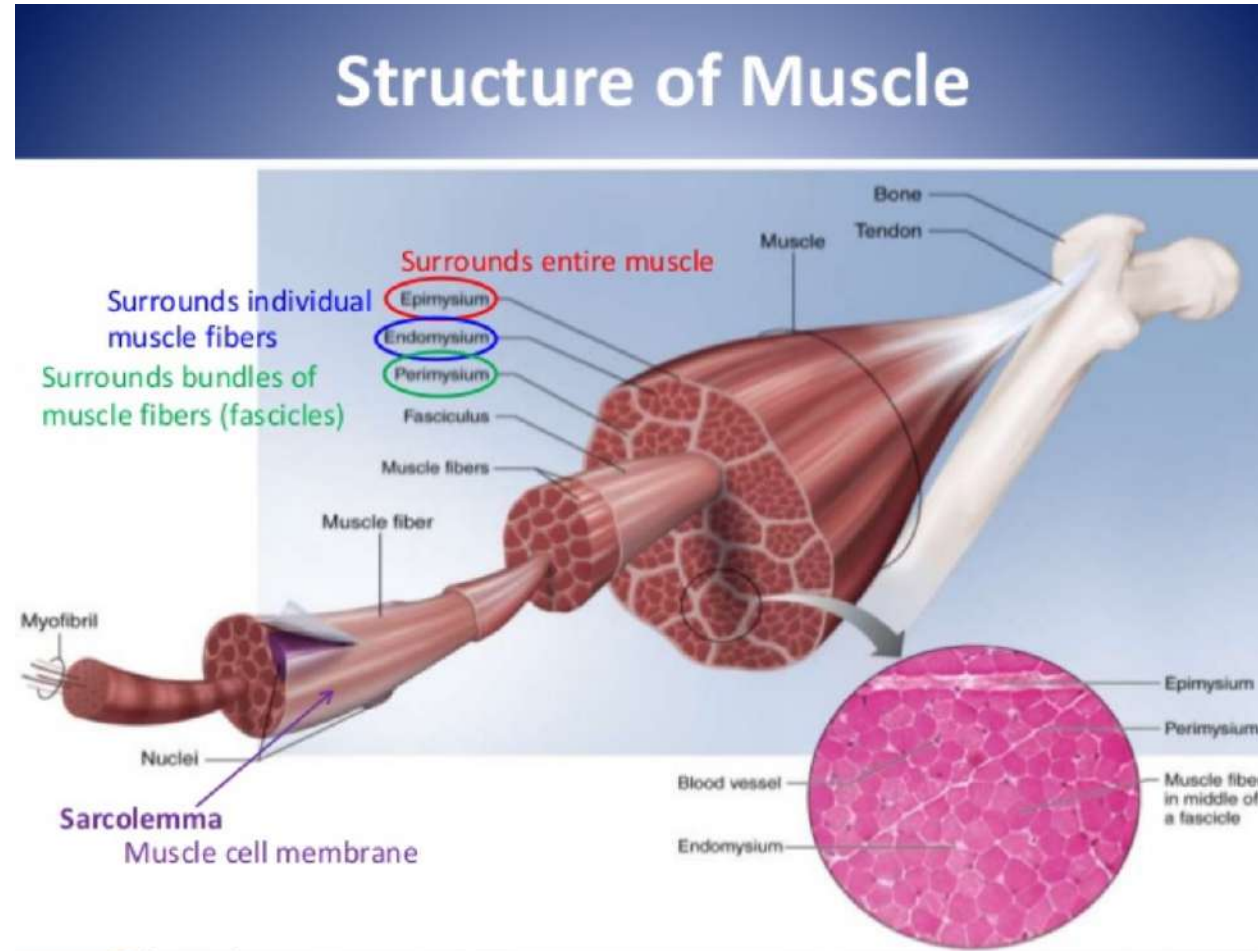
def f(A, B, C, D):
    plt.figure(2)
    x = np.linspace(-10, 10, num=1000)
    plt.plot(x, A * np.sin(B * (x + C)) + D)
    plt.ylim(-5, 5)
    plt.show()

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



Biological Models: Muscle Model: Start Session 4



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)

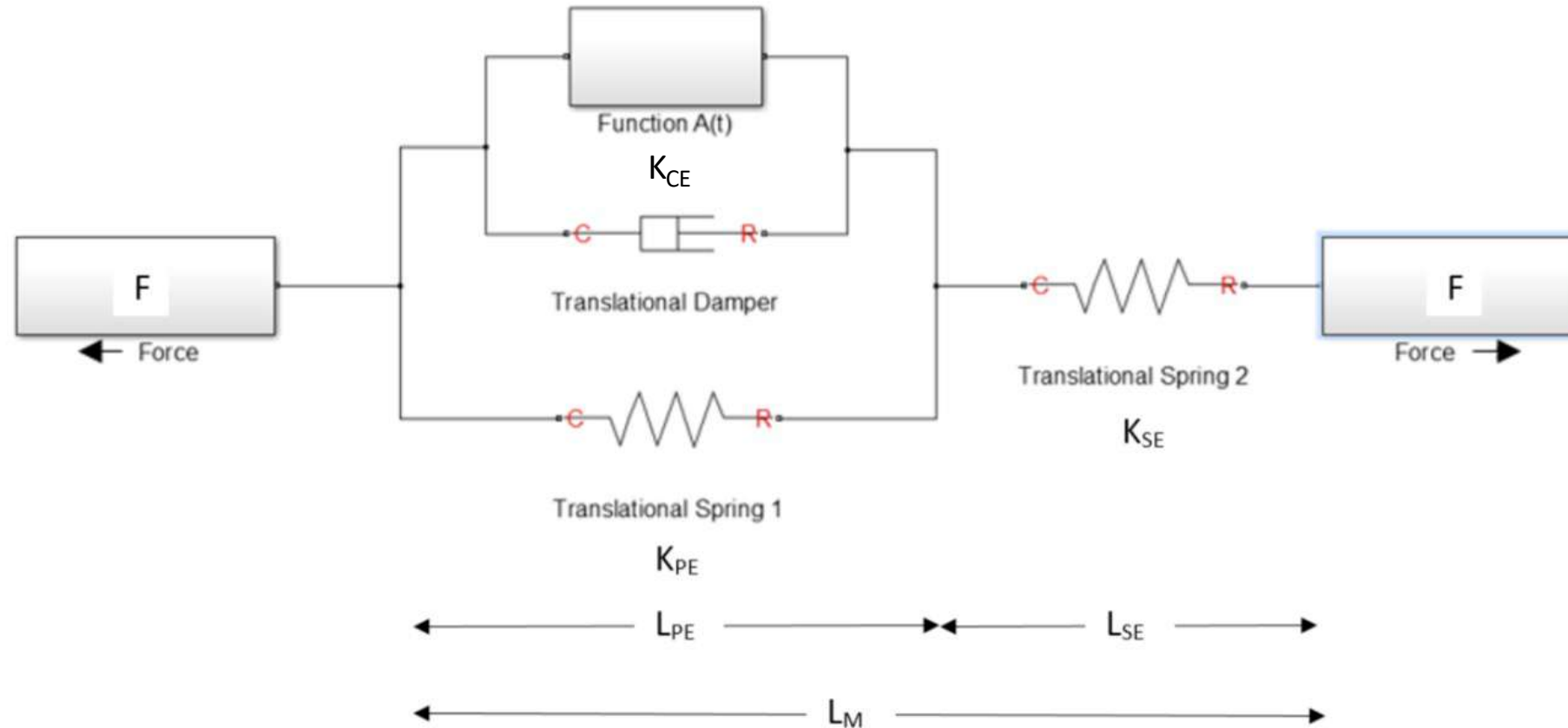
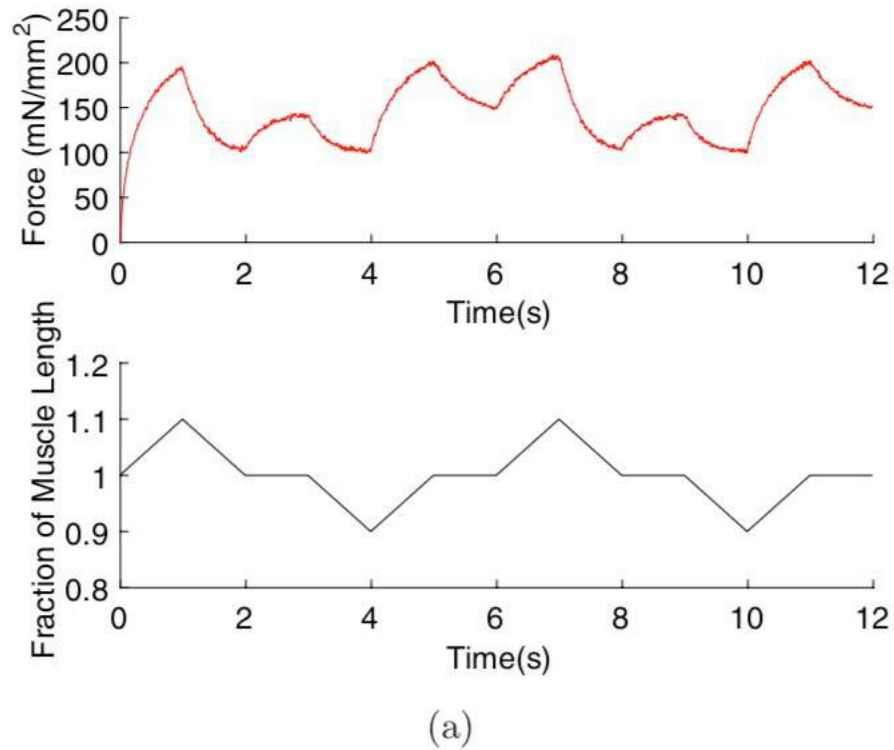


Fig. 18. A Simscape simulation of the Hill muscle model comprising of a series element (SE), a contractile element (CE) and a parallel element (PE).

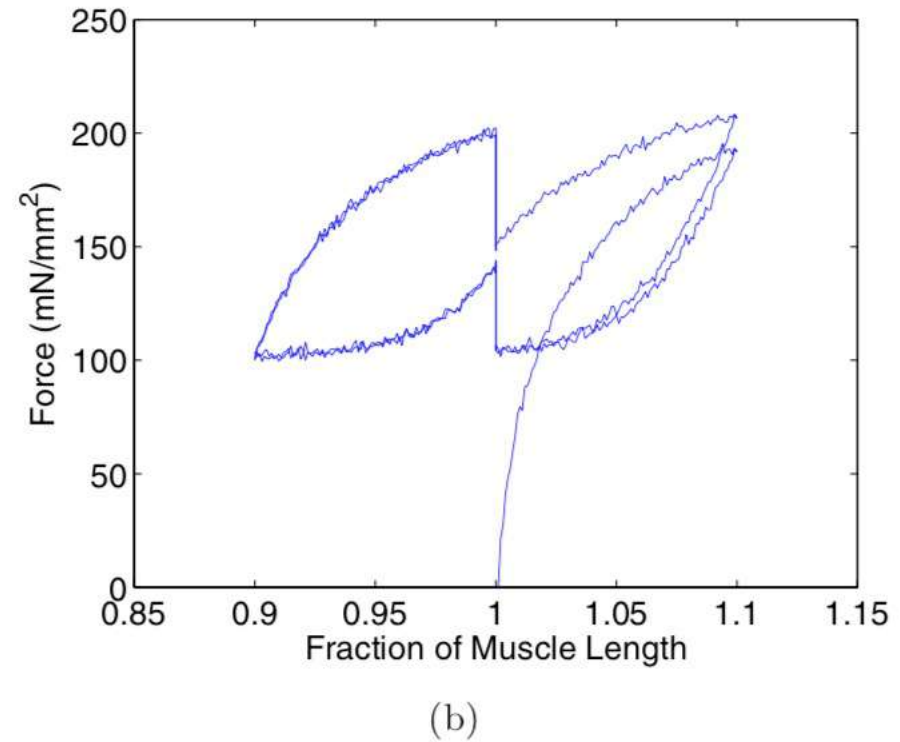
Muscle Model (Python Program of Hill Model)

```
1  # Muscle Hill model.
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # From Hill's paper.
6  Length, a, b = 1200, 380 * 0.098, 0.325
7  P0 = a / 0.257
8  vm = P0 * b / a
9  alpha = P0 / 0.1
10 LSE0 = 0.3
11 k = a / 25
12
13 t = [0 + 0.01 * i for i in range(1201)]      # Time
14
15 # Stretching, holding and contracting muscle.
16 A = [1.001 + 0.001 * i for i in range(100)] # Length A.
17 B = [1.099 - 0.001 * i for i in range(100)] # Length B.
18 C = np.ones(100).tolist()                   # Length C.
19 D = [0.999 - 0.001 * i for i in range(100)]
20 E = [0.901 + 0.001 * i for i in range(100)]
21 F = np.ones(100).tolist()
22 G = [1.001 + 0.001 * i for i in range(100)]
23 H = [1.099 - 0.001 * i for i in range(100)]
24 HH = np.ones(100).tolist()
25 J = [0.999 - 0.001 * i for i in range(100)]
26 K = [0.901 + 0.001 * i for i in range(100)]
27 KK = np.ones(101).tolist()
28 L = A+B+C+D+E+F+G+H+HH+J+K+KK
29
30 # Muscle model continued...
31 LSE = np.zeros(1200).tolist()
32 LCE = np.zeros(1200).tolist()
33 P = np.zeros(1201).tolist()
34
35 # Hill's differential equations.
36 for i in range(1200):
37     LSE[i] = 0.3 * P[i] / alpha
38     LCE[i] = L[i] - LSE[i]
39     dt = t[i + 1] - t[i]
40     dL = L[i + 1] - L[i]
41     dP = alpha * ((dL/dt) + b * ((P0 - P[i]) / (a + P[i]))) * dt
42     P[i + 1] = P[i] + dP
43
44 P = np.array(P)
45 PP = (P0 / 100) * np.random.randn(1201) # Add some noise.
46 P = P + PP
47 P = P.tolist()
48
49 plt.figure()
50 plt.plot(L, P) # Plot length v Force.
51 plt.xlabel('Fraction of Muscle Length mm', fontsize = 15)
52 plt.ylabel('Force ($mN / mm^2$)', fontsize = 15)
53 plt.tick_params(labelsize = 15)
```

Muscle Model (Lengthening and Shortening)

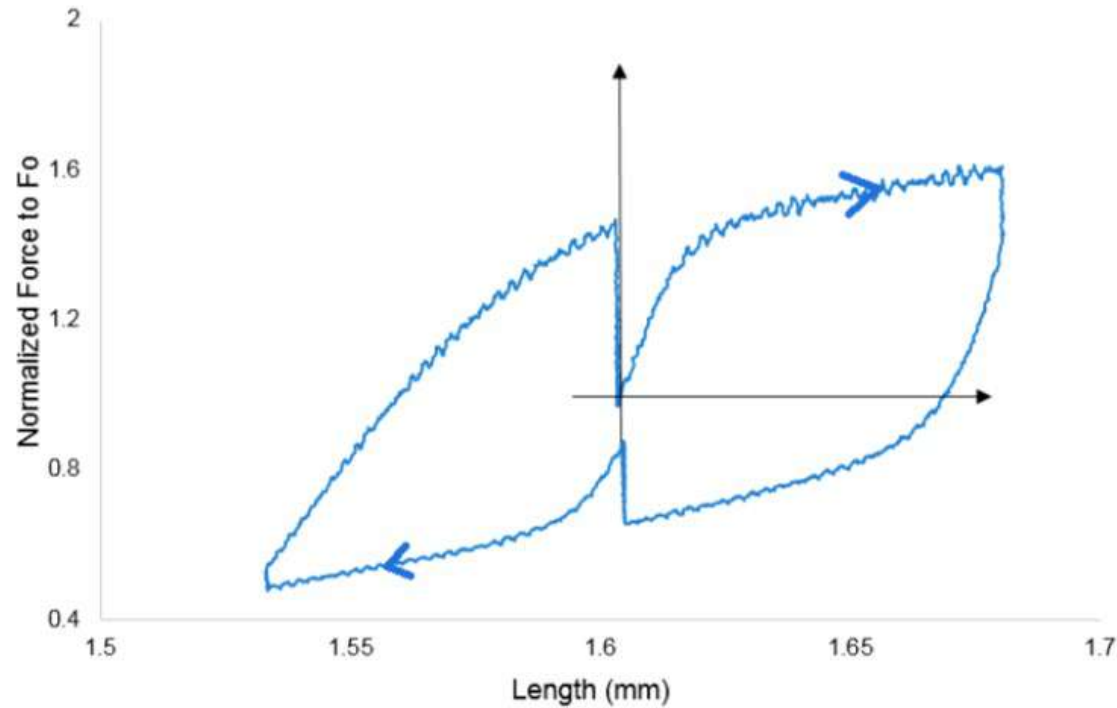


Lengthening and shortening with rests.

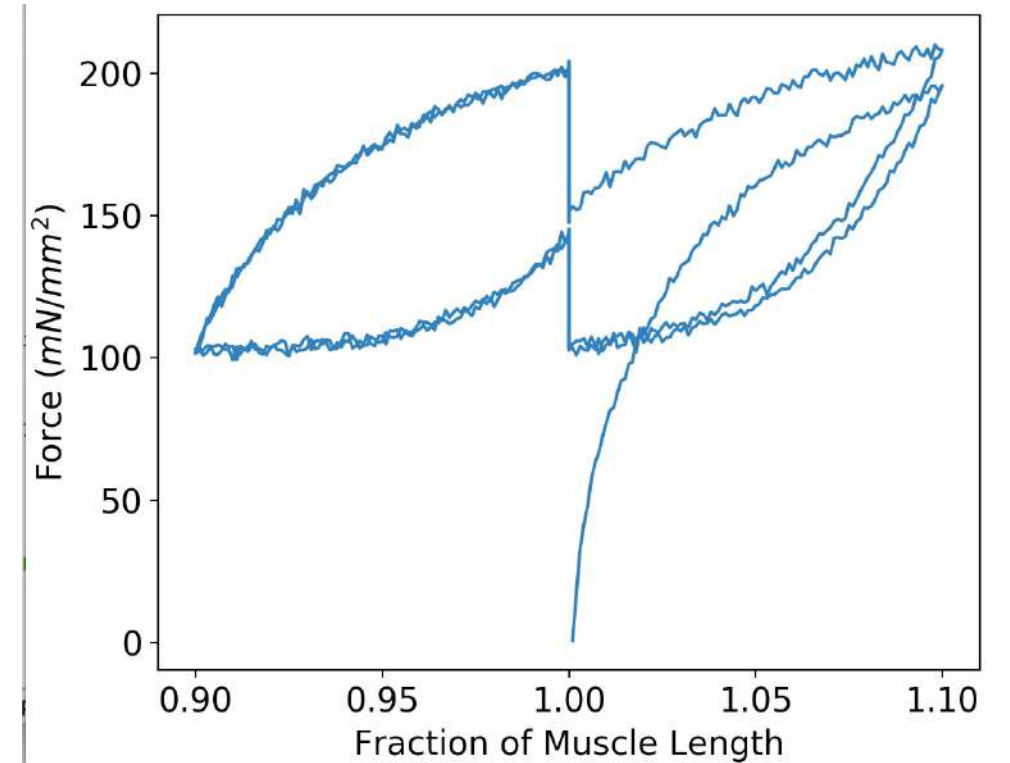


Hysteresis curves from modelling.

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.

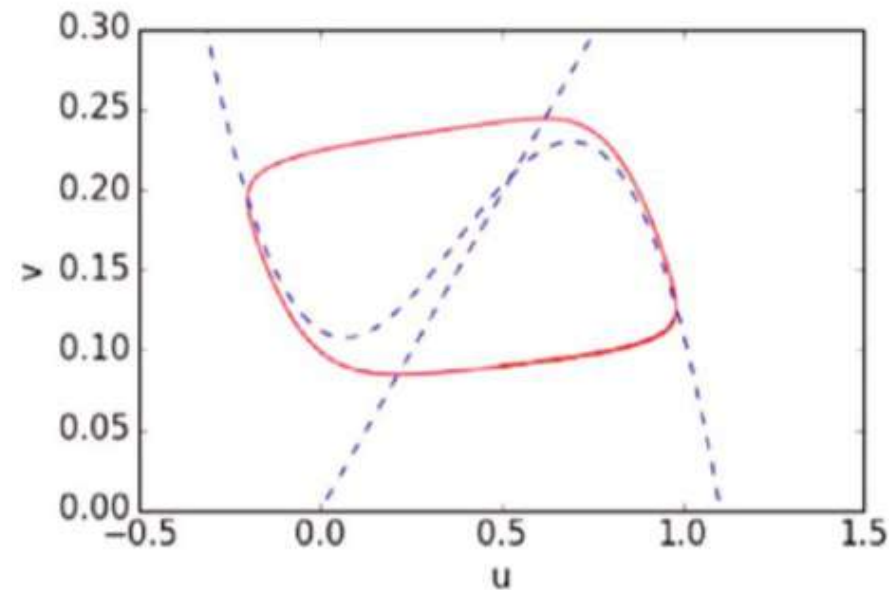
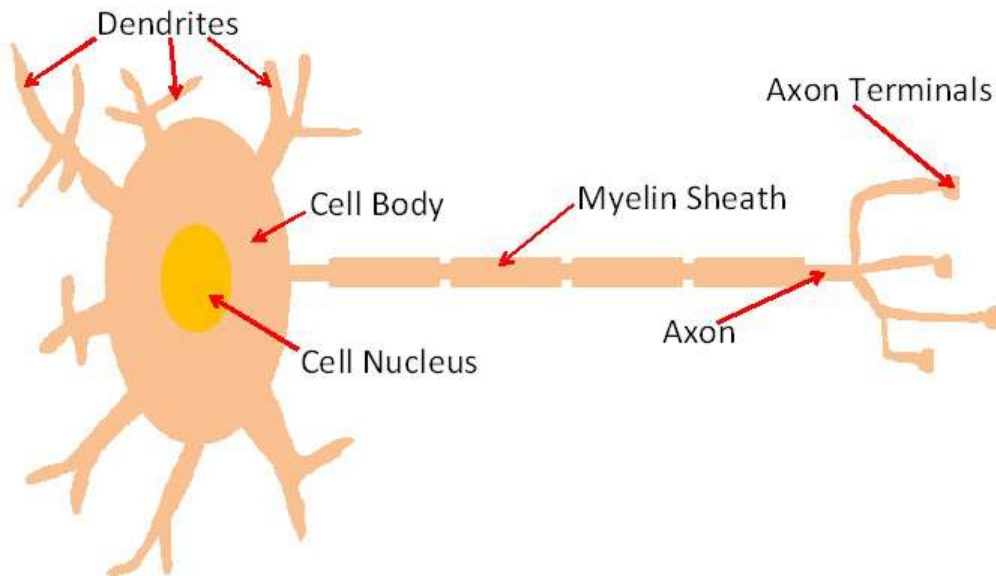


Figure 5.3: [Python] A limit cycle for the Fitzhugh-Nagumo oscillator. In this case, $\gamma = 2.54$, $\theta = 0.14$, $\omega = 0.112$, and $\epsilon = 0.01$. The blue dashed curves are the nullclines, where the trajectories cross horizontally and vertically.

A Simple Neuron Model: Fitzhugh-Nagumo Limit Cycle

```
Program_05a.py*
1 # Program 05a: Limit cycle for Fitzhugh-Nagumo.
2 # See Figure 5.3.
3
4 import matplotlib.pyplot as plt
5 import numpy as np
6 from scipy.integrate import odeint
7
8 theta, omega, gamma, epsilon = 0.14, 0.112, 2.54, 0.01
9 xmin, xmax, ymin, ymax = -0.5, 1.5, 0, 0.3
10 def dx_dt(x, t):
11     return [-x[0] * (x[0] - theta) * (x[0] - 1) - x[1] + omega,
12            epsilon * (x[0] - gamma * x[1])]
13 # Trajectories in forward time.
14 xs = odeint(dx_dt, [0.5, 0.09], np.linspace(0, 100, 1000))
15 plt.plot(xs[:, 0], xs[:, 1], 'r-')
16 # Label the axes and set font sizes.
17 plt.xlabel('u', fontsize=15)
18 plt.ylabel('v', fontsize=15)
19 plt.tick_params(labelsize=15)
20 plt.xlim(xmin, xmax)
21 plt.ylim(ymin, ymax);
22 # Plot the isoclines.
23 x=np.arange(xmin, xmax, 0.01)
24 plt.plot(x, x/gamma, 'b--', x, -x * (x - theta) * (x - 1) + omega, 'b--')
25 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.

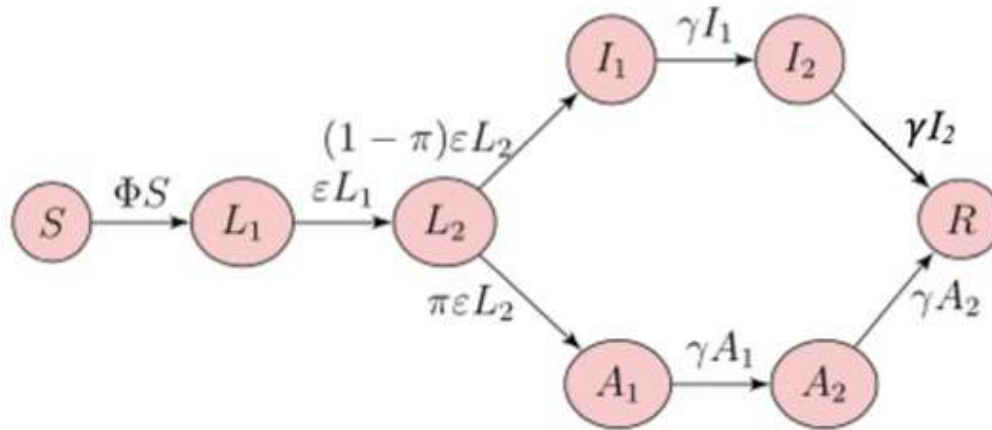
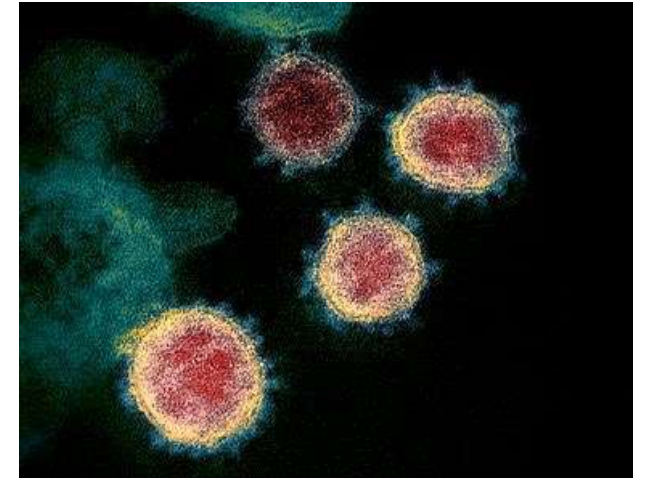


Figure 1: Compartmental model of the spread of coronavirus in Manchester. The acronym SLIAR stands for Susceptible, Latently Infected, symptomatic and Asymptomatic infectious and Removed individuals. In this case $\Phi = \beta (I_1 + I_2 + \xi (A_1 + A_2) + \eta L_2)$, is the force of infection.

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.



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

Compartmental Model Epidemiology

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

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

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

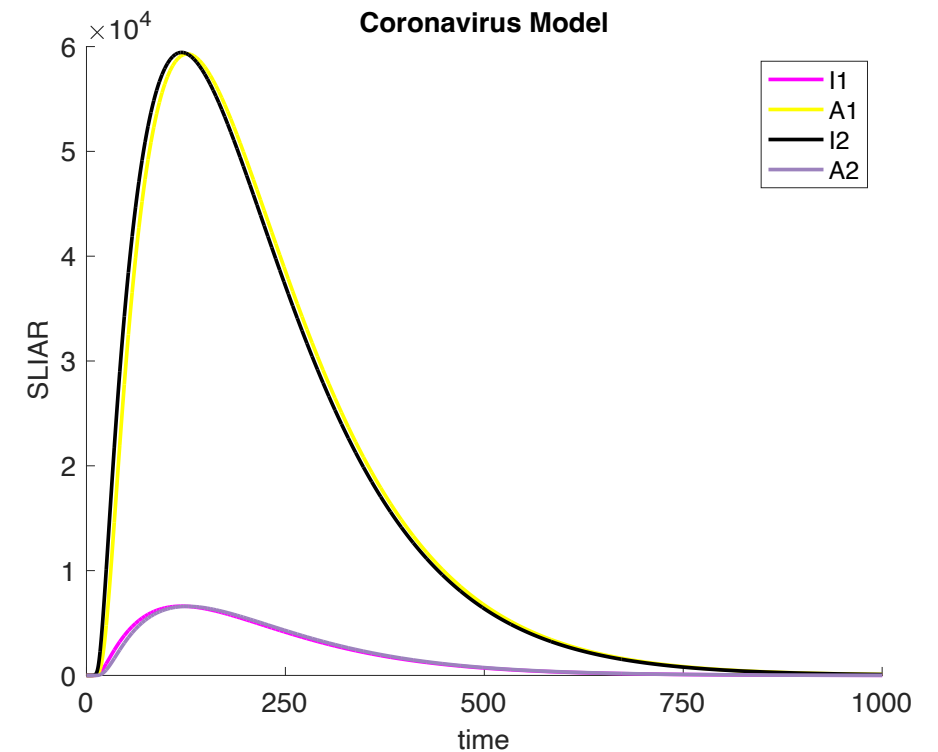
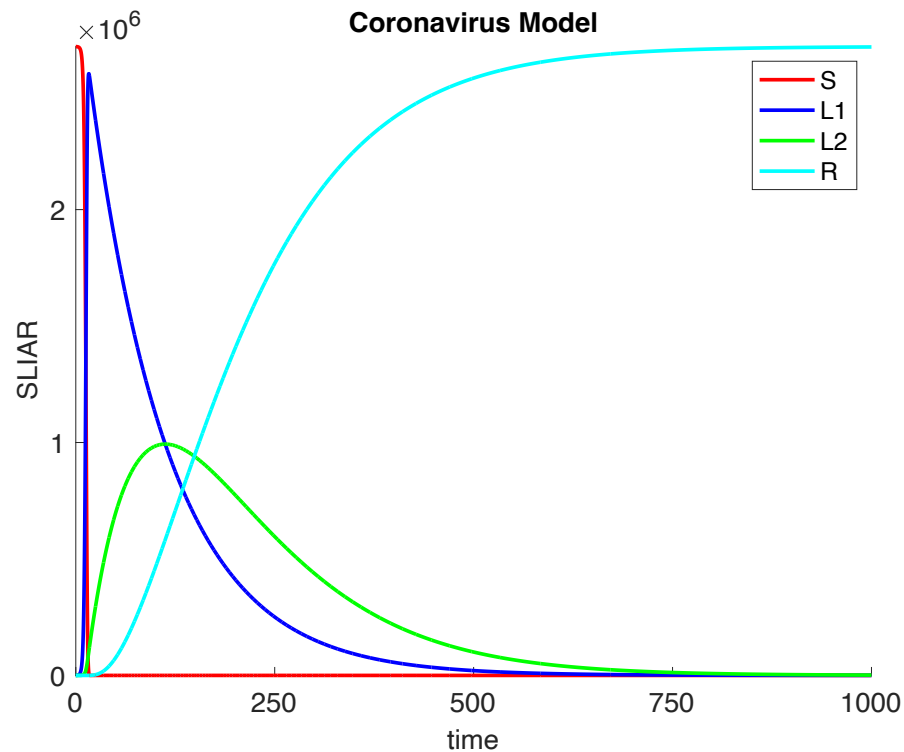
$$\frac{dI_1}{dt} = (1 - \pi)\varepsilon L_2 - \gamma I_1$$

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

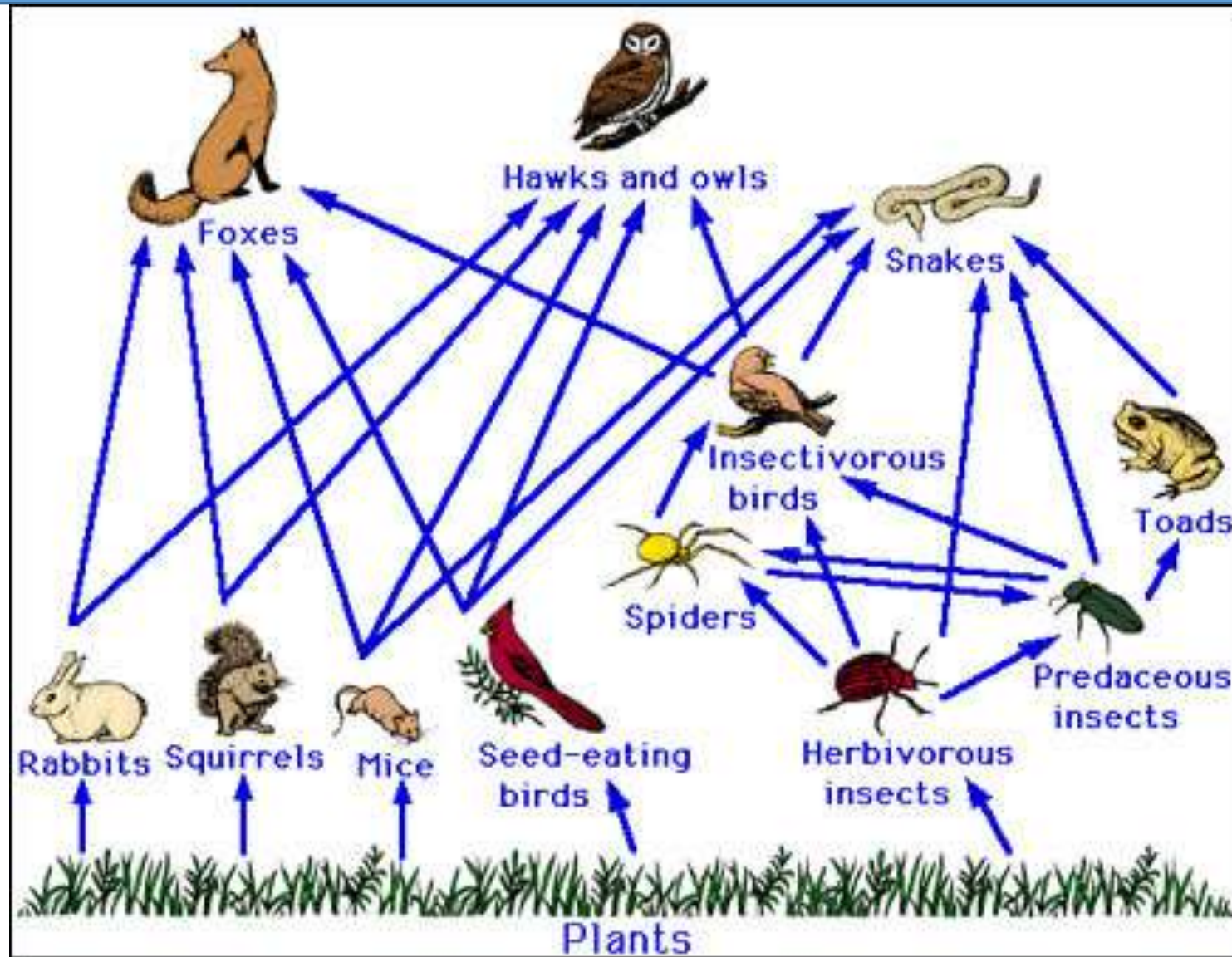
$$\frac{dA_1}{dt} = \pi\varepsilon L_2 - \gamma A_1$$

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

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



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.

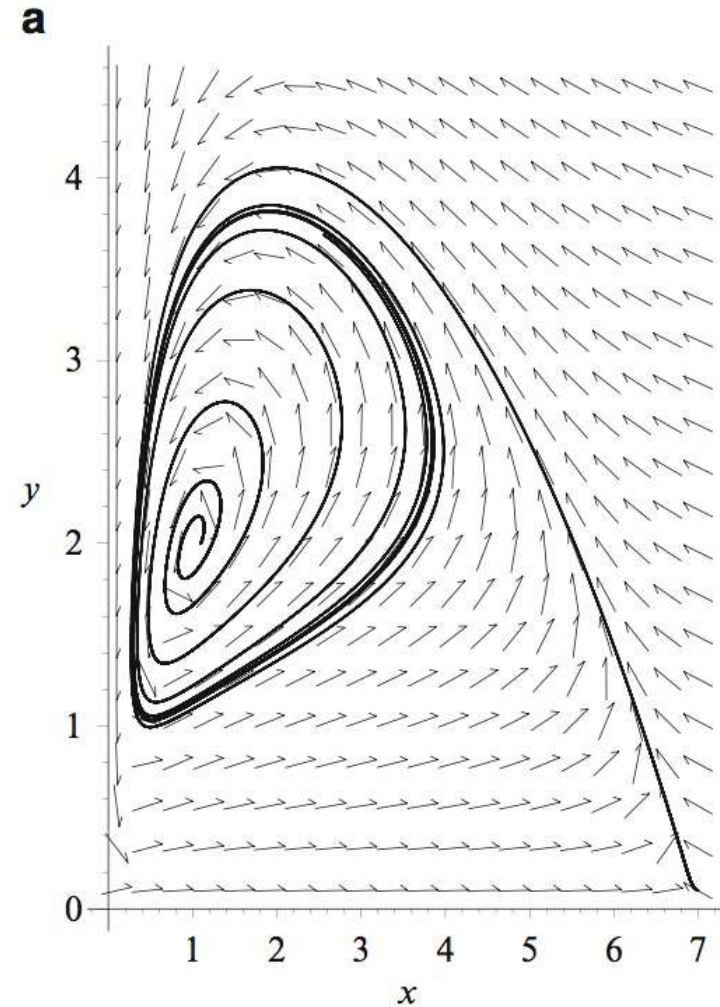
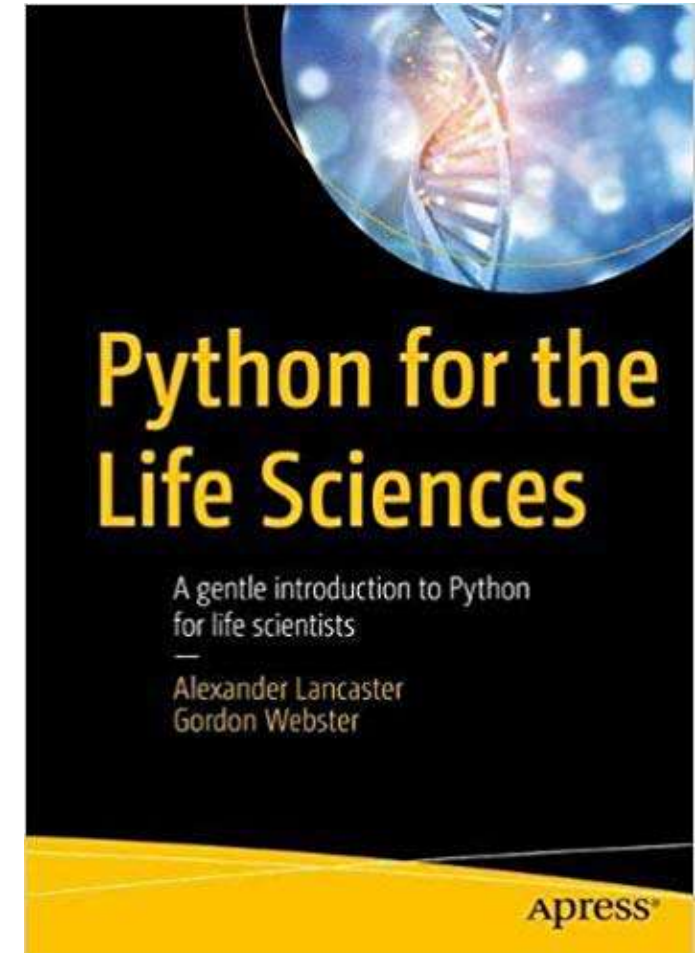


Fig. 4.5 A limit cycle of the Holling-Tanner Model

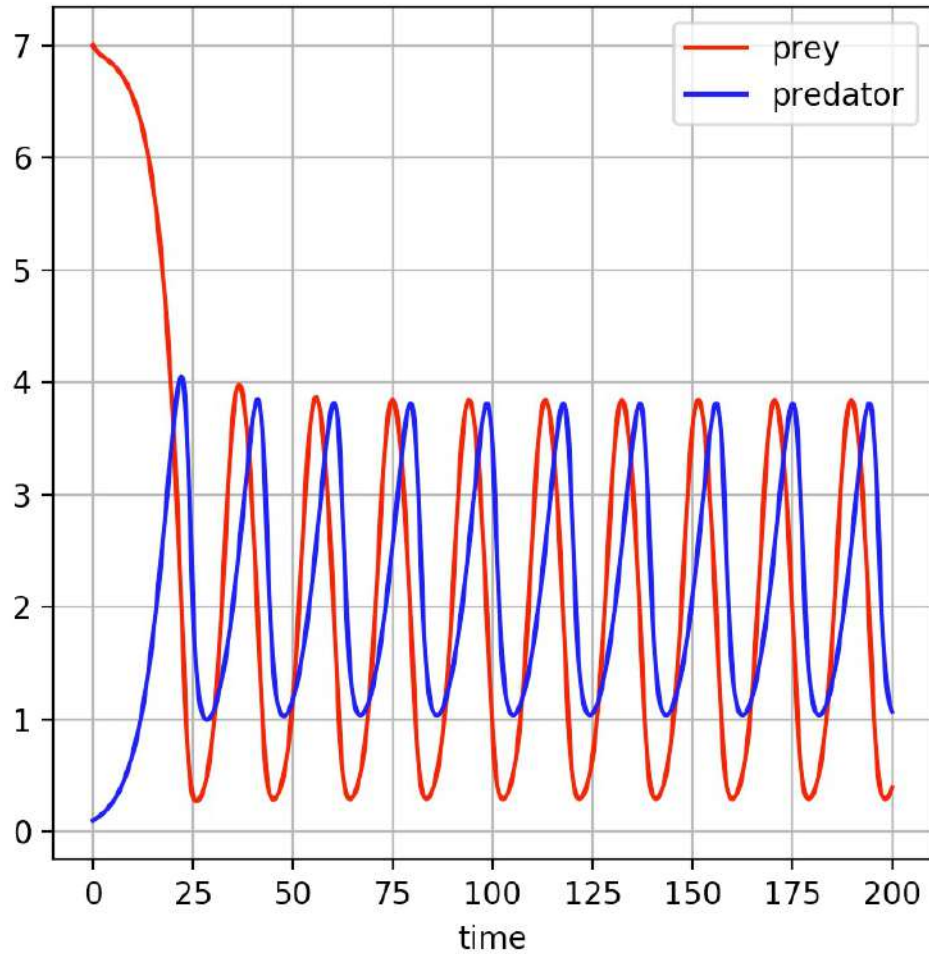
The Holling-Tanner Model (Python Program)

```
1 # Program 04a: Holling-Tanner model. See Figures 4.5 and 4.6.
2 # Time series and phase portrait for a predator-prey system.
3 import numpy as np
4 from scipy import integrate
5 import matplotlib.pyplot as plt
6
7 # The Holling-Tanner model.
8 def holling_tanner(X, t=0):
9     # here X[0] = x and X[1] = y
10    return np.array([X[0] * (1 - X[0]/7) - 6 * X[0] * X[1] / (7 + 7*X[0]),
11                    0.2 * X[1] * (1 - 0.5 * X[1] / X[0])])
12
13 t = np.linspace(0, 200, 1000)
14 # initial values: x0 = 7, y0 = 0.1
15 Sys0 = np.array([7, 0.1])
16
17 X, infodict = integrate.odeint(holling_tanner, Sys0, t, full_output=True)
18 x, y = X.T
19
20 fig = plt.figure(figsize=(15, 5))
21 fig.subplots_adjust(wspace=0.5, hspace=0.3)
22 ax1 = fig.add_subplot(1, 2, 1)
23 ax2 = fig.add_subplot(1, 2, 2)
24
25 ax1.plot(t, x, 'r-', label='prey')
26 ax1.plot(t, y, 'b-', label='predator')
27 ax1.set_title('Time Series')
28
29 ax1.set_xlabel('time')
30 ax1.grid()
31 ax1.legend(loc='best')
32
33 ax2.plot(x, y, color='blue')
34 ax2.set_xlabel('x')
35 ax2.set_ylabel('y')
36 ax2.set_title('Phase portrait')
37 ax2.grid()
38
39 plt.show()
```

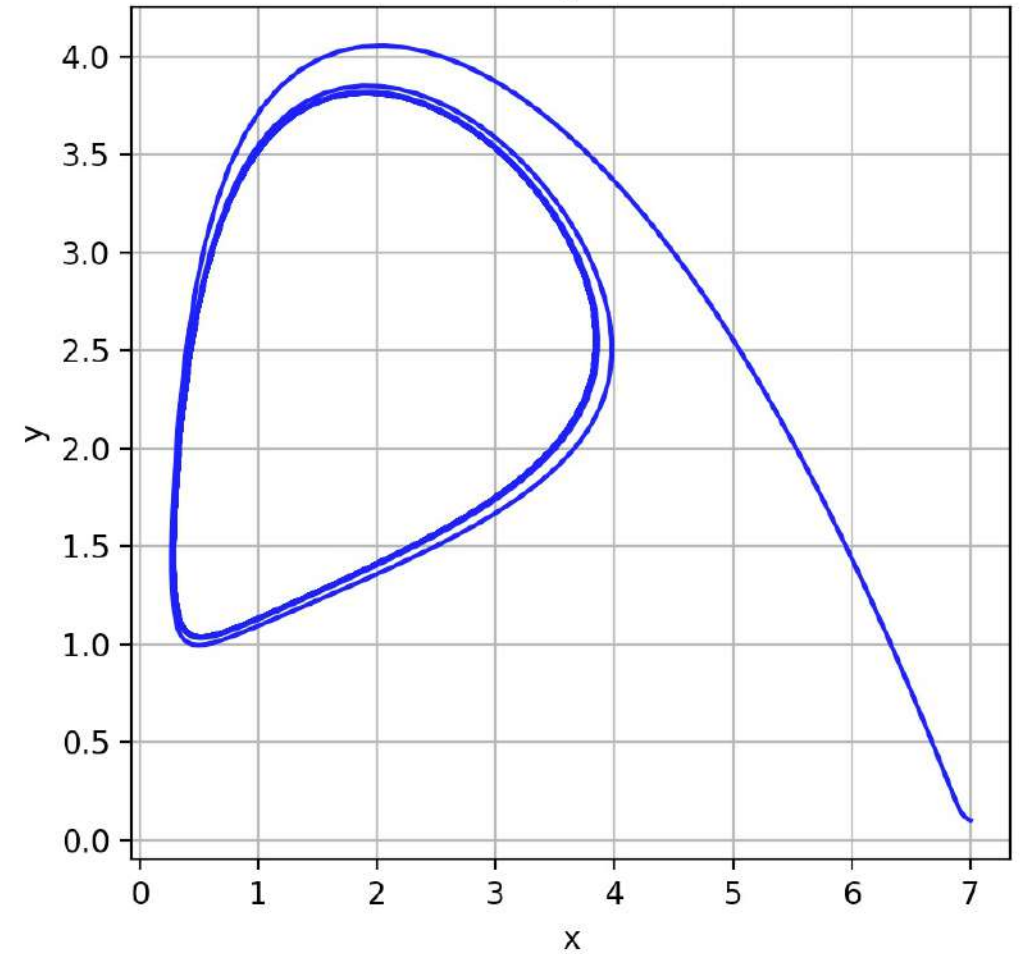


The Holling-Tanner Model

Time Series



Phase portrait



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/>

