4-Day Hands-on Workshop on:

Python for Scientific Computing
and
TensorFlow for Artificial Intelligence

By Dr Stephen Lynch NATIONAL TEACHING FELLOW FIMA SFHEA

Inventor of BINARY OSCILLATOR COMPUTING

Author of PYTHON™, MATLAB®, MAPLE™ AND MATHEMATICA® BOOKS

STEM Ambassador, Public Engagement Champion and Speaker for Schools

s.lynch@mmu.ac.uk

https://www.mmu.ac.uk/computing-and-maths/staff/profile/dr-stephen-lynch

Schedule (Day 2)

Day 2			
Jupyter and Colab Notebooks	10am-11am	Economics	2pm-3pm
Biology and Chemistry	11am-12pm	Engineering	3pm-4pm
Data Science	12pm-1pm		

Download all files from GitHub:

https://github.com/proflynch/CRC-Press/

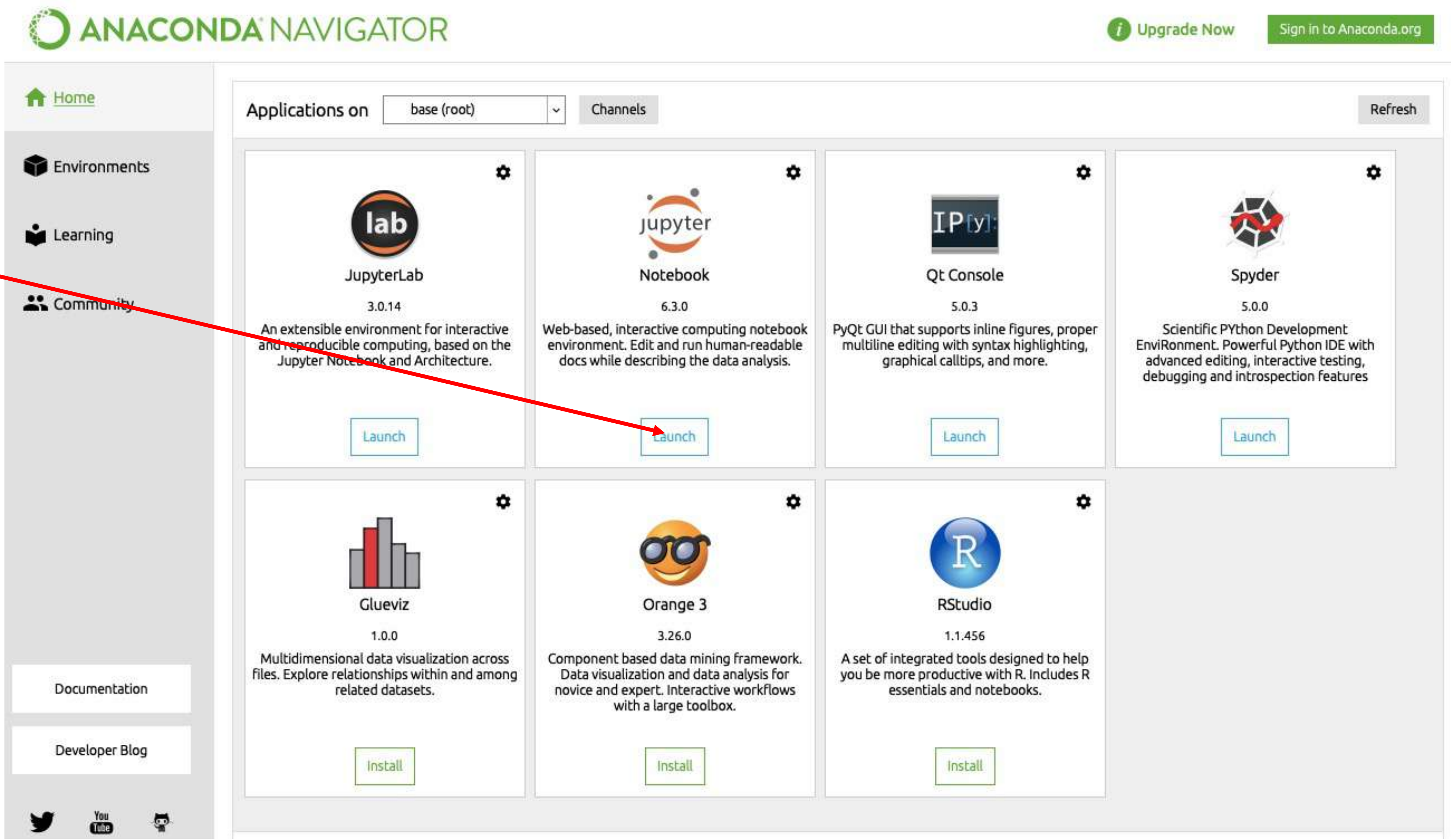
Solutions to the Exercises in Section 2:

https://drstephenlynch.github.io/webpages/Solutions_Section_2.html

python™

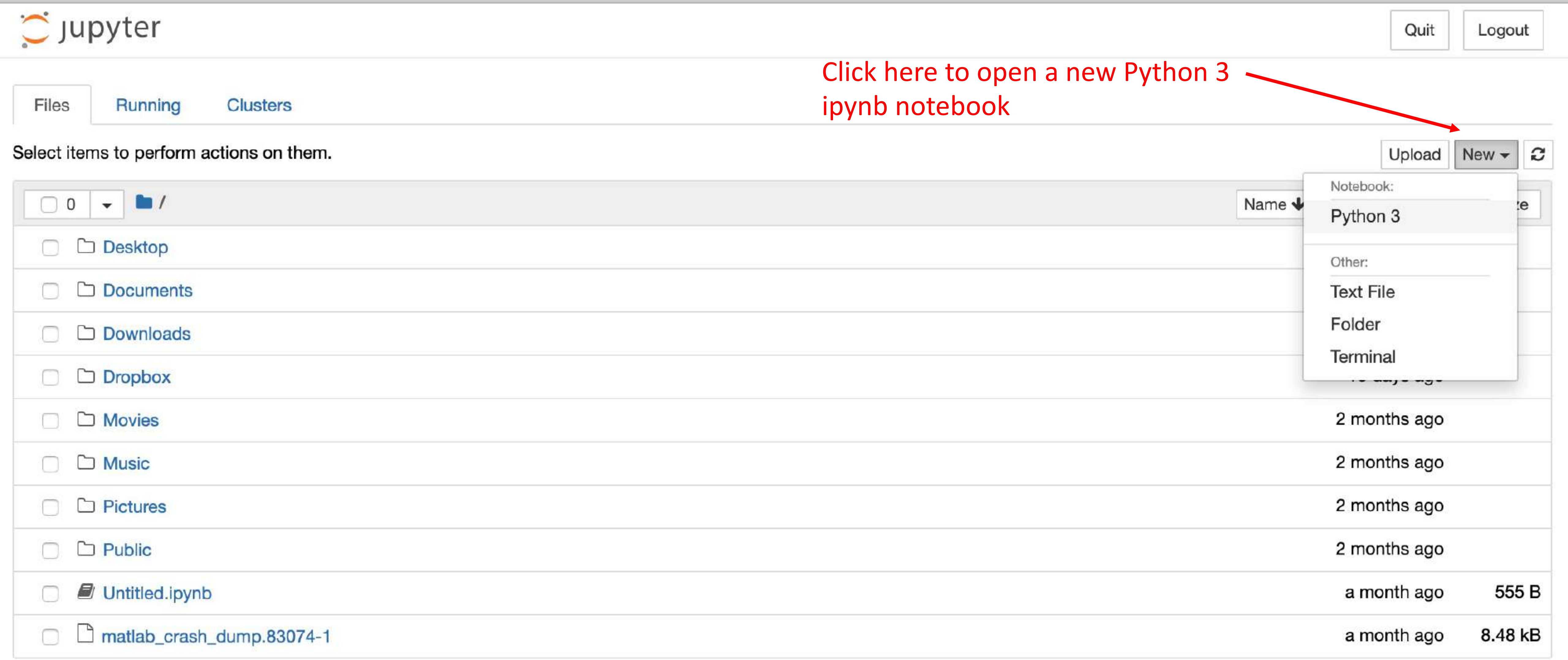
Anaconda: Launch a Jupyter Notebook: Start Session 1

Click here to open a new Python 3 ipynb notebook



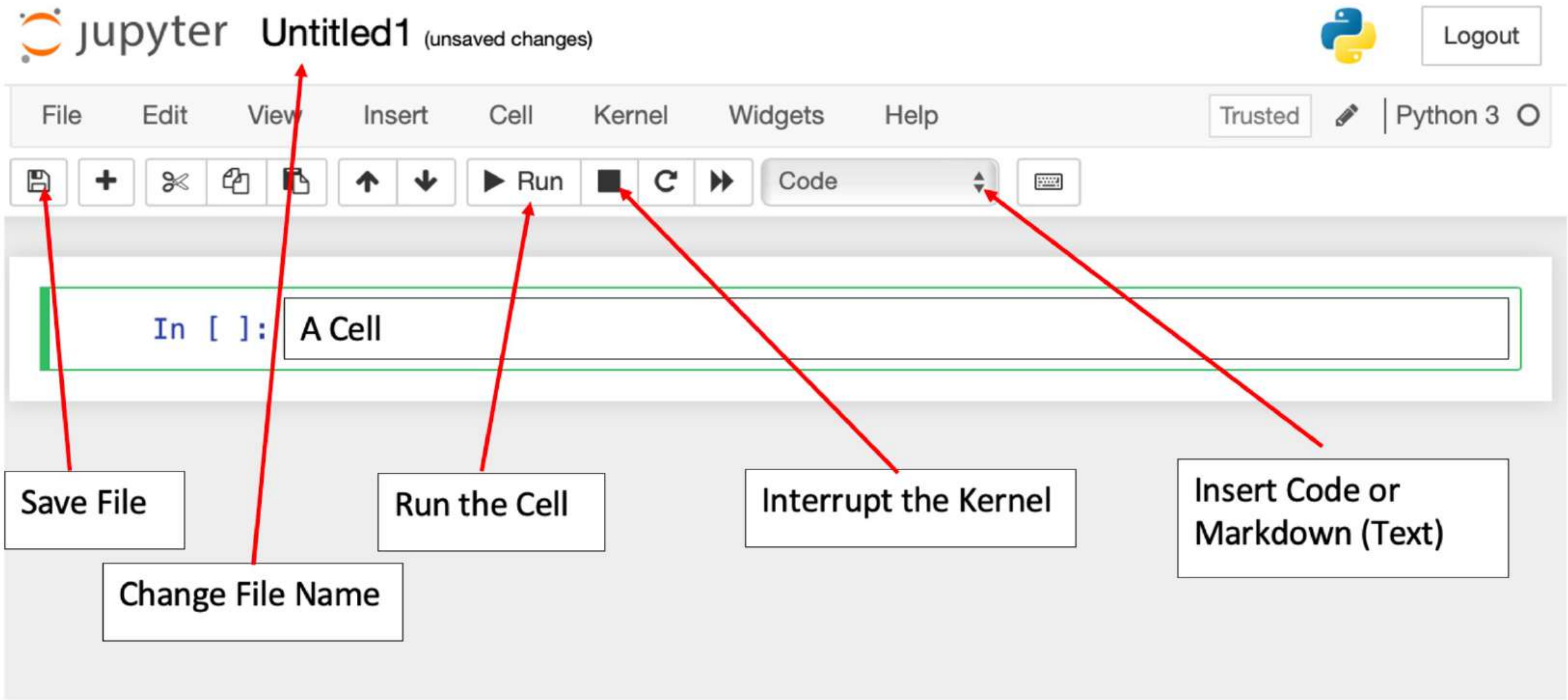
Manchester Metropolitan University

Anaconda: Launch a Jupyter Notebook

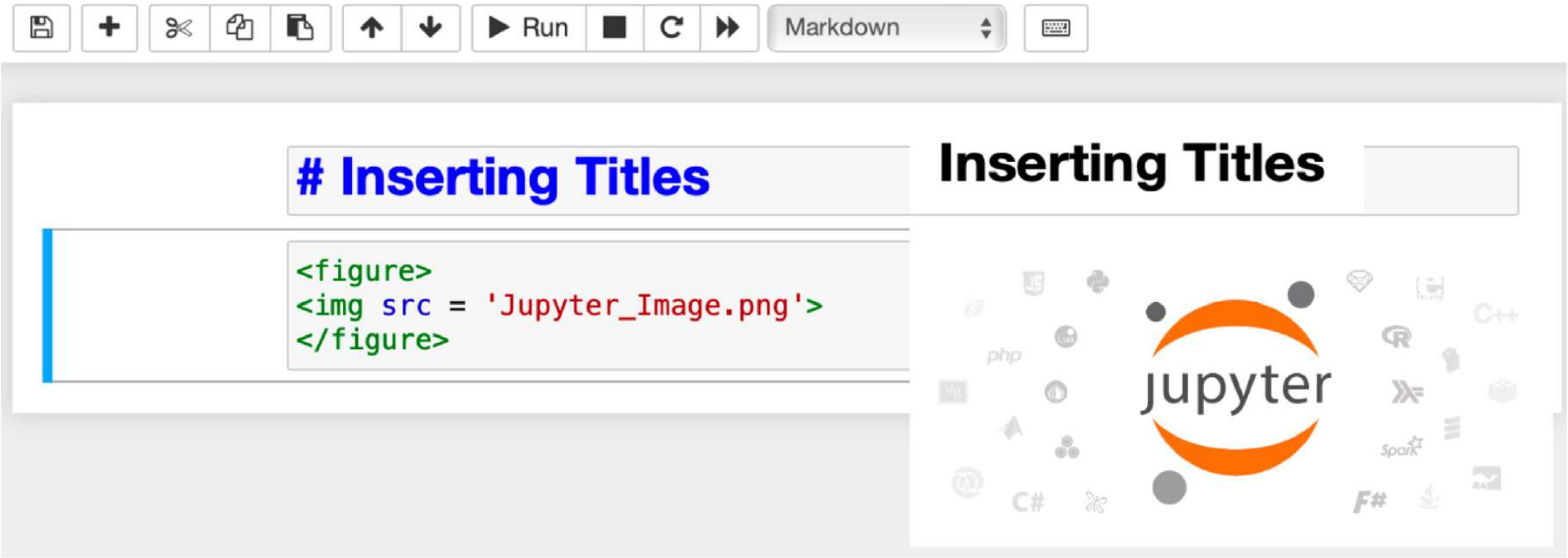


Click here to open a new Python 3 ipynb notebook

Jupyter Notebook



Jupyter Notebook: Inserting Titles and Figures



You can **File** > **Download as** > Notebook (.pynb) or Webpage (.html)
Save (ipynb) file in a folder with figure and **Open** using Jupyter.

Simple Programming with Jupyter Notebooks (Solving ODEs)

```
Solve the 1st order ODE:

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

Solve the 2nd order ODE:

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

Solve the 1st order ODE:

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

Solve the 2nd order ODE:

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

In [1]:

```
# Solve the 1st order ODE.
from sympy import *
t = symbols('t')
x = symbols('x' , cls = Function)
ODE1 = Eq(x(t).diff(t) , 1 - x(t))   # dx/dt=1-x.
sol1 = dsolve(ODE1 , x(t))
print(sol1)
```

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

https://oeis.org/wiki/List_of_LaTeX_mathematical_symbols

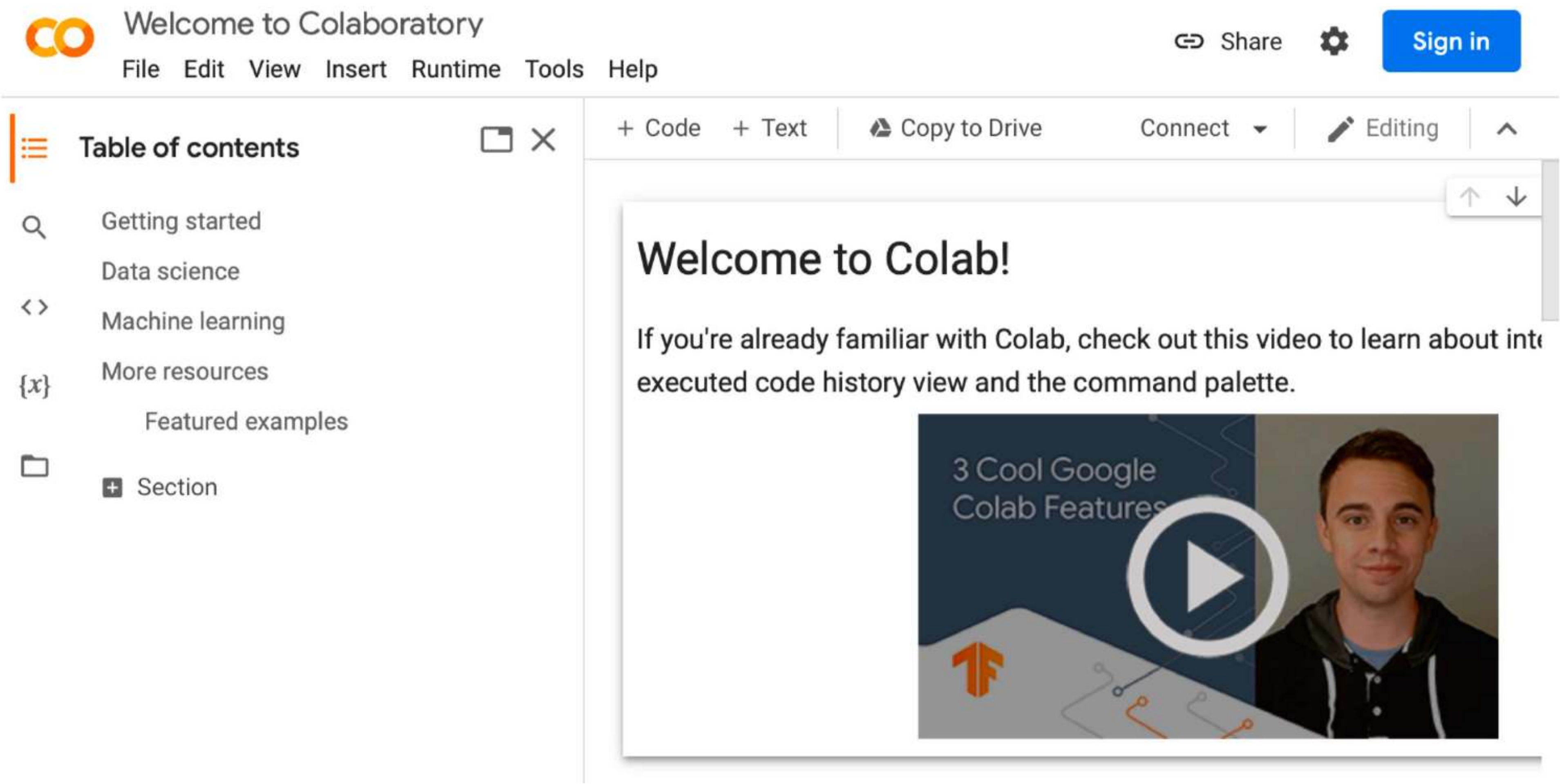
Jupyter Notebooks: LaTeX Symbols

Table 3.1 Table of popular LaTeX symbols for jupyter Notebooks.

Greek Letters	LaTeX	Mathematics	LaTeX
α	\alpha	$\frac{dx}{dt}$	\frac{dx}{dt}
β	\beta	$\dot{x}$	\dot{ x }
γ, Γ	\gamma, \Gamma	$\ddot{x}$	\ddot{x}
δ	\delta	$\sin(x)$	\sin(x)
ϵ	\epsilon	$\cos(x)$	\cos(x)
θ	\theta	$\leq$	\leq
λ, Λ	\lambda, \Lambda	$\geq$	\geq
μ	\mu	x^2	x^2
σ, Σ	\sigma \Sigma	$\in$	\in
τ	\tau	$\pm$	\pm
ϕ	\phi	$\rightarrow$	\rightarrow
ω, Ω	\omega, \Omega	$\int$	\int

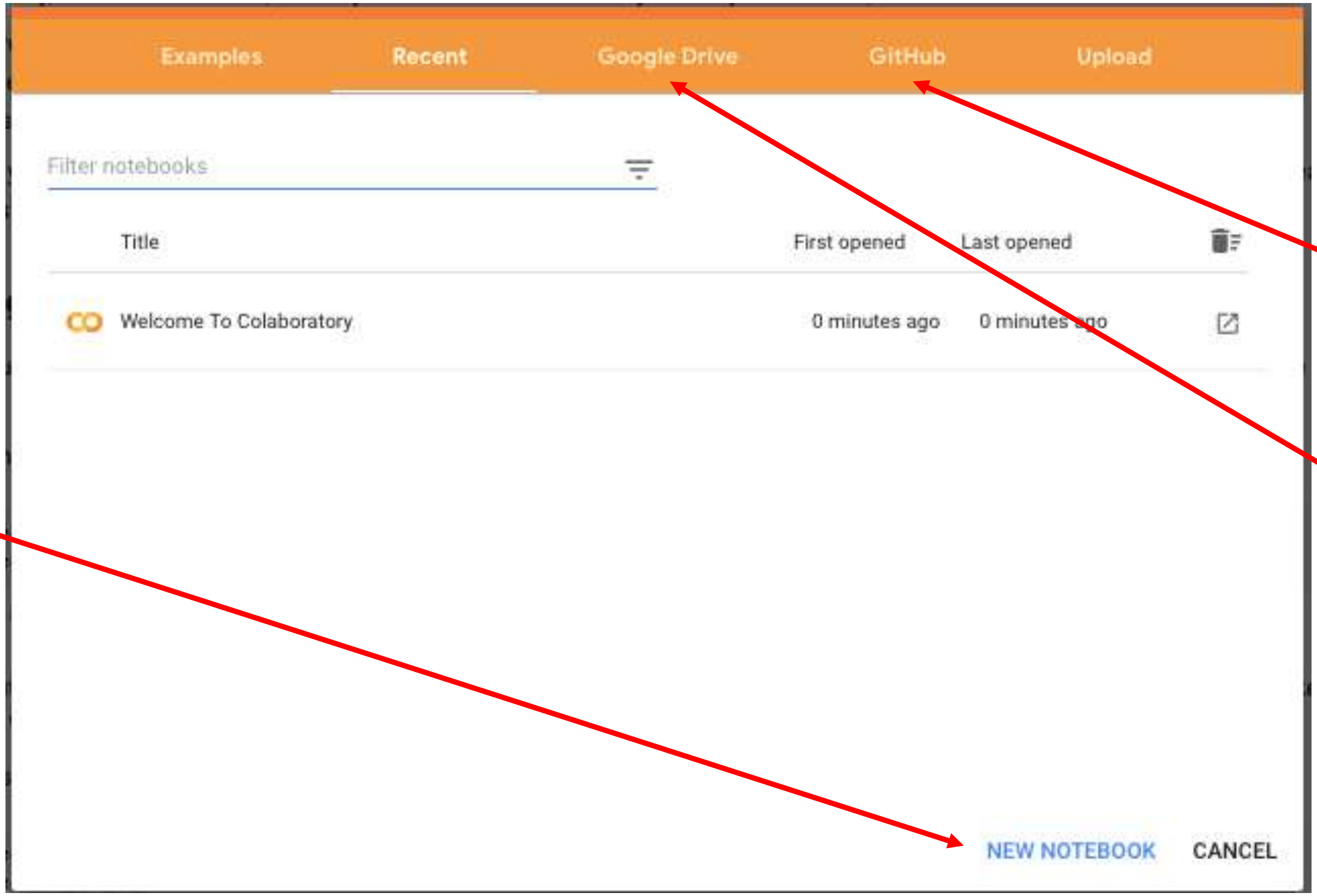
Manchester Metropolitan University

Google Colaboratory



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

Google Colaboratory



Click here to open a new Python 3 ipynb notebook

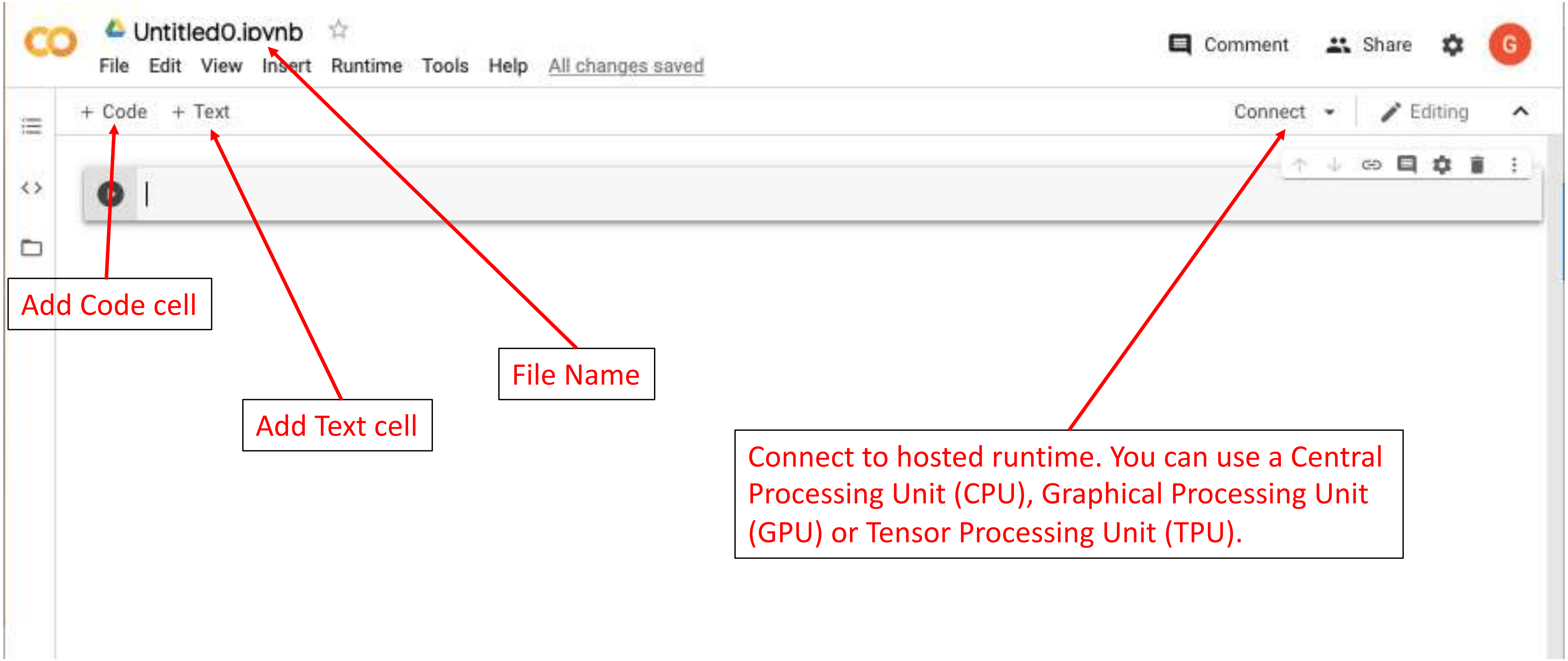
GitHub

Google Drive

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

Manchester Metropolitan University

Google Colab: Untitled0.ipynb Notebook



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

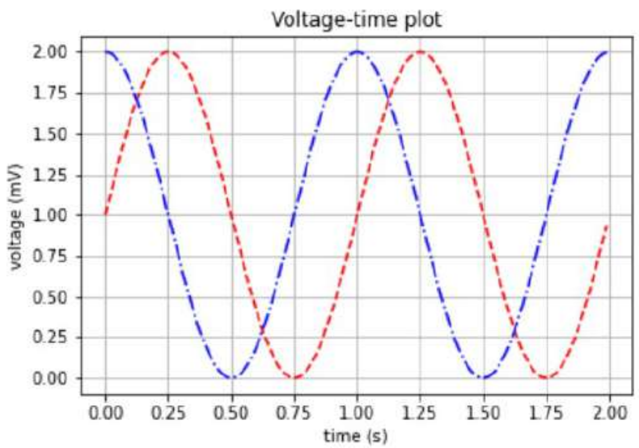
Loading a figure from your computer:

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

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

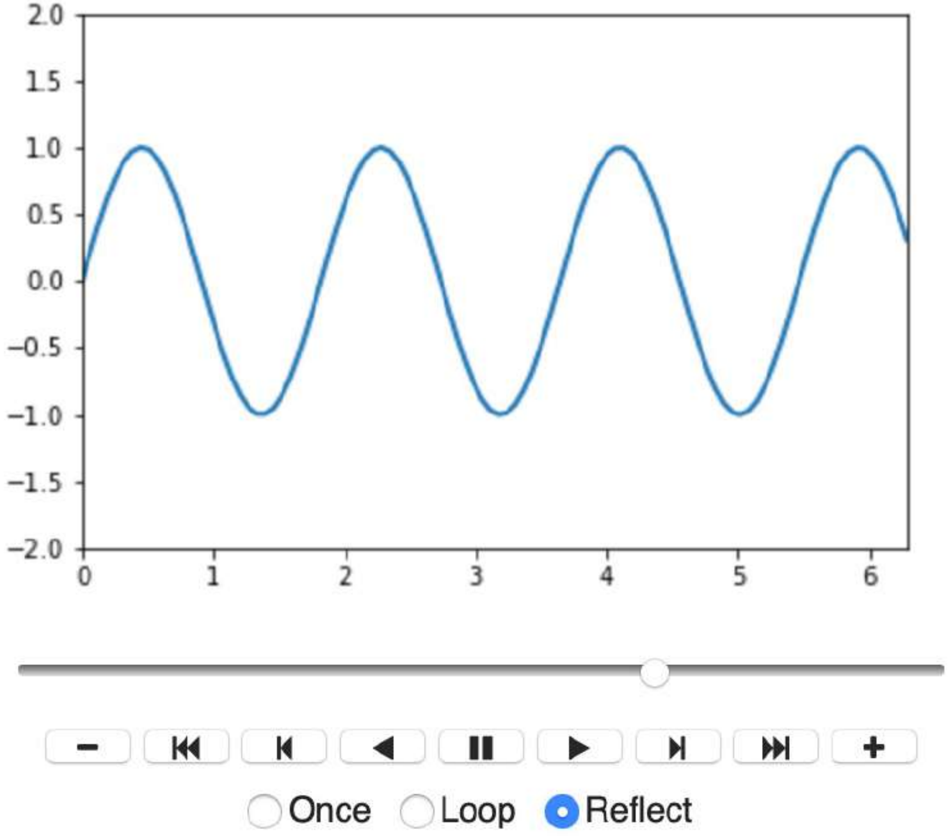
Choose Files no files selected Cancel upload

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



Animation in Google Colab

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation, rc
from IPython.display import HTML
# Set up figure.
fig, ax = plt.subplots()
plt.close()
# Set domain and range.
ax.set_xlim(( 0, 2 * np.pi))
ax.set_ylim((-2, 2))
# Set line width.
line, = ax.plot([], [], lw=2)
# Draw a clear frame.
def init():
    line.set_data([], [])
    return (line,)
# Function to animate.
def animate(n):
    x = np.linspace(0, 2 * np.pi, 100)
    y = np.sin(0.05 * x * n)
    line.set_data(x, y)
    return (line,)
# Animate. The interval command changes speed of animation.
anim = animation.FuncAnimation(fig, animate, init_func=init,
                               frames=100, interval=100, blit=True)
# Note: below is the part which makes it work on Colab.
rc('animation', html='jshtml')
anim
```



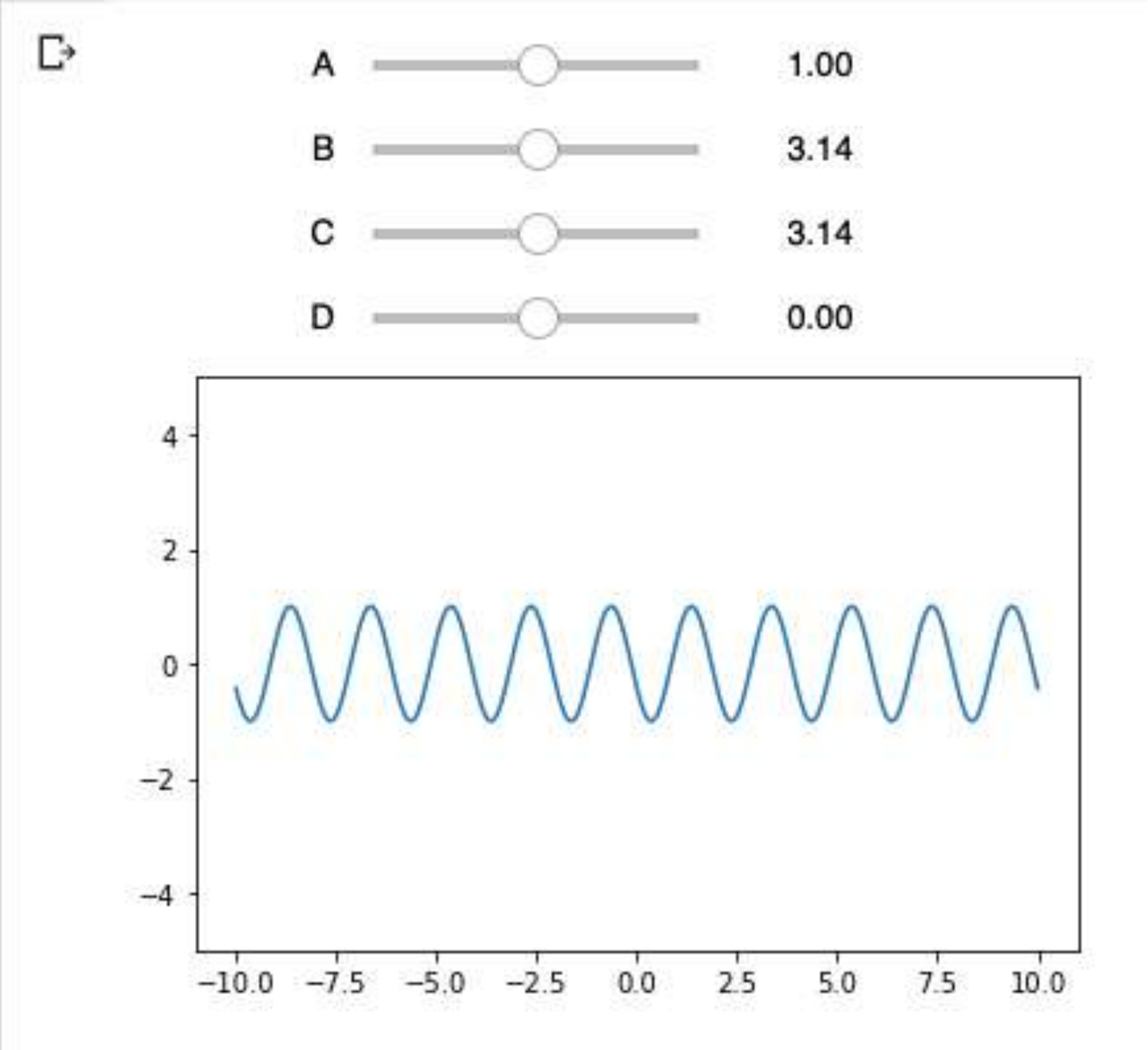
Edit to animate $y = e^{-0.01at} \sin(t)$, for $0 \leq a \leq 50$?

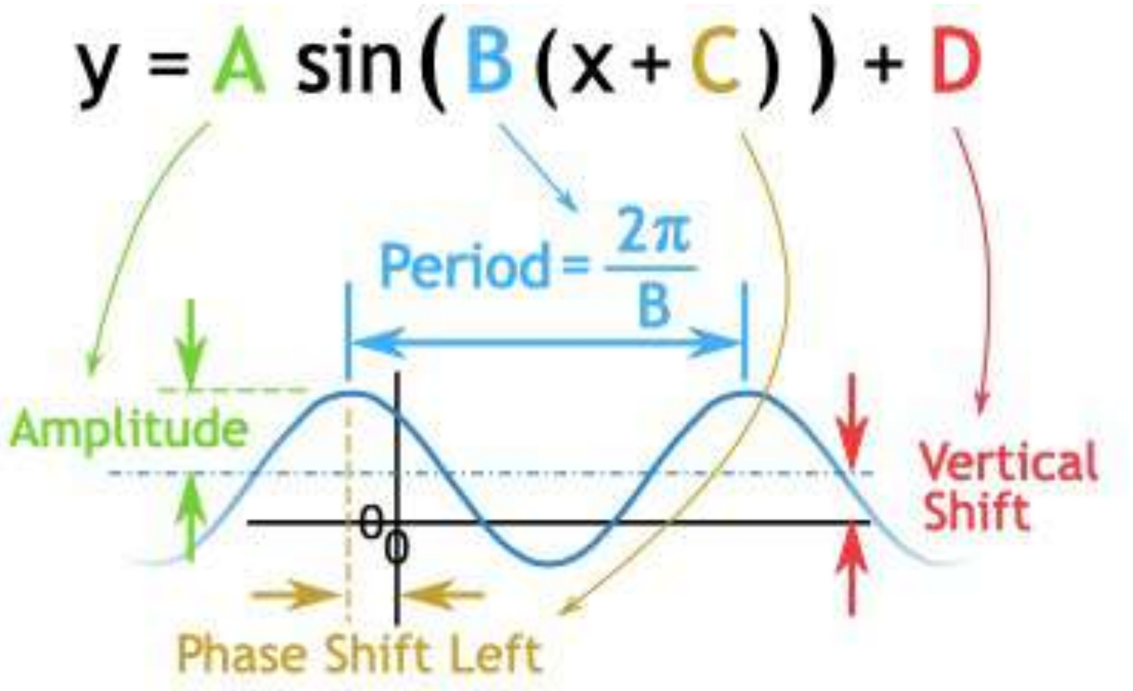
Manchester Metropolitan University

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 1





Biology: Population of Blowfly: Start Session 2

$x_{n+1} = \mu x_n(1 - x_n)$, x_n is population and μ is a parameter.

```
# Program_6a.py: Iteration of the logistic map function.
import numpy as np
import matplotlib.pyplot as plt
mu , x = 4 , 0.2                  # For case (iv).
xs = [0.2]                        # Initially, 20% of the tank is full.
for i in range(50):
  x = mu * x * (1 - x)
  xs = np.append(xs , x)
  #print(x)                       # Print the x values if you like.
plt.plot(xs)
plt.xlabel("n")
plt.ylabel("$x_n$")
plt.show()
```

Biology: The Logistic Map

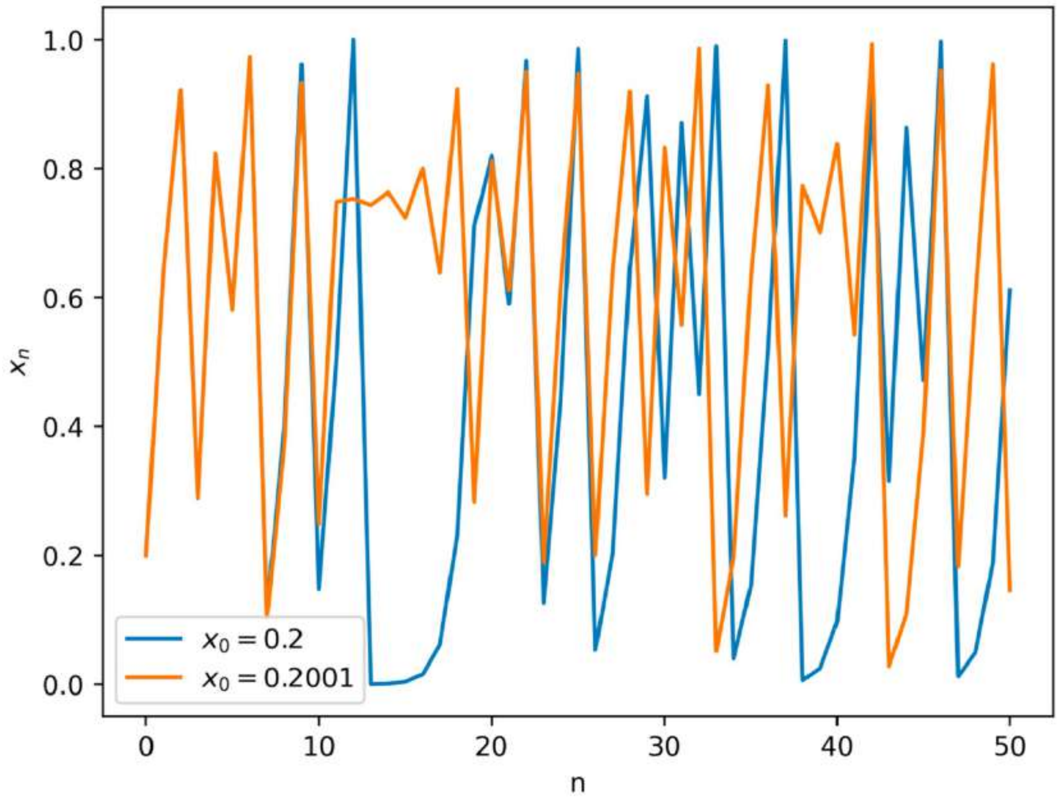


Figure: Sensitivity to initial conditions: chaos.

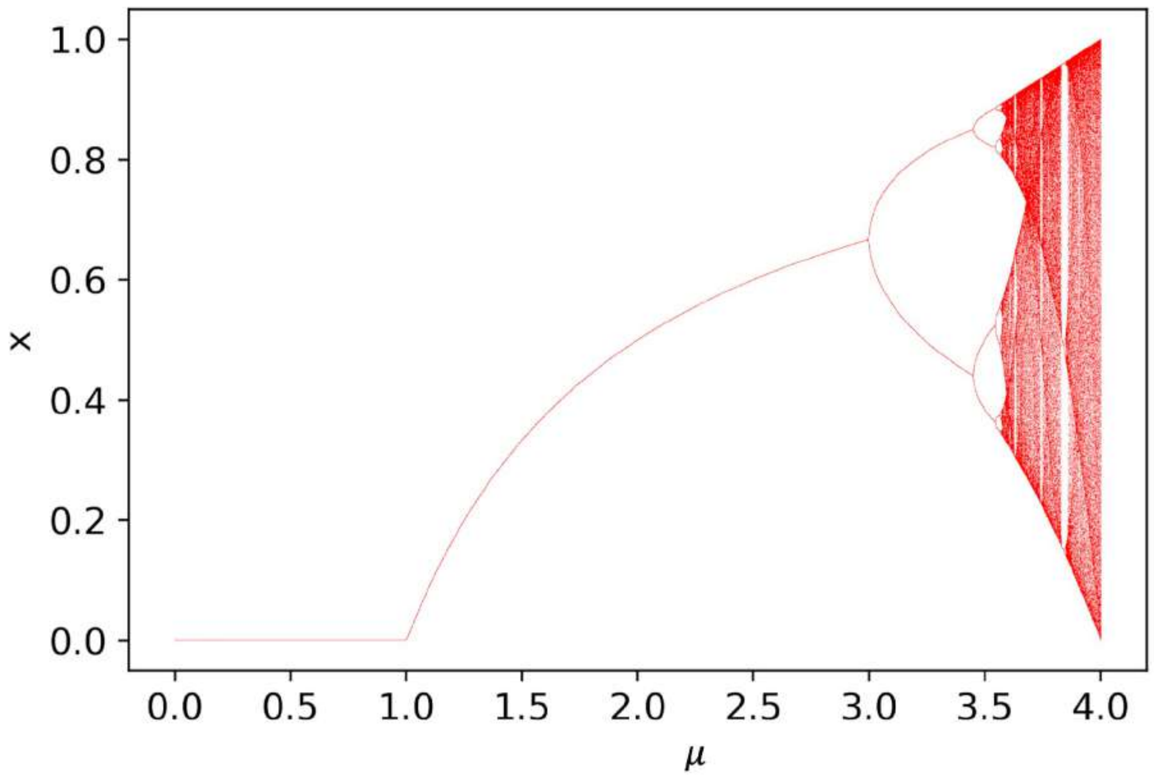
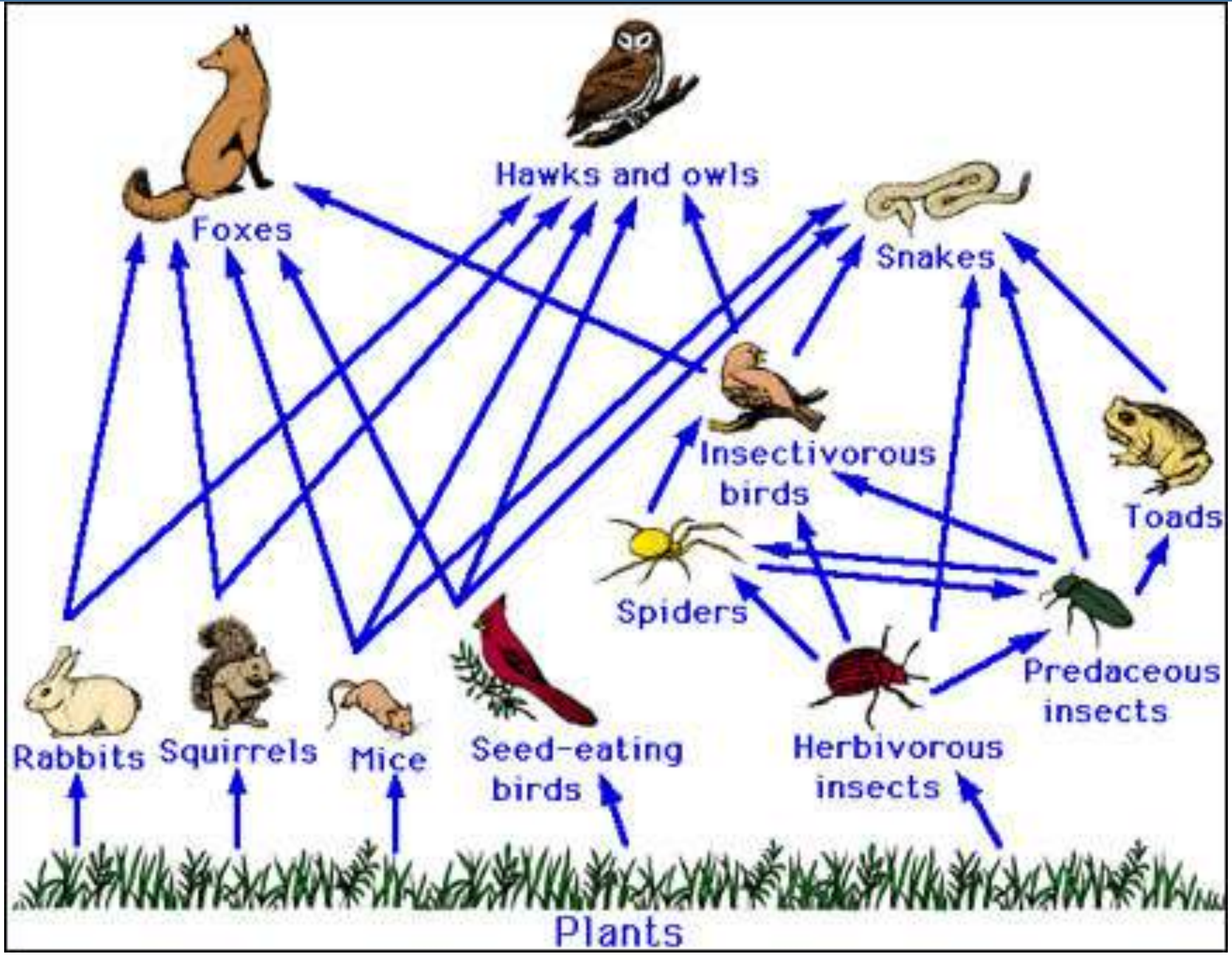


Figure: Bifurcation diagram.

Manchester Metropolitan University

Interacting Species in the UK



The Holling-Tanner Model: Predator-Prey

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.

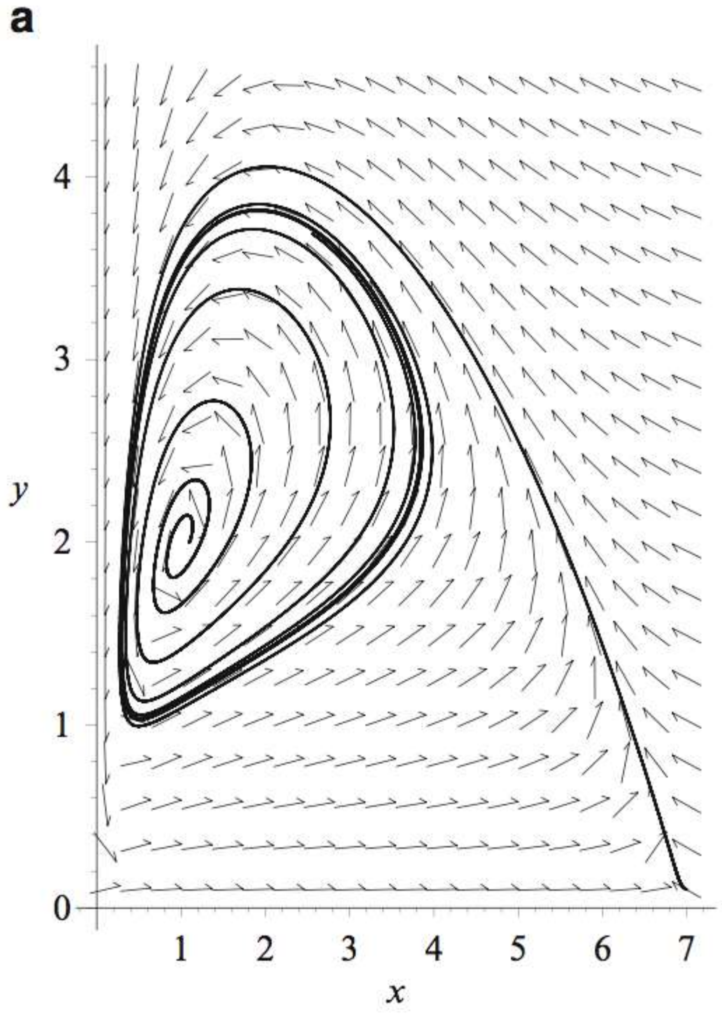
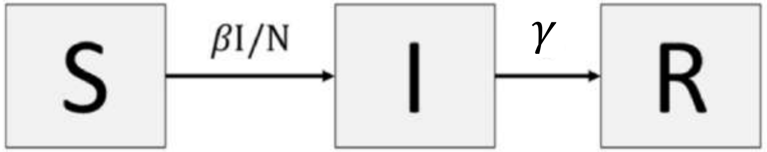


Fig. A limit cycle of the Holling-Tanner Model

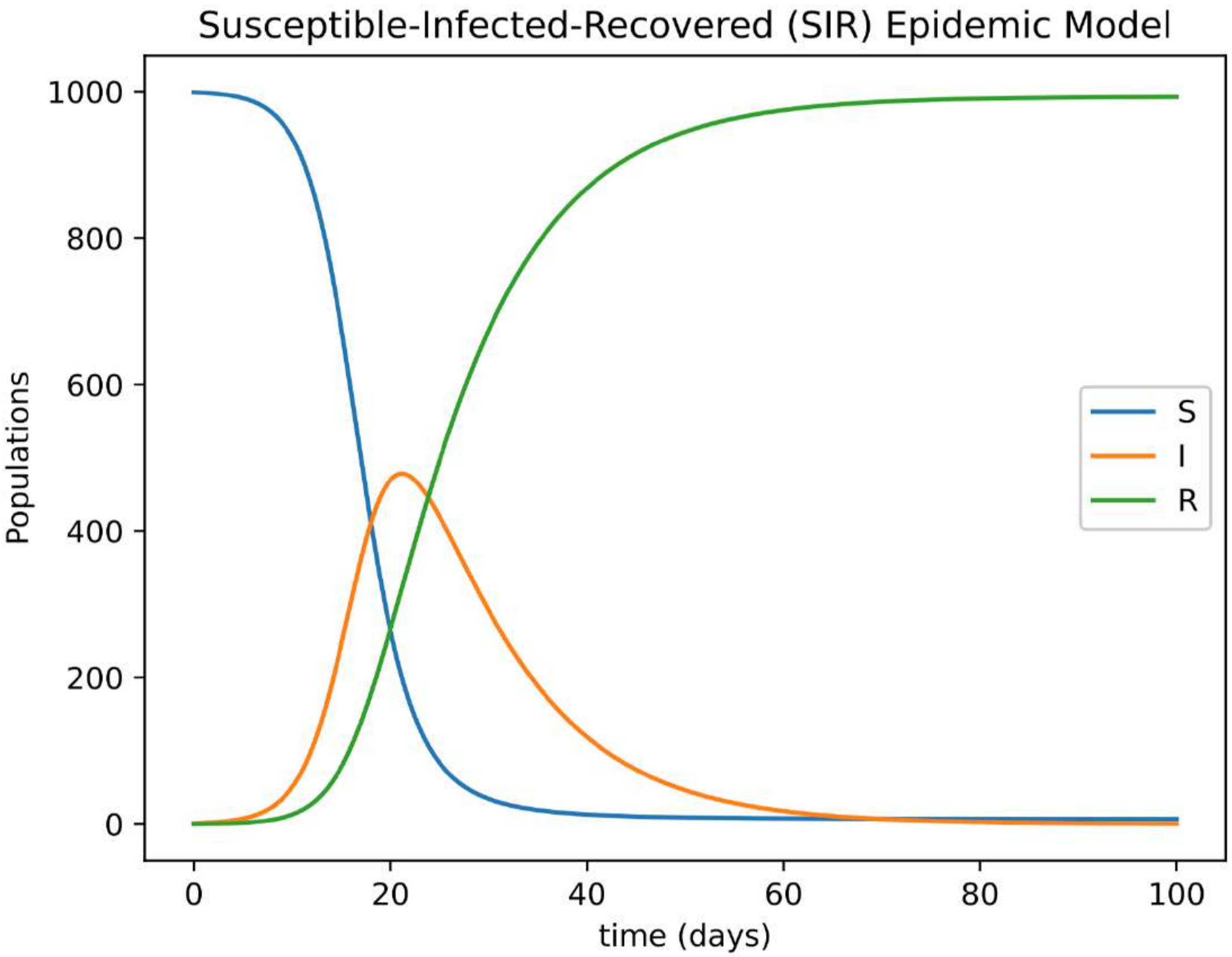
Modelling Epidemics with ODEs



$$\frac{dS}{dt} = -\frac{\beta S I}{N},$$

$$\frac{dI}{dt} = \frac{\beta S I}{N} - \gamma I,$$

$$\frac{dR}{dt} = \gamma I.$$



Modelling Epidemics with ODEs

$$S \xrightarrow{\beta I/N} I \xrightarrow{\gamma} R$$

$$\frac{dS}{dt} = -\frac{\beta SI}{N},$$

$$\frac{dI}{dt} = \frac{\beta SI}{N} - \gamma I,$$

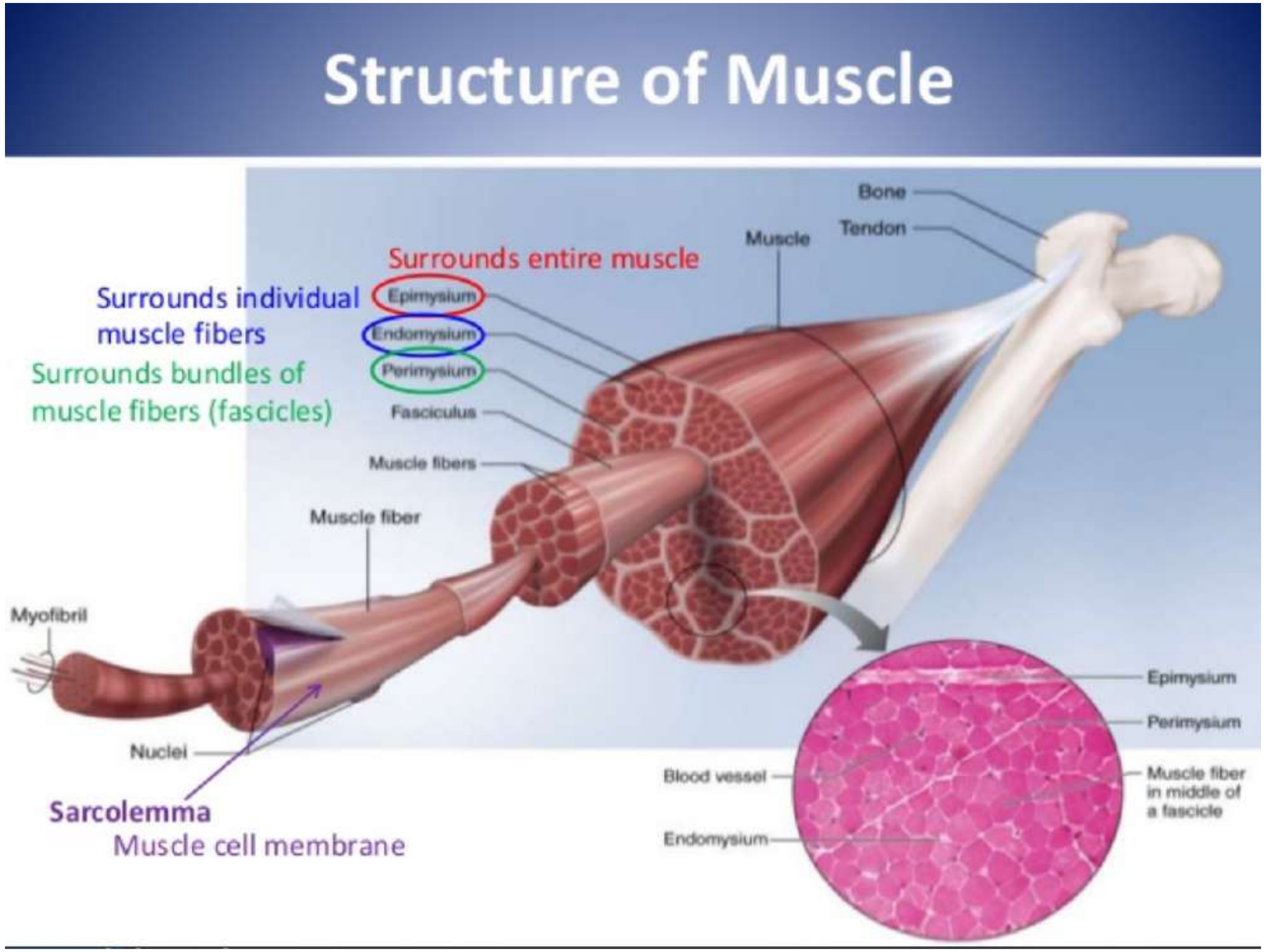
$$\frac{dR}{dt} = \gamma I.$$

```
# Program_6e.py: SIR Epidemic model.
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
# Set the parameters.
beta, gamma = 0.5, 0.1
S0, I0, R0, N = 999, 1, 0, 1000
tmax, n = 100, 1000
def SIR_Model(X, t, beta, gamma):
    S, I, R = X
    dS = - beta * S * I / N
    dI = beta * S * I / N - gamma * I
    dR = gamma * I
    return(dS, dI, dR)
t = np.linspace(0, tmax, n)
f = odeint(SIR_Model, (S0, I0, R0), t, args = (beta, gamma))
S, I, R = f.T
plt.figure(1)
plt.xlabel("Time (days)")
plt.ylabel("Populations")
plt.title("Susceptible-Infected-Recovered (SIR) Epidemic Model")
plt.plot(t, S, label = "S")
plt.plot(t, I, label = "I")
plt.plot(t, R, label = "R")
legend = plt.legend(loc = "best")
plt.show()
```

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)

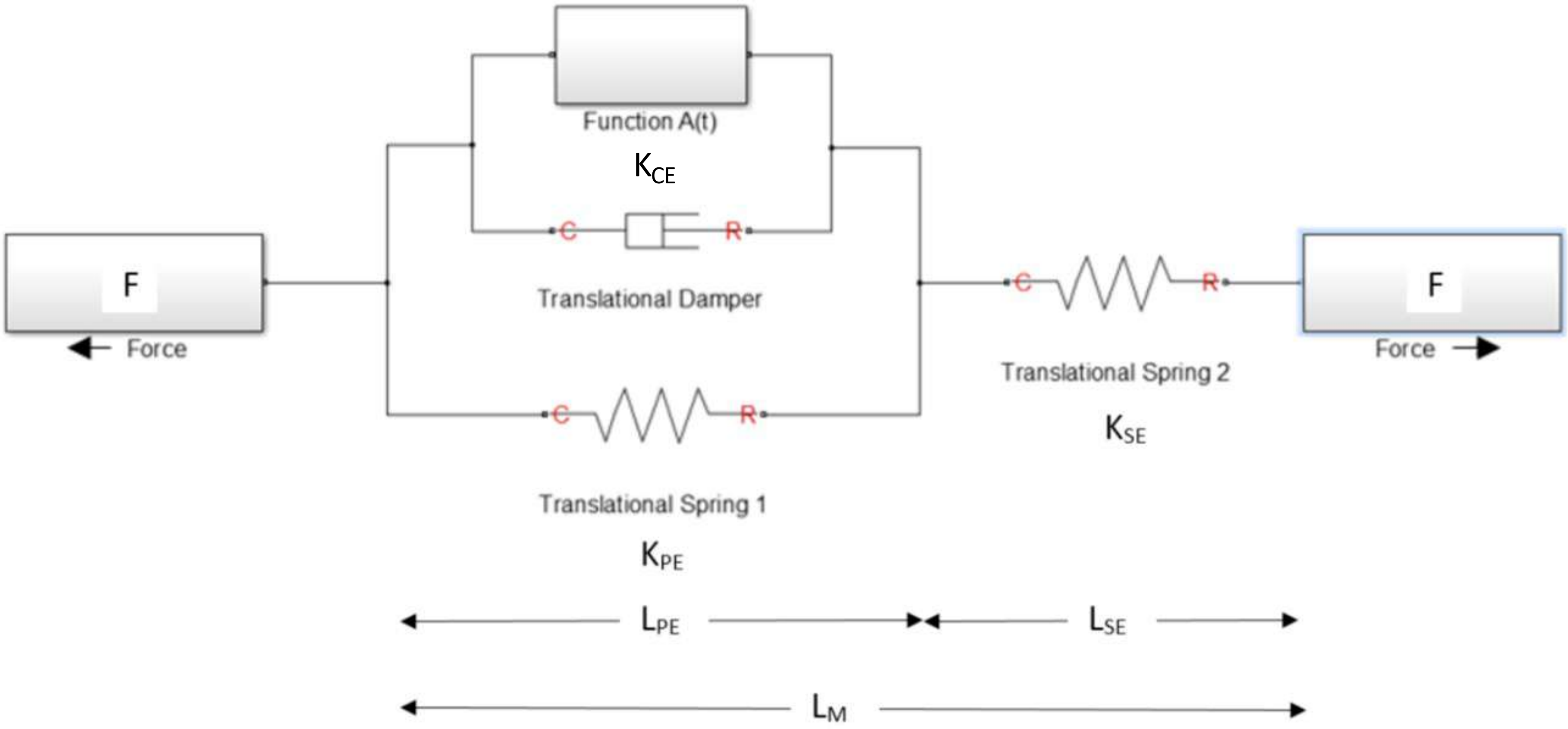


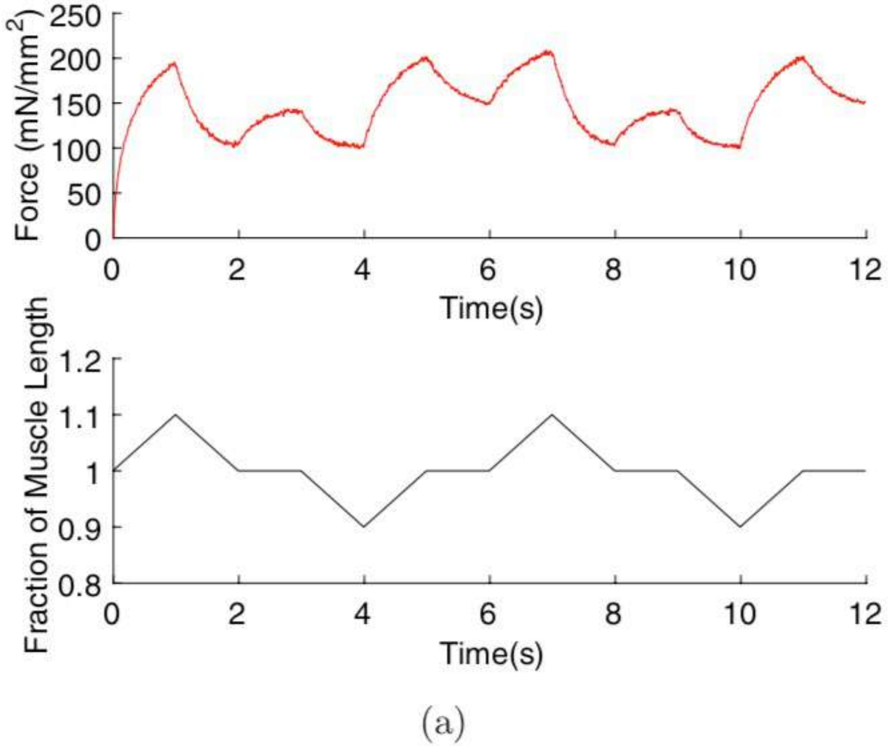
Fig. 18. A Simscape simulation of the Hill muscle model comprising of a series element (SE), a contractile element (CE) and a parallel element (PE).

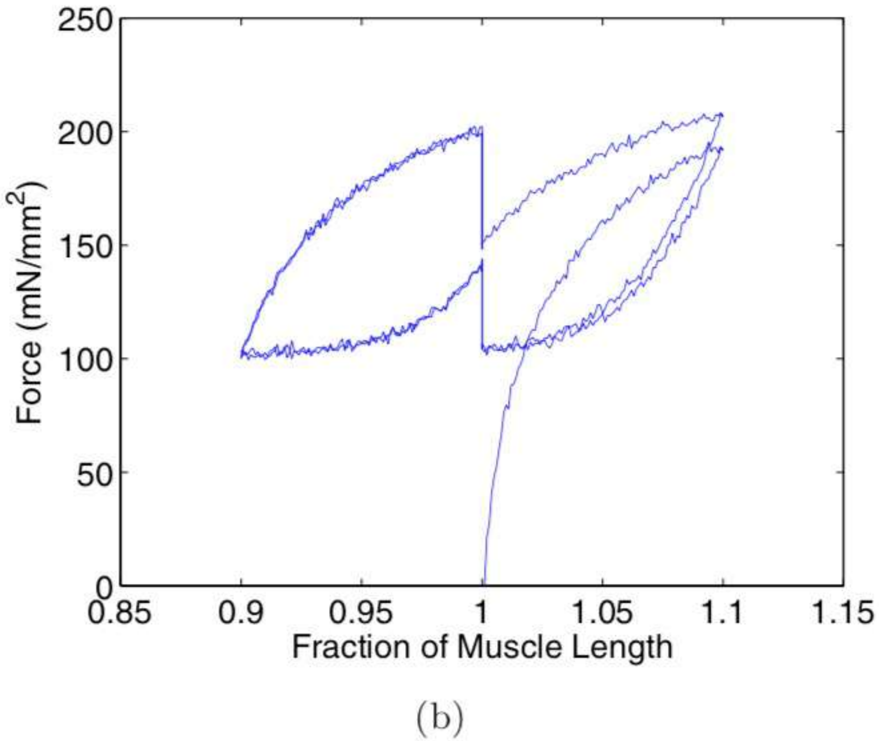
Muscle Model (Python Program of Hill Model)

```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)
```

Manchester Metropolitan University

Muscle Model (Lengthening and Shortening)

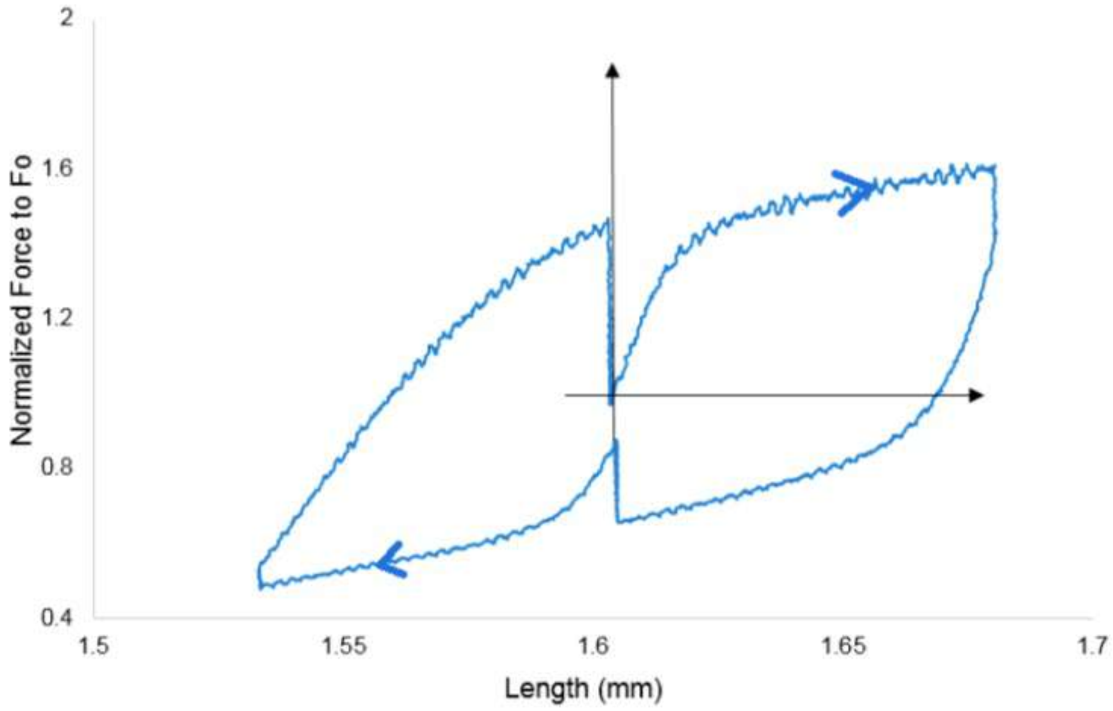


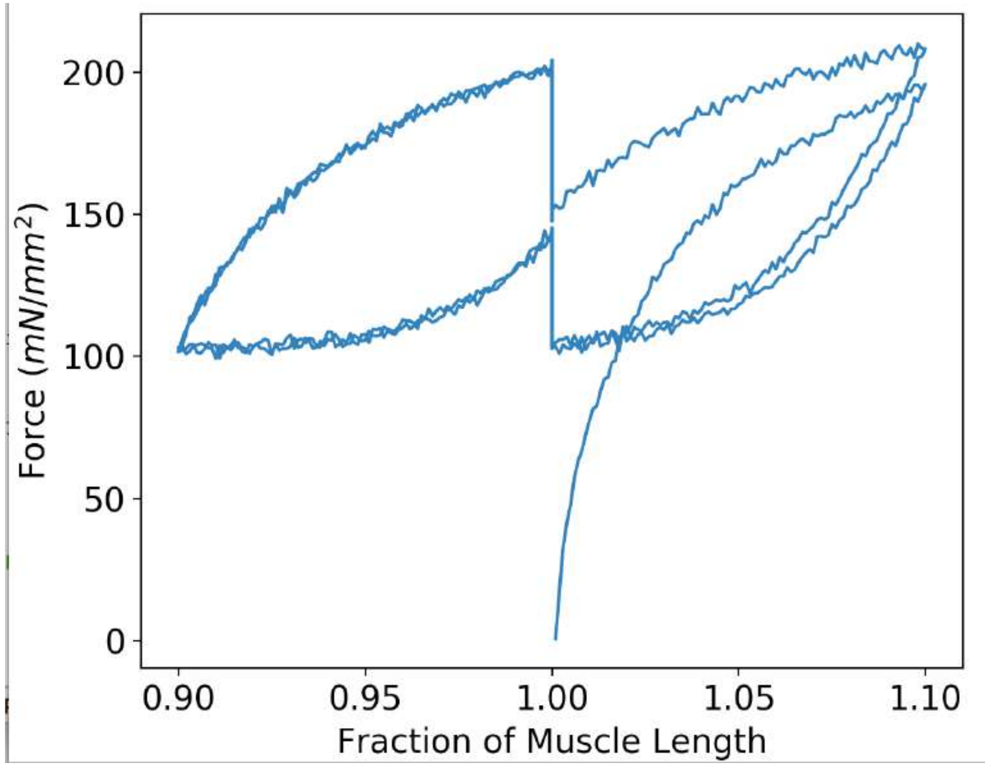


Lengthening and shortening with rests.

Hysteresis curves from modelling.

Muscle Mode





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.

Manchester Metropolitan University

Chemistry: Balancing Chemical Equations

$$x_1\mathrm{KI} + x_2\mathrm{KClO_3} + x_3\mathrm{HCl} \rightleftharpoons x_4\mathrm{I_2} + x_5\mathrm{H_2O} + x_6\mathrm{KCl},$$

Chemical Composition Table

Element	KI	$KClO_3$	HCl	I_2	H_2O	KCl
K	1	1	0	0	0	1
I	1	0	0	2	0	0
O	0	3	0	0	1	0
H	0	0	1	0	2	0
Cl	0	1	1	0	0	1

Chemistry: Balancing Chemical Equations

```
# Program_7a.py: Compute the matrix null-space vector.
from sympy import Matrix
# Construct the augmented matrix.
ACCM=Matrix([[1,1,0,0,0,1],\
            [1,0,0,2,0,0],\
            [0,3,0,0,1,0],\
            [0,0,1,0,2,0],\
            [0,1,1,0,0,1],\
            [0,0,0,0,0,1]])
print(ACCM)
invACCM=ACCM.inv() # Find the inverse matrix.
print(invACCM)
Nullv=invACCM.col(5) / min(abs(invACCM.col(5))) # Last column.
print(Nullv) # Scaled null-space vector.
```

The solution is, Nullv=Matrix([[−6], [−1], [−6], [3], [3], [7]]), giving:

$$x_1 = 6, x_2 = 1, x_3 = 6, x_4 = 3, x_5 = 3, x_6 = 7,$$

and the balanced chemical-reaction equation is:

$$6KI + KClO_3 + 6HCl \rightleftharpoons 3I_2 + 3H_2O + 7KCl.$$

Manchester Metropolitan University

Chemical Kinetics: A Simple Model

```python
# Chemical kinetics - conservation of mass.
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
# Set parameters and initial conditions
r1, r2 = 0.01, 0.02
x0, y0, z0 = 1000, 0, 0
# Maximum time point and total number of time points
tmax, n = 500, 10000
def Chemical_Kinetics(X, t, r1, r2):
#The Differential Equations
    x, y, z = X
    dx = -r1 * x
    dy = r1 * x - r2 * y
    dz = r2 * y
    return (dx, dy, dz)
# Integrate differential equations on the time grid t.
t = np.linspace(0, tmax, n)
f = odeint(Chemical_Kinetics, (x0, y0, z0), t, args=(r1, r2))
x, y, z = f.T
plt.figure(1)
plt.xlabel("Time")
plt.ylabel("Concentrations")
plt.title("Chemical Kinetics")
plt.plot(t, x, label="|A|")
plt.plot(t, y, label="|B|")
plt.plot(t, z, label="|C|")
legend = plt.legend(loc="best")
plt.show()
```

Compartmental Model

$|\mathrm{A}| = x, \ |\mathrm{B}| = y, \ |\mathrm{C}| = z$

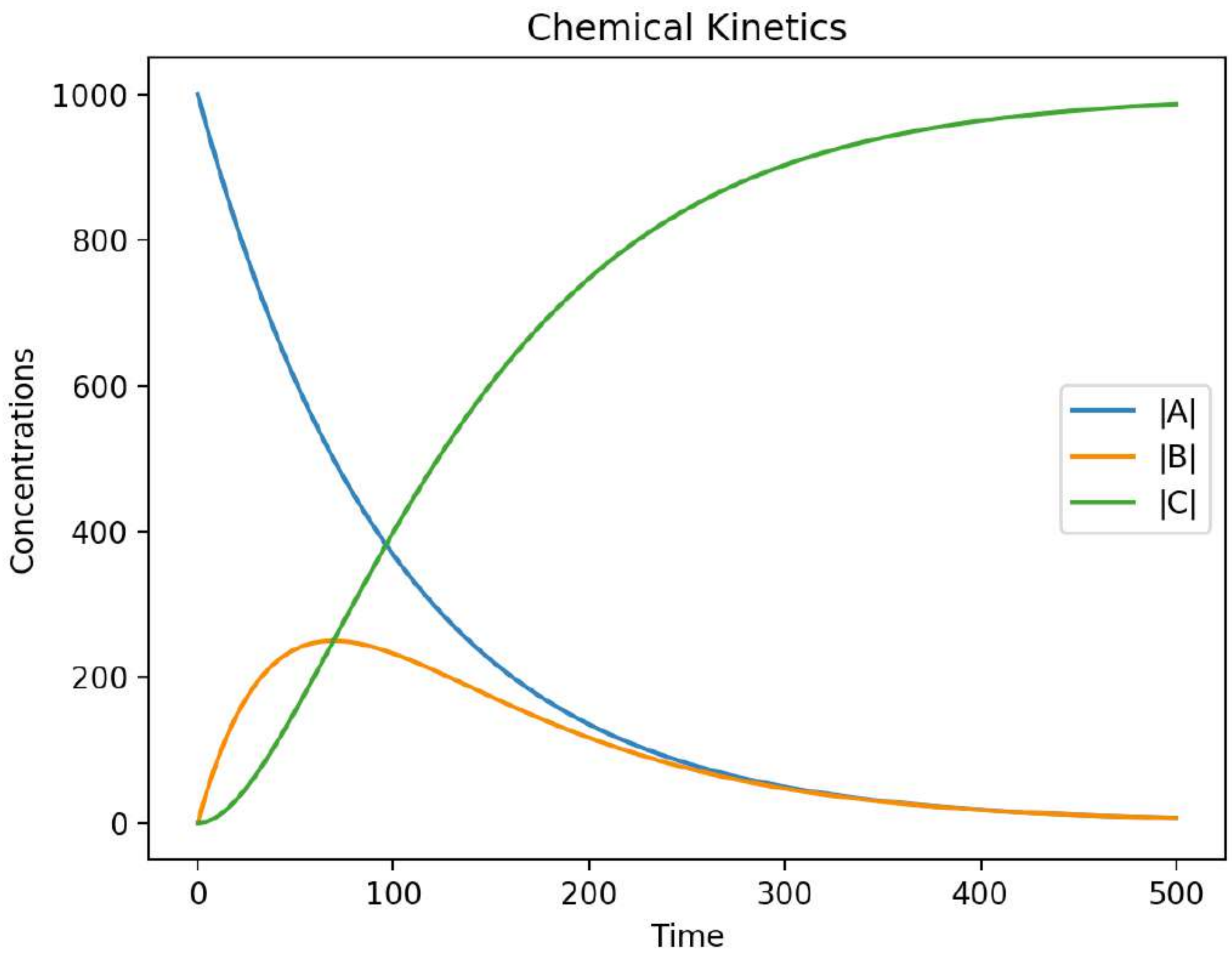
$$\frac{dx}{dt} = -r_1 x$$

$$\frac{dy}{dt} = r_1 x - r_2 y$$

$$\frac{dz}{dt} = r_2 y$$

$$\boxed{\mathrm{A}} \xrightarrow{\mathrm{r1}} \boxed{\mathrm{B}} \xrightarrow{\mathrm{r2}} \boxed{\mathrm{C}}$$

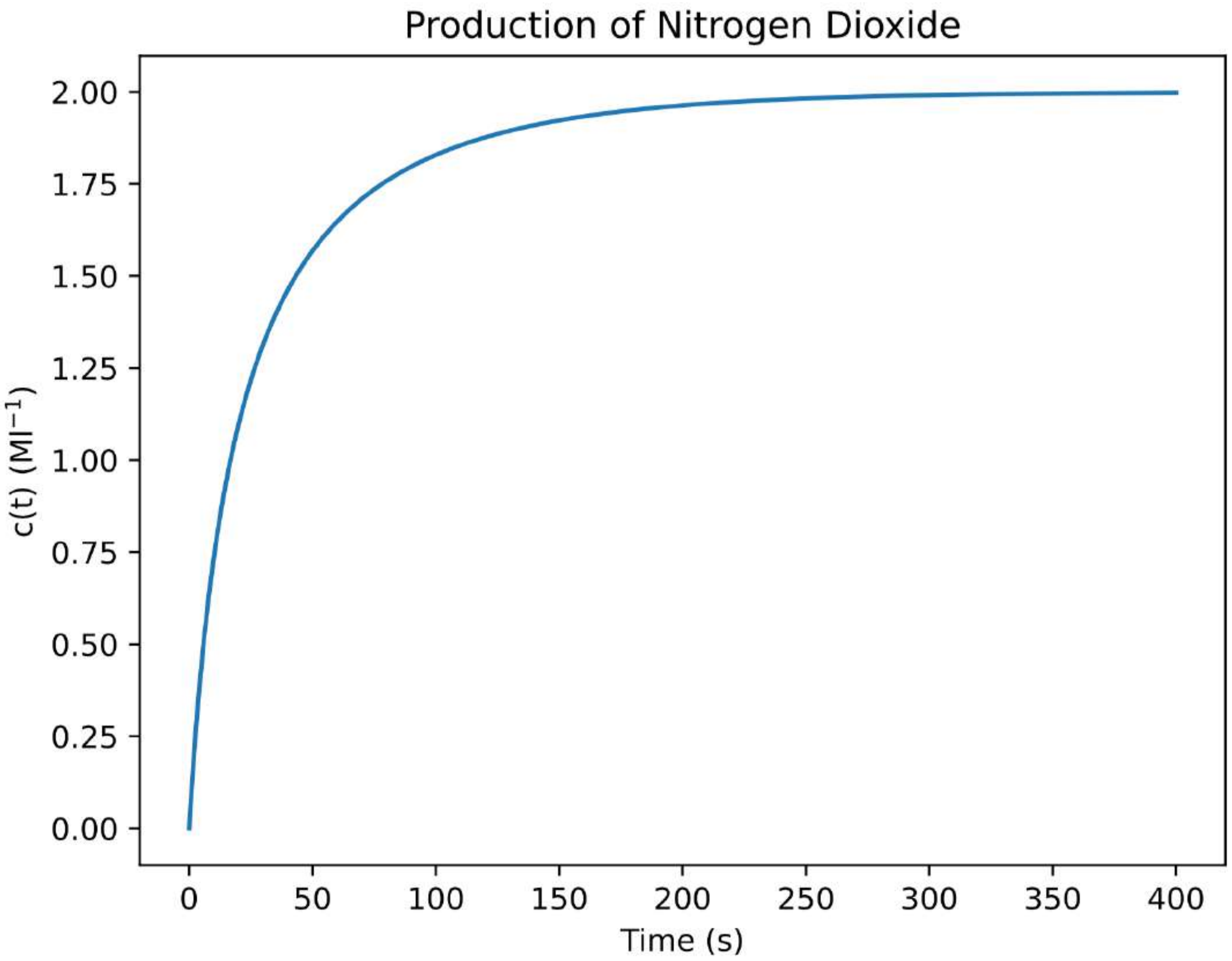
Chemical Kinetics: A Simple Model



Chemistry: Chemical Kinetics

$$2\mathrm{NO} + \mathrm{O_2} \rightleftharpoons 2\mathrm{NO_2}$$

$$\frac{dc}{dt} = k\left(a_0 - c\right)^2\left(b_0 - \frac{c}{2}\right)$$

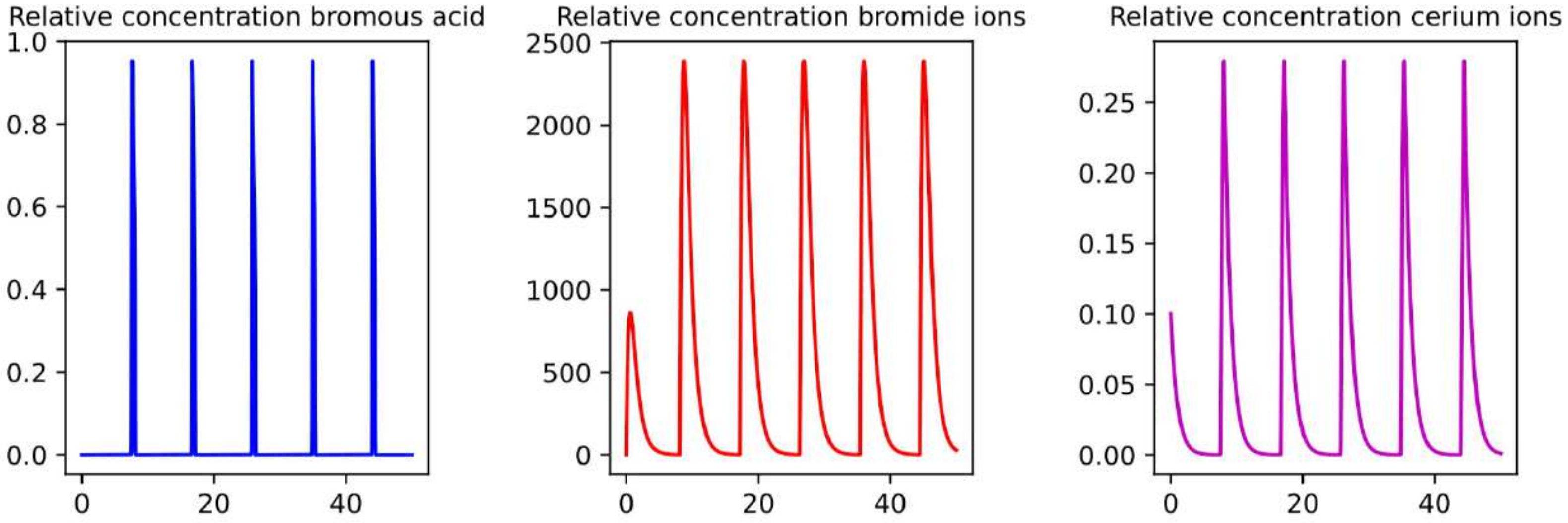


Chemistry: Chemical Kinetics and ODEs

```
# Program_7b.py: Production of Nitrogen Dioxide.
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
k , a0 , b0 , c0 = 0.00713 , 4 , 1, 0
def ode(c , t):
  dcdt = k * (a0 - c)**2 * (b0 - c / 2)
  return dcdt
t = np.linspace(0 , 400 , 401) # t=[0,1,2,...,400]
c = odeint(ode , c0 , t)
plt.xlabel("Time (s)")
plt.ylabel("c(t) (Ml$^{-1})$")
plt.title("Production of Nitrogen Dioxide")
plt.plot(t , c)
print("c(100) = ", c[100], "Moles per litre")
plt.show()
```

Chemistry: Oscillating Chemical Reactions

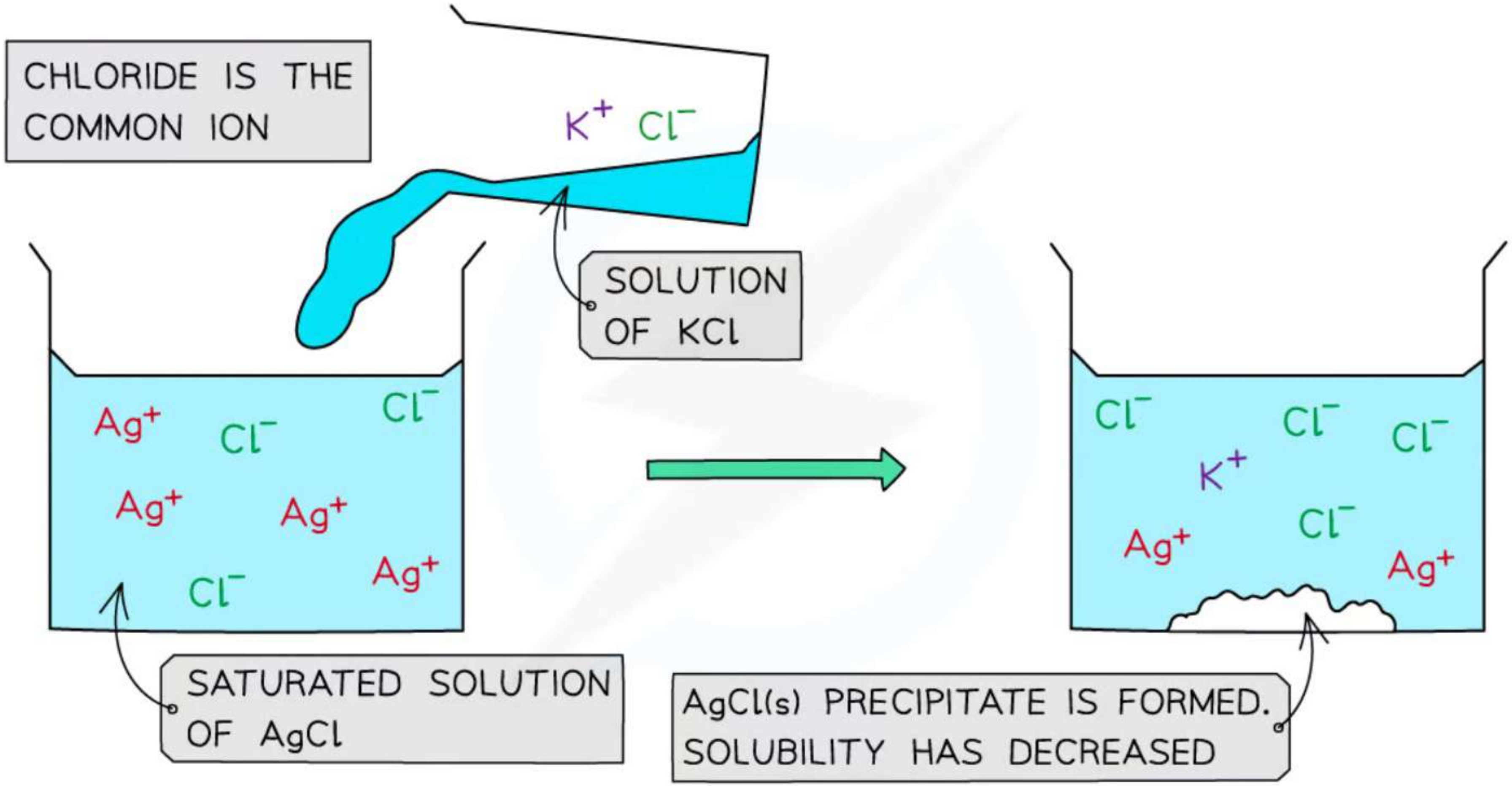
$$
\begin{aligned}
&BrO_3^- + Br^- \rightarrow HBrO_2 + HOBr, && \text{Rate} = k_1[BrO_3^-][Br^-] \\
&HBrO_2 + Br^- \rightarrow 2HOBr, && \text{Rate} = k_2[HBrO_2][Br^-] \\
&BrO_3^- + HBrO_2 \rightarrow 2HBrO_2 + 2M_{OX}, && \text{Rate} = k_3[BrO_3^-][HBrO_2] \\
&2HBrO_2 \rightarrow BrO_3^- + HOBr, && \text{Rate} = k_4[HBrO_2]^2 \\
&OS + M_{OX} \rightarrow \tfrac{1}{2}CBr^-, && \text{Rate} = k_5[OS][M_{OX}]
\end{aligned}
$$



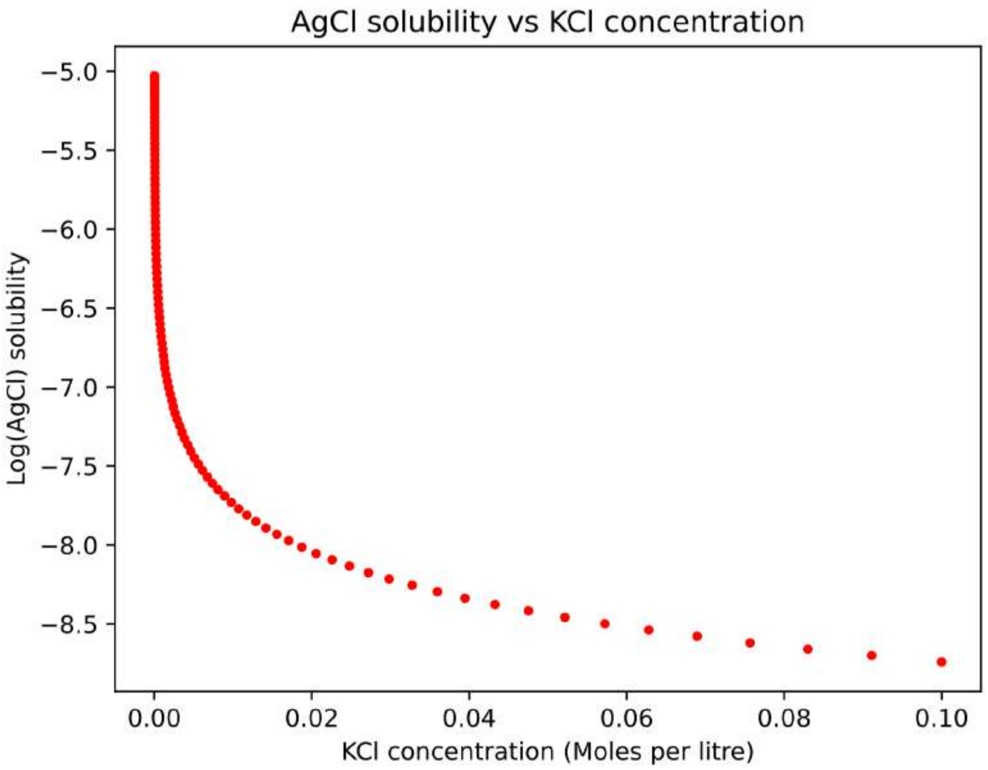
Chemistry: Periodic Reaction

Chemical Oscillation - Belousov-Zhabotinsky reaction

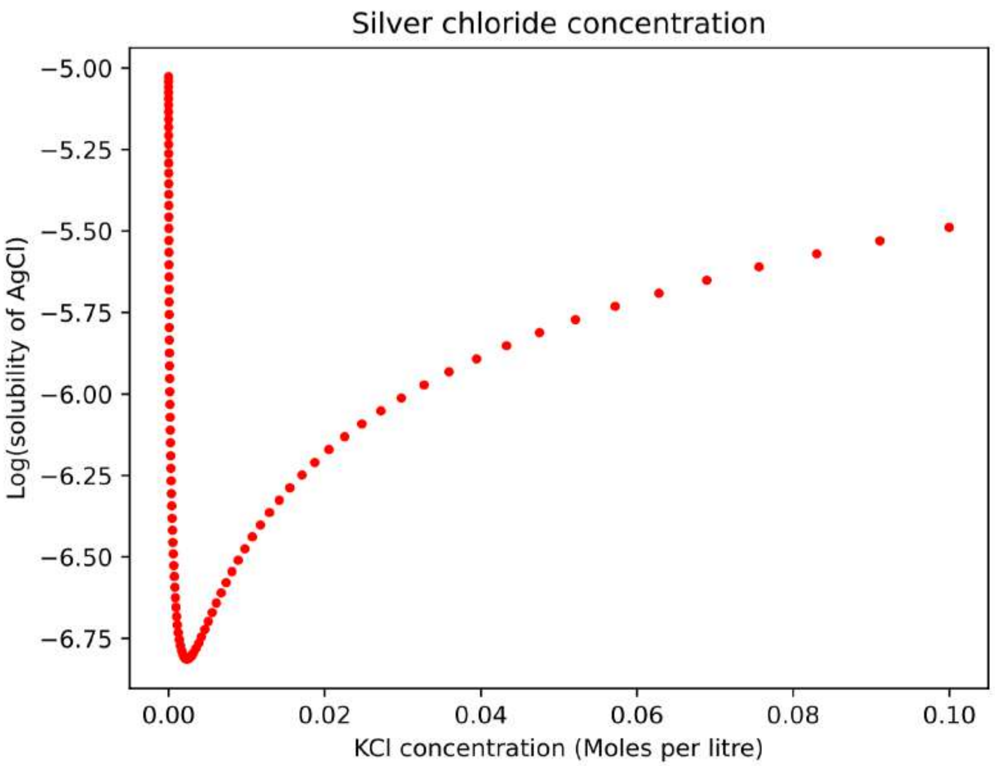
Chemistry: Common Ion Effect in Solubility



Chemistry: Common Ion Effect Silver Chloride: End Session 2



$$AgCl(s) \rightleftharpoons Ag^{+}(aq) + Cl^{-}(aq)$$



$$AgCl(s) \rightleftharpoons Ag^{+}(aq) + Cl^{-}(aq)$$

$$Ag^{+}(aq) + 2Cl^{-}(aq) \rightleftharpoons AgCl_2^{-}(aq)$$

Data Science: Pandas and Drug Efficacy: Start Session 3

```
# Program_8a.py: Create a data frame for drug efficacy.
import numpy as np
import pandas as pd
df1=pd.DataFrame({
    "Date": pd.Timestamp("20220125"),
    "Dosage (mg)": np.array(list(range(2,22,2))),
    "Sex " : pd.Categorical(["F","F","M","F","M",
                            "M","F","F","F","M"]),
    "Weight (Kg)" : pd.Series([52.2,65.8,80.7,53.5,40.9,
                     52.2,64.4,61.7,53.5,61.2],
                    dtype="float32"),
    "Efficacy (%)" : pd.Series([0,0,0,0,5,
                           20,90,100,100,100],dtype="int32")
    })
df2=pd.DataFrame({
    "Date": pd.Timestamp("20220126"),
    "Dosage (mg)": np.array(list(range(22,42,2))),
    "Sex " : pd.Categorical(["M","M","F","M","M",
                            "F","M","F","F","M"]),
    "Weight (Kg)" : pd.Series([
                     86.2,72.6,59.0,67.1,56.2,
                     61.2,78.0,45.3,54.4,88.9],
                    dtype="float32"),
    "Efficacy (%)" : pd.Series([

                           70,65,55,50,50,
                           45,10,0,0,0],dtype="int32")
     })
 df = pd.concat([df1 , df2] , ignore_index = True)
 df
```

	Date	Dosage (mg)	Sex	Weight (Kg)	Efficacy (%)
0	2022-01-25	2	F	52.200001	0
1	2022-01-25	4	F	65.800003	0
2	2022-01-25	6	M	80.699997	0
3	2022-01-25	8	F	53.500000	0
4	2022-01-25	10	M	40.900002	5
5	2022-01-25	12	M	52.200001	20
6	2022-01-25	14	F	64.400002	90
7	2022-01-25	16	F	61.700001	100
8	2022-01-25	18	F	53.500000	100
9	2022-01-25	20	M	61.200001	100
10	2022-01-26	22	M	86.199997	70
11	2022-01-26	24	M	72.599998	65
12	2022-01-26	26	F	59.000000	55
13	2022-01-26	28	M	67.099998	50
14	2022-01-26	30	M	56.200001	50
15	2022-01-26	32	F	61.200001	45
16	2022-01-26	34	M	78.000000	10
17	2022-01-26	36	F	45.299999	0
18	2022-01-26	38	F	54.400002	0
19	2022-01-26	40	M	88.900002	0

Manchester Metropolitan University

Data Science: Pandas

Python Command Lines	Comments
In[1]: df.dtypes	# Lists data types of columns.
In[2]: df.head()	# Lists the first 5 rows of df.
In[3]: df.head(10)	# Lists the first 10 rows of df.
In[4]: df.tail(3)	# Lists the last 3 rows of df.
In[5]: df.index	# Lists the range index.
In[6]: df.columns	# Lists the column headings.
In[7]: df.describe	# Statistics of numerical columns.
In[8]: df["Dosage (mg)"]	# Select a single column.
In[9]: df.sort_values(by="Weight (Kg)"	# Sort data by one column.
In[10]: df.loc[1 : 3]	# Slice rows.
In[11]: df[12 :]	# Slice rows.
In[12]: df.iloc[19]	# Lists data in index 19.
In[13]: df.iloc[1 : 3 , 0 : 2]	# Slice rows and columns using index.
In[14]: df.iat[2 , 2]	# Data in row 3, column 3.
In[15]: df[df["Weight (Kg)"]>60]	# Weights bigger than 60kg.
In[16]: df.to_csv("Drug_Trial.csv")	# Writes to csv file.
In[17]: pd.csv_read("Drug_Trial.csv")	# Loads csv file.
In[18]: df.to_excel("Drug_Trial.xlsx")	# Writes to excel file.
In[19]: df.T	# Transpose the data.
In[20]: df.dropna()	# Drop rows with NaN entries.

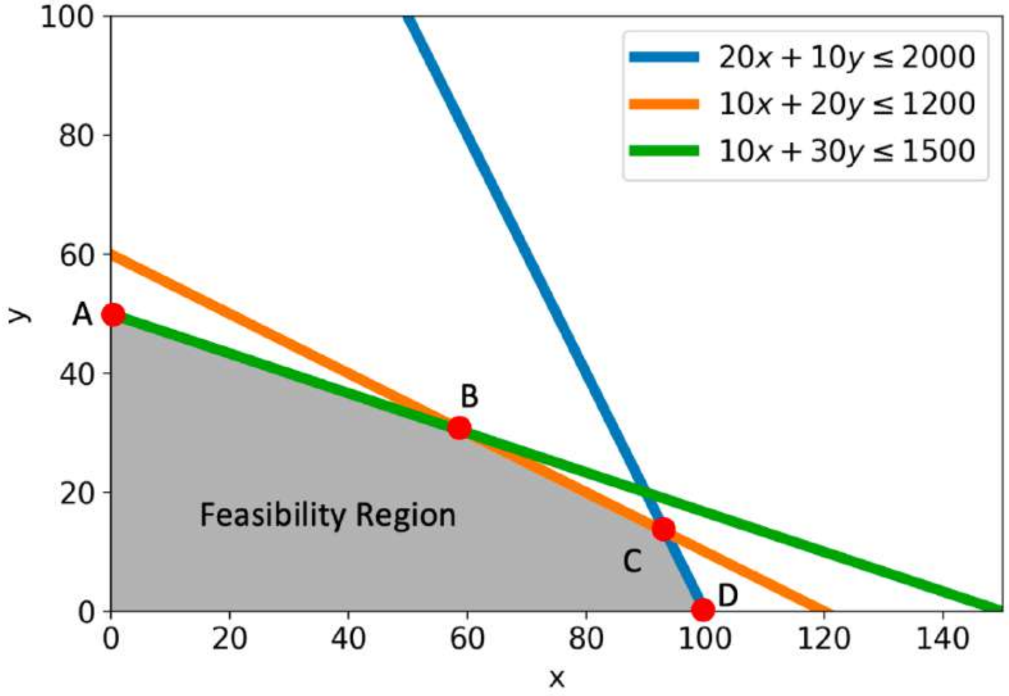
Data Science: Linear Programming

```
# Program_8b.py: Plotting the feasibility region and maximizing profit.
# Maximize P = 5x+12y, given 20x+10y<=2000, 10x+20y<=1200,10x+30y<=1500.
# x,y >= 0.
import numpy as np
import matplotlib.pyplot as plt
m = np.linspace(0,200,200)
x , y = np.meshgrid(m , m)
plt.imshow(((x>=0) & (y>=0) & (20*x+10*y<=2000) & (10*x+20*y<=1200) \
          & (10*x+30*y<=1500)).astype(int) , extent=(x.min(),x.max(), \
            y.min(),y.max()),origin="lower",cmap="Greys",alpha=0.3)

# Plot the constraint lines.
x = np.linspace(0, 200, 200)
y1 = (-20*x+2000) / 10
y2 = (-10*x+1200) / 20
y3 = (-10*x+1500) / 30
plt.rcParams["font.size"] = "14"
plt.plot(x, y1, label=r"$20x+10y \leq 2000$" , linewidth = 4)
plt.plot(x, y2, label=r"$10x+20y \leq 1200$" ,  linewidth = 4)
plt.plot(x, y3, label=r"$10x+30y \leq 1500$" , linewidth = 4)
# The maximum profit line.
# plt.plot(x , (-5 * x + 660) / 12 , label = r"$P_{max}=5x+12y$")
plt.xlim(0,150)
plt.ylim(0,100)
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

$$P = 5x + 12y.$$

$$10x + 30y \leq 1500, \quad 20x + 10y \leq 2000, \quad 10x + 20y \leq 1200,$$

$$x \geq 0, \quad y \geq 0.$$



Data Science: K Means Clustering: Program_8e.py

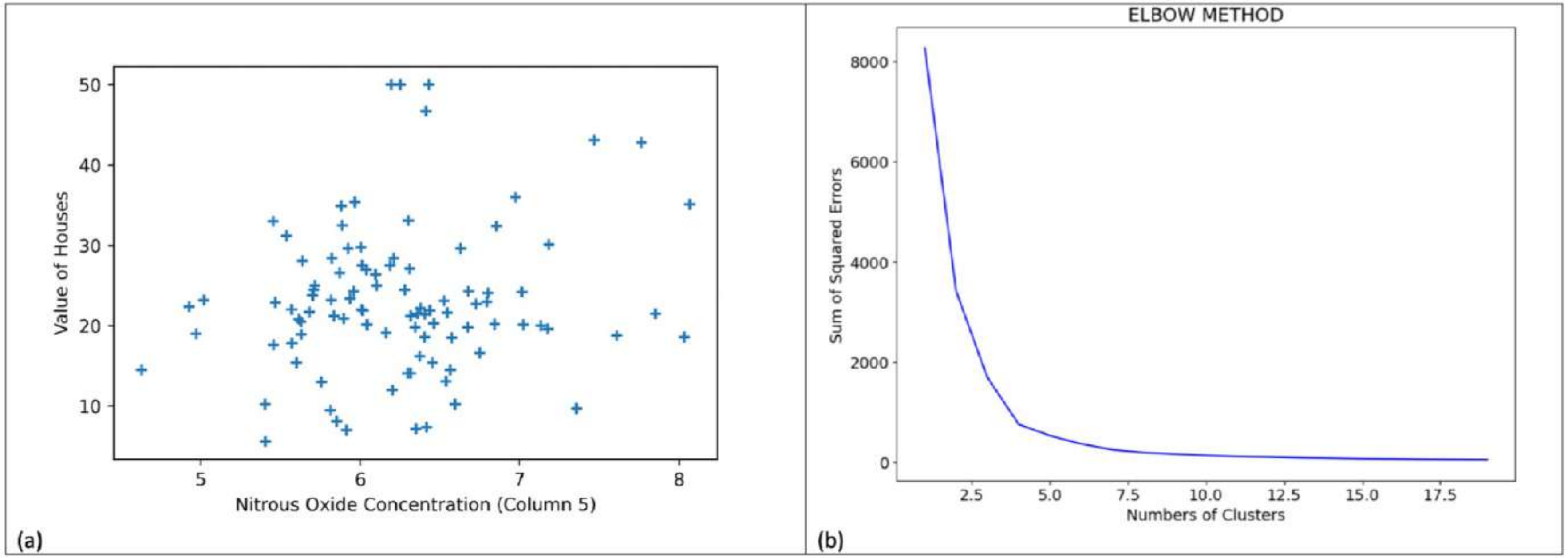


Figure 8.3 (a) Data from column five of the Boston-housing data. (b) The elbow method to determine the number of clusters.

Manchester Metropolitan University

Data Science: Program_8g.py: Clusters and Outliers

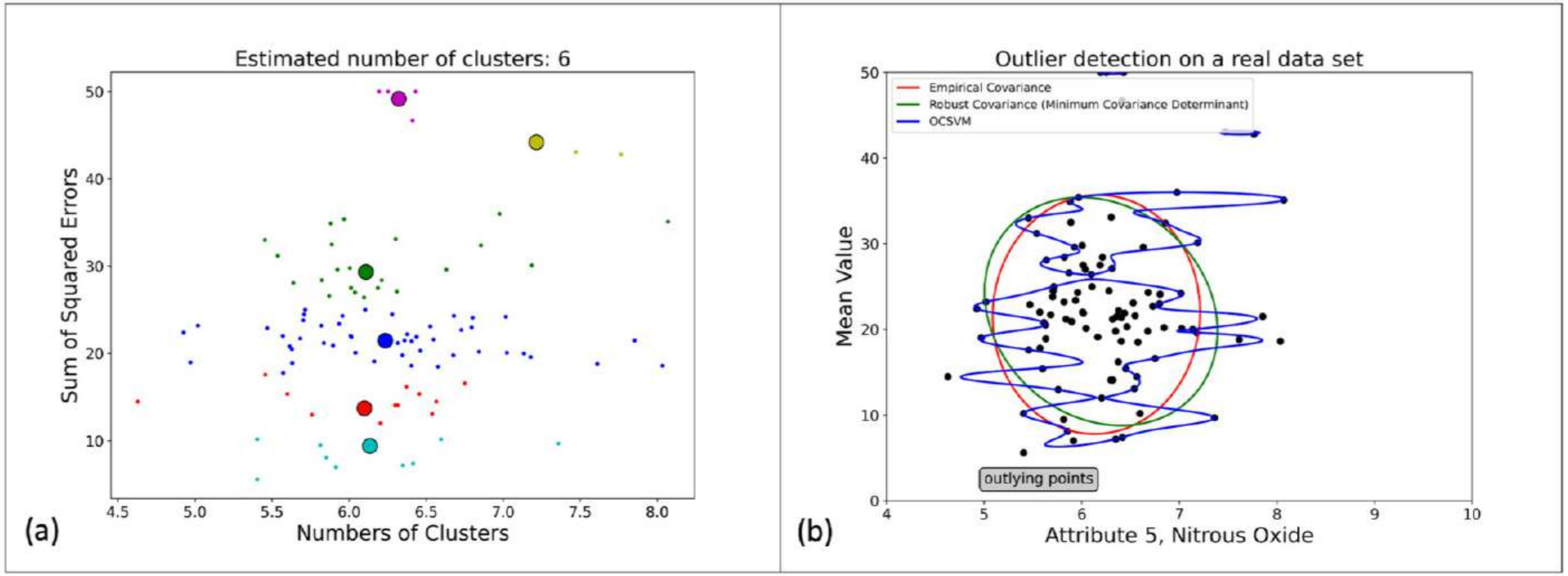
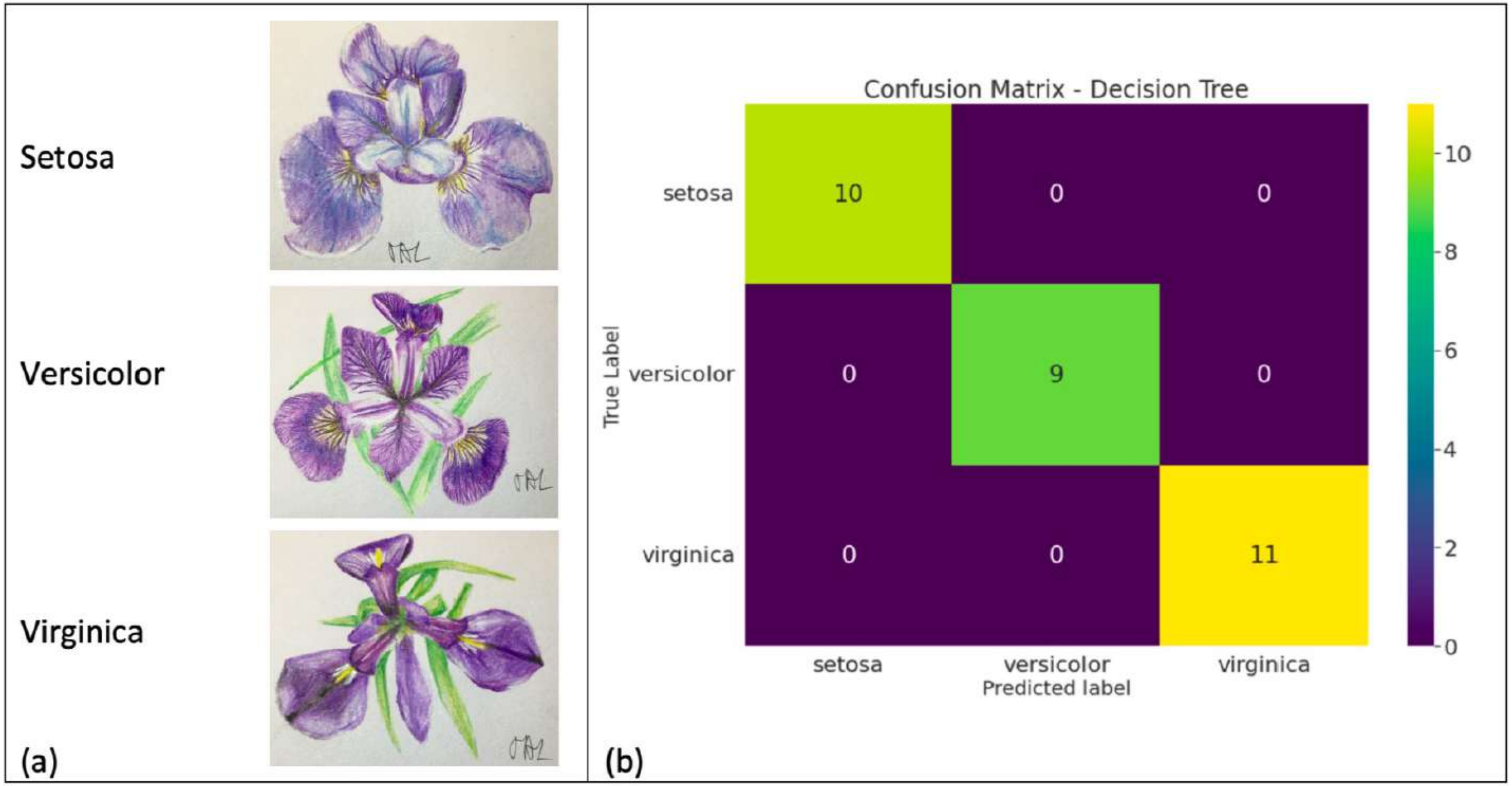
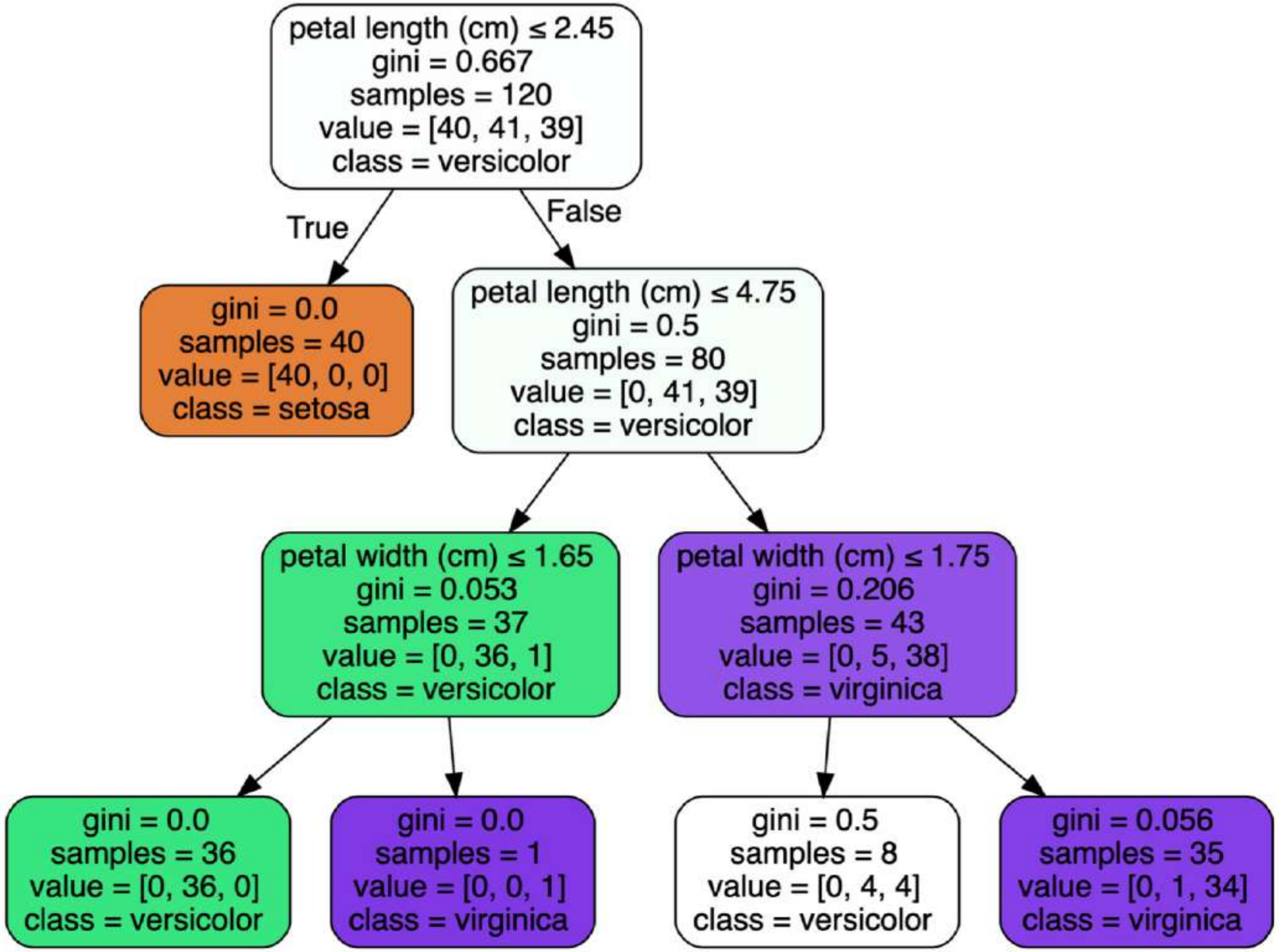


Figure 8.4 (a) Six cluster sets from the mean shift clustering algoritm. (b) Envelope curves for empirical covariance, robust covariance, and OCSVM, to detect rare events.

Data Science: Decision Trees: Predicting Species of Iris



Decision Trees: Predicting Species of Iris: Program_8h.py: End Session 3



Economics: Cobb Douglas Quantity of Production Model: Session 4

Microeconomics: Minimize cost and maximize quantity of product.

Consider a manufacturer of steel cans:

Quantity of product:

$$Q(L,K) = 200L^{\frac{2}{3}}K^{\frac{1}{3}}$$

Cost of construction:

$$C(L,K) = 20L + 750K,$$

where L is labour and K is capital.

Use Lagrange multipliers.

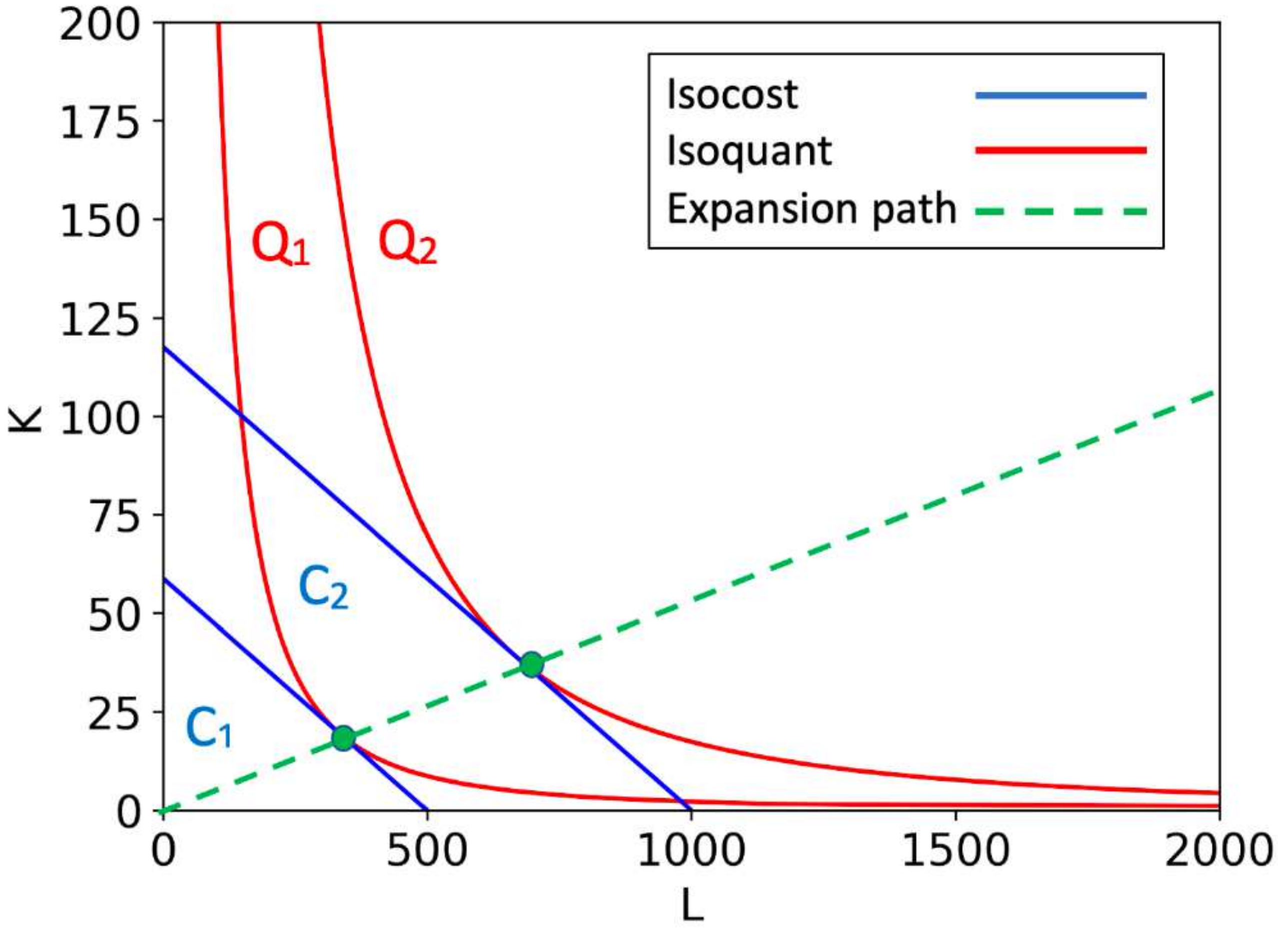


Fig. Isocost-isoquant graph.

Economics: Cobb Douglas Quantity of Production Model

```
# Program_9a.py: Cobb-Douglas Model of Production.
import numpy as np
import matplotlib.pyplot as plt
from sympy import symbols , diff , solve
L,K,lam=symbols("L K lam")
Lmax , Kmax  = 2000 , 200
w , r = 20 , 170
Y = 200*L**(2/3)*K**(1/3)
C=10000
Lagrange=Y-lam*(w*L+r*K-C)
L1 = diff(Lagrange,L)
L2 = diff(Lagrange,K)
L3 = w*L+r*K-C
sol=solve([L1,L2,L3],L,K,lam)
Y1 = 200*sol[0][0]**(2/3)*sol[0][1]**(1/3)
C=20000
Lagrange=Y-lam*(w*L+r*K-C)
L1 = diff(Lagrange,L)
L2 = diff(Lagrange,K)
L3 = w*L+r*K-C
sol=solve([L1,L2,L3],L,K,lam)
Y2 = 200*sol[0][0]**(2/3)*sol[0][1]**(1/3)
Llist = np.linspace(0,Lmax, 1000)
Klist = np.linspace(0, Kmax, 120)
L, K = np.meshgrid(Llist, Klist)
plt.figure()
Z = 200*L**(2/3)*K**(1/3)
plt.contour(L,K,Z,[Y1,Y2],colors="red")
Z = 20*L+170*K
plt.contour(L,K,Z,[10000,20000],colors="blue")
plt.xlabel("L",fontsize=15)
plt.ylabel("K",fontsize=15)
plt.tick_params(labelsize=15)
plt.show()
```

Economics: Solow-Swan Exogenous Growth Model: Program_9b.py

Macroeconomics

The ODE is:

$$\frac{dk}{dt} = s\big(f(k(t))\big) - (n + \delta + g)k(t),$$

where k is capital intensity, s is the savings rate, n is population growth, δ is deprecation and g is technological progress.

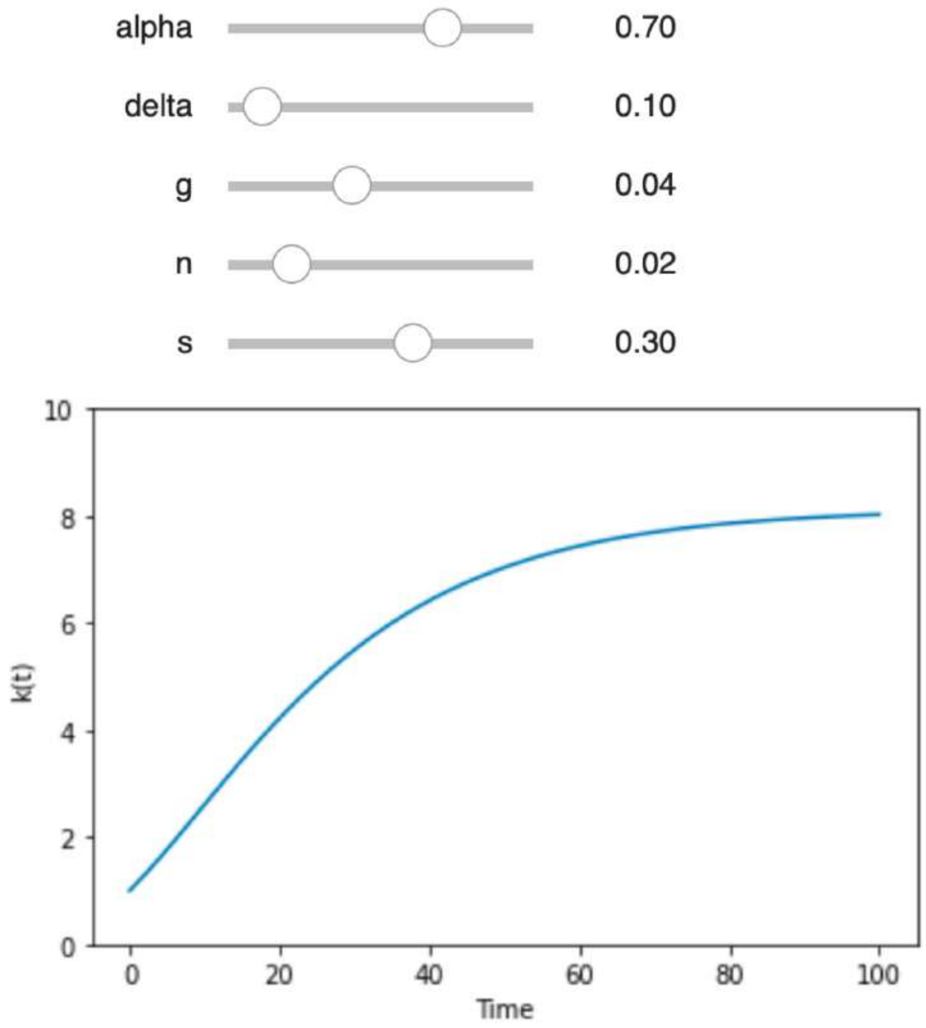
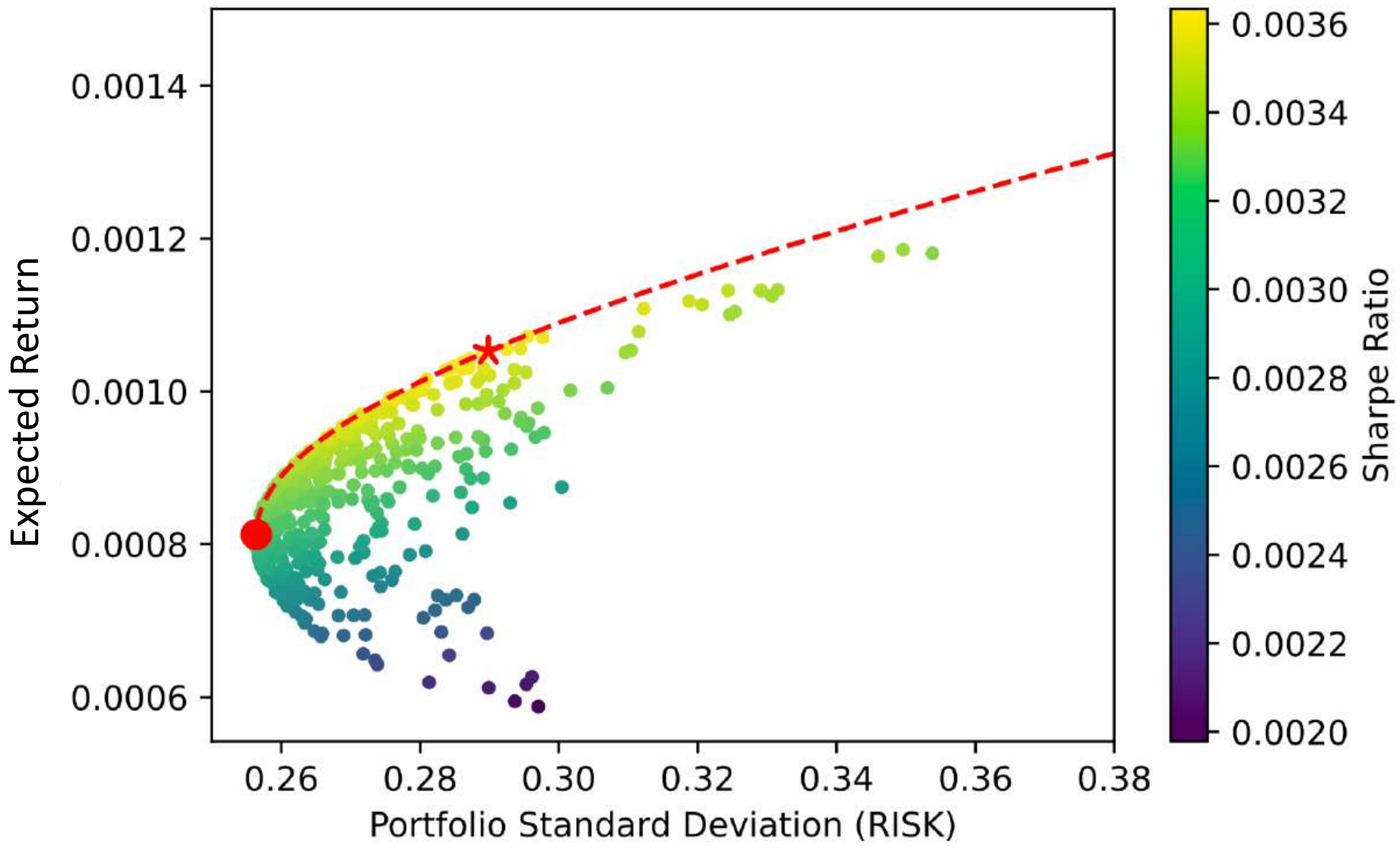


Fig. Interactive plot.

Economics: Modern Portfolio Theory: Program_9c.py

Load financial datasets using Yahoo Finance. The three stocks Apple, Caterpillar and Google are used in this example.



- Pension fund manager – minimize risk
- Hedge fund manager – best risk-reward combination.

Economics: The Black-Scholes Model: Program_9d.py: End of Session 4

```
# Program_9d.py: Black-Scholes Option Prices for Call/Put.
# Computing the Black-Scholes Greeks.
import numpy as np
from scipy.stats import norm
# Parameters: r=interest rate,S=underlying price ($),Strike price ($),
#T=240/365 days, sigma=volatility, C=CALL, P=PUT.
r , S , K , T , sigma  = 0.01 , 30 , 40, 240/365 , 0.3
def Black_Scholes(r,S,K,T,sigma,type="C"):
  d1 = (np.log(S/K)+(r+sigma**2/2)*T)/(sigma*np.sqrt(T))
  d2 = d1 - sigma*np.sqrt(T)
  try:
    if type=="C":
      price=S*norm.cdf(d1,0,1)-K*np.exp(-r*T)*norm.cdf(d2,0,1)
      delta_calc=norm.cdf(d1,0,1)
      gamma_calc=norm.pdf(d1,0,1)/(S*sigma*np.sqrt(T))
      vega_calc=S*norm.pdf(d1,0,1)*np.sqrt(T)*0.01
      theta_calc=(-S*norm.pdf(d1,0,1)*sigma/(2*np.sqrt(T))-\
                  r*K*np.exp(-r*T)*norm.cdf(d2,0,1)) / 365
      rho_calc=K*T*np.exp(-r*T)*norm.cdf(d2,0,1)*0.01
    elif type=="P":
      price=K*np.exp(-r*T)*norm.cdf(-d2,0,1)-S*norm.cdf(-d1,0,1)
      delta_calc=-norm.cdf(-d1,0,1)
      gamma_calc=norm.pdf(d1,0,1)/(S*sigma*np.sqrt(T))
      vega_calc=S*norm.pdf(d1,0,1)*np.sqrt(T) * 0.01
      theta_calc=(-S*norm.pdf(d1,0,1)*sigma/(2*np.sqrt(T))-\
                  r*K*np.exp(-r*T)*norm.cdf(-d2,0,1)) / 365
      rho_calc=-K*T*np.exp(-r*T)*norm.cdf(-d2,0,1) * 0.01
    return [price,delta_calc,gamma_calc,vega_calc,theta_calc,rho_calc]
  except:
    print("Please input correct parameters")
BS_Call=Black_Scholes(r,S,K,T,sigma,type="C")
BS_Put=Black_Scholes(r,S,K,T,sigma,type="P")
print("r=",r,"S=",S,"K=",K,"T=",T,"sigma=",sigma)
print("Option CALL price is: ", round(BS_Call[0],2))
print("Option PUT price is: ", round(BS_Put[0],2))
print("delta Call is: ", round(BS_Call[1],4))
print("delta Put is: ", round(BS_Put[1],4))
print("gamma Call/Put is: ", round(BS_Call[2],4))
print("vega Call/Put is: ", round(BS_Call[3],4))
print("theta Call is: ", round(BS_Call[4],4))
print("theta Put is: ", round(BS_Put[4],4))
```

The Black-Scholes partial differential equation describing the price of a European call or put option over time.

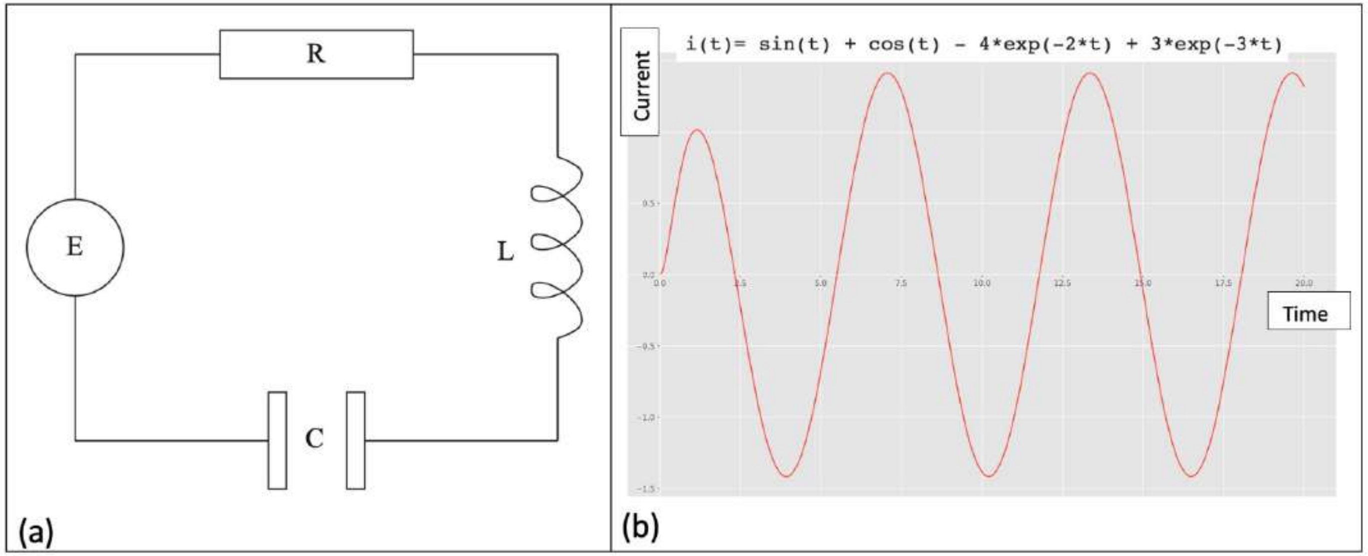
$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0$$

where V is the price of the option as a function of stock price S and time t, r is risk-free interest rate, and σ is the volatility of the stock.

Manchester Metropolitan University

Engineering: Linear Electric Circuits: Resistor-Inductor-Capacitor: Session 5

```
# Program_10a.py: Current in a Resistor-Inductor-Capacitor circuit.
from sympy import symbols , diff , Eq , Function , dsolve , cos , plot
from matplotlib import style
t=symbols("t")
i=symbols("i",cls=Function)
deqn1=Eq(i(t).diff(t,t) + 5*i(t).diff(t) + 6*i(t), 10*cos(t))
odesol1=dsolve(deqn1, i(t),ics={i(0): 0, diff(i(t), t).subs(t,0): 0})
print("i(t)=",odesol1.rhs)
style.use("ggplot")
plot(odesol1.rhs , (t , 0 , 20),xlabel = "Time", ylabel = "Current")
```



Electric Circuits: The Memristor: Program_10b.py

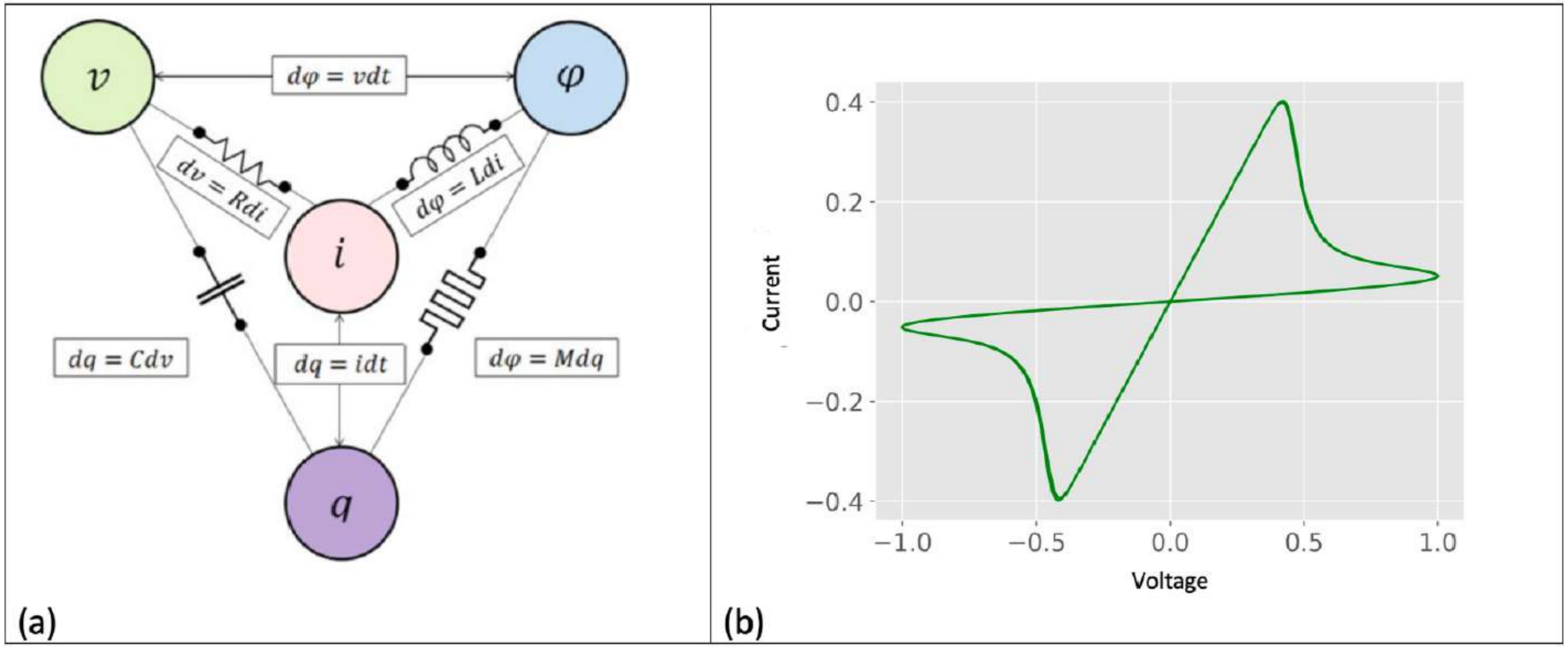


Figure 10.2 (a) Relations between voltage v, flux ϕ, current i, and charge q. (b) Pinched hysteresis curve when $\omega_0 = 0.6$.

Nonlinear Electric Circuits: Chua’s Circuit: Program_10c.py

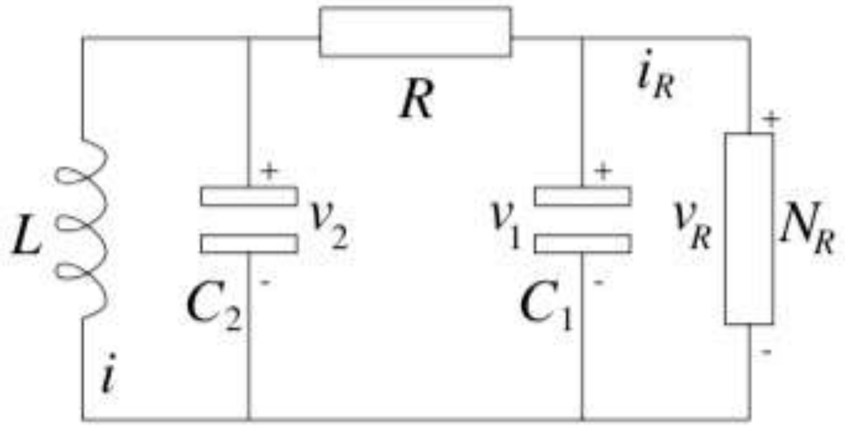


Figure 8.13: Chua’s electric circuit.

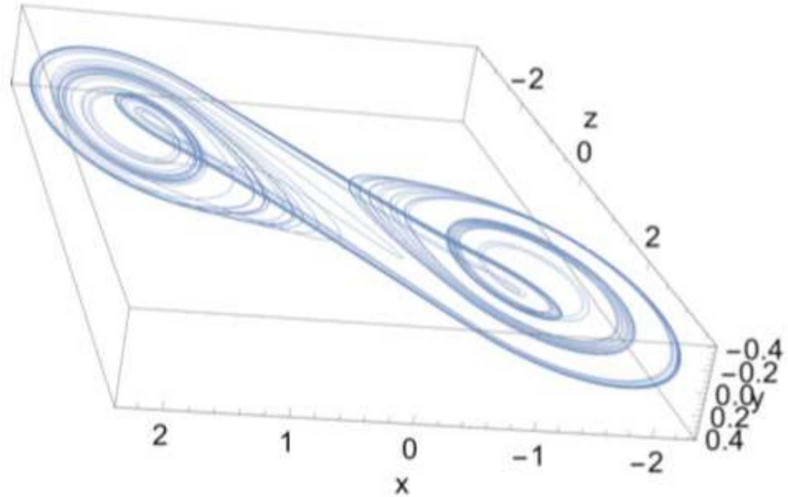
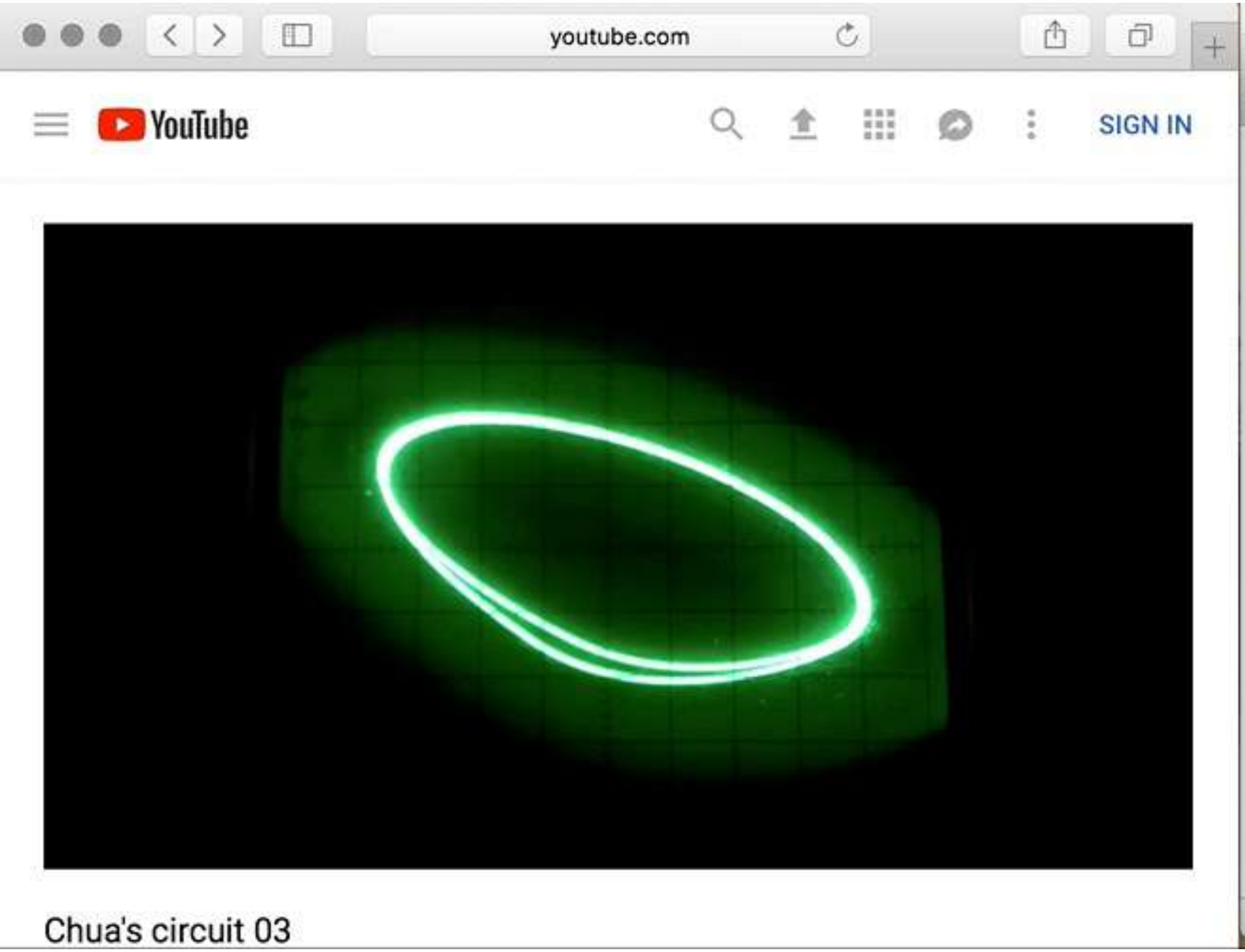
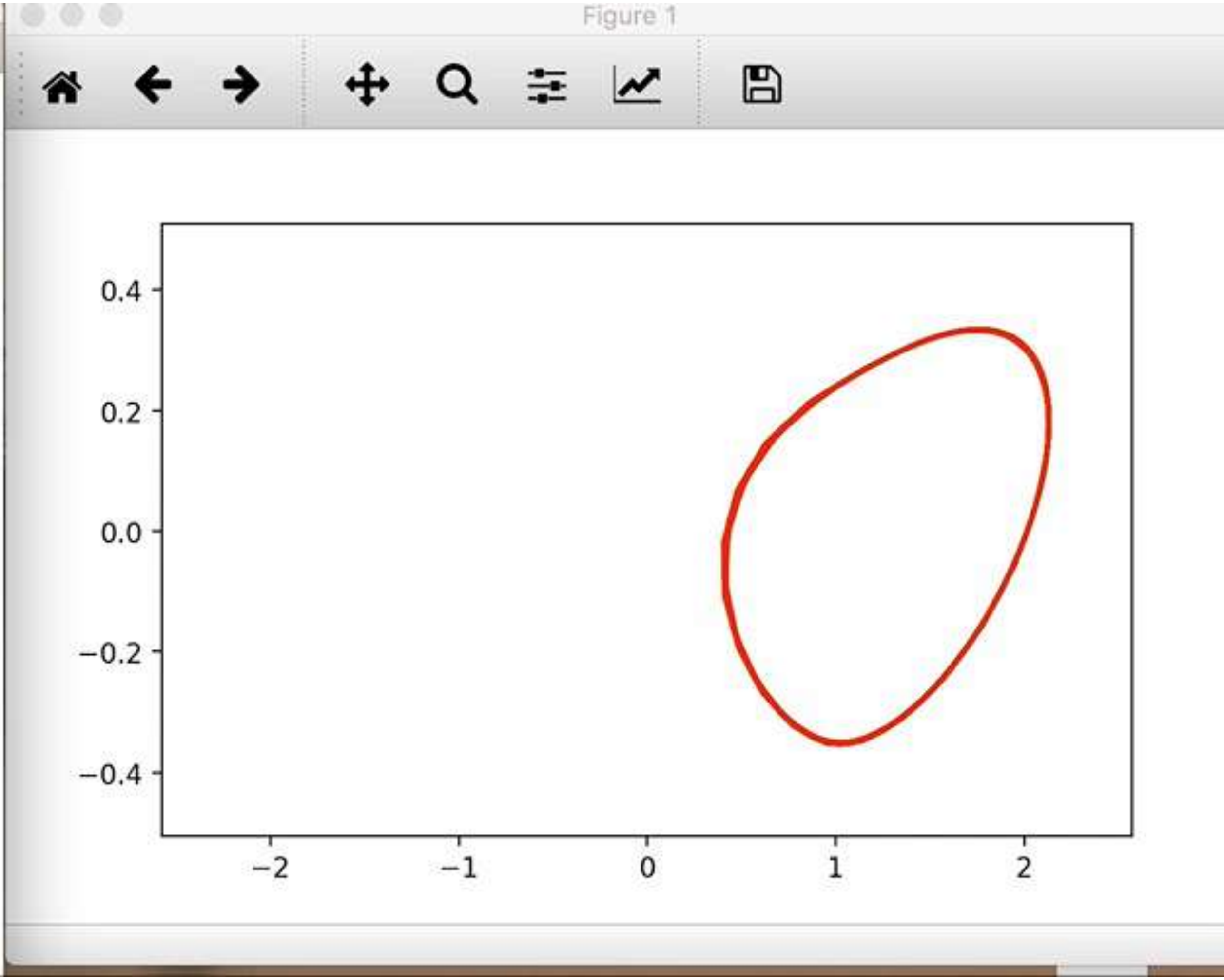


Figure 8.14: [Python animation] Chua’s double-scroll attractor: Phase portrait for system (8.9) when $a = 15, b = 25.58, c = -5/7$, and $d = -8/7$. The initial conditions are $x(0) = -1.6$, $y(0) = 0$, and $z(0) = 1.6$.

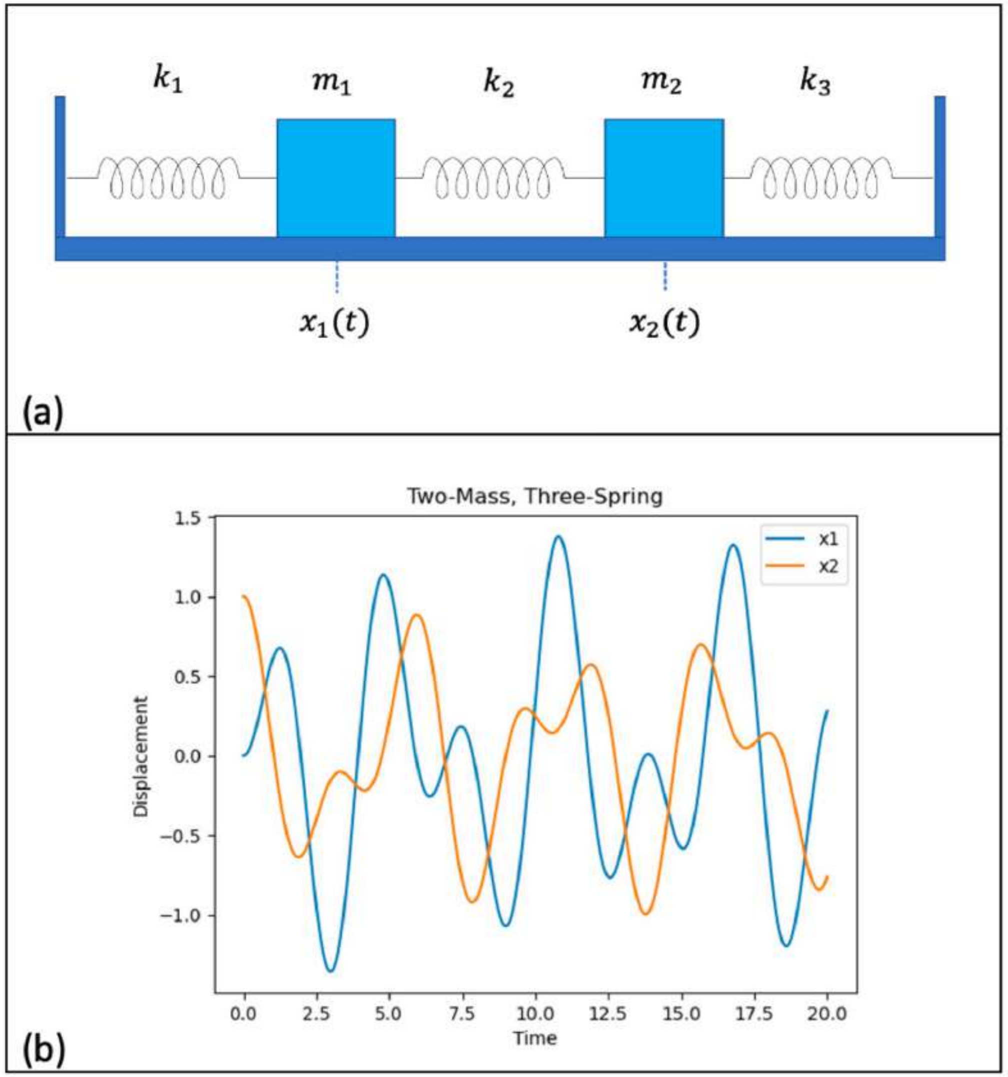
Chua Circuit Animation



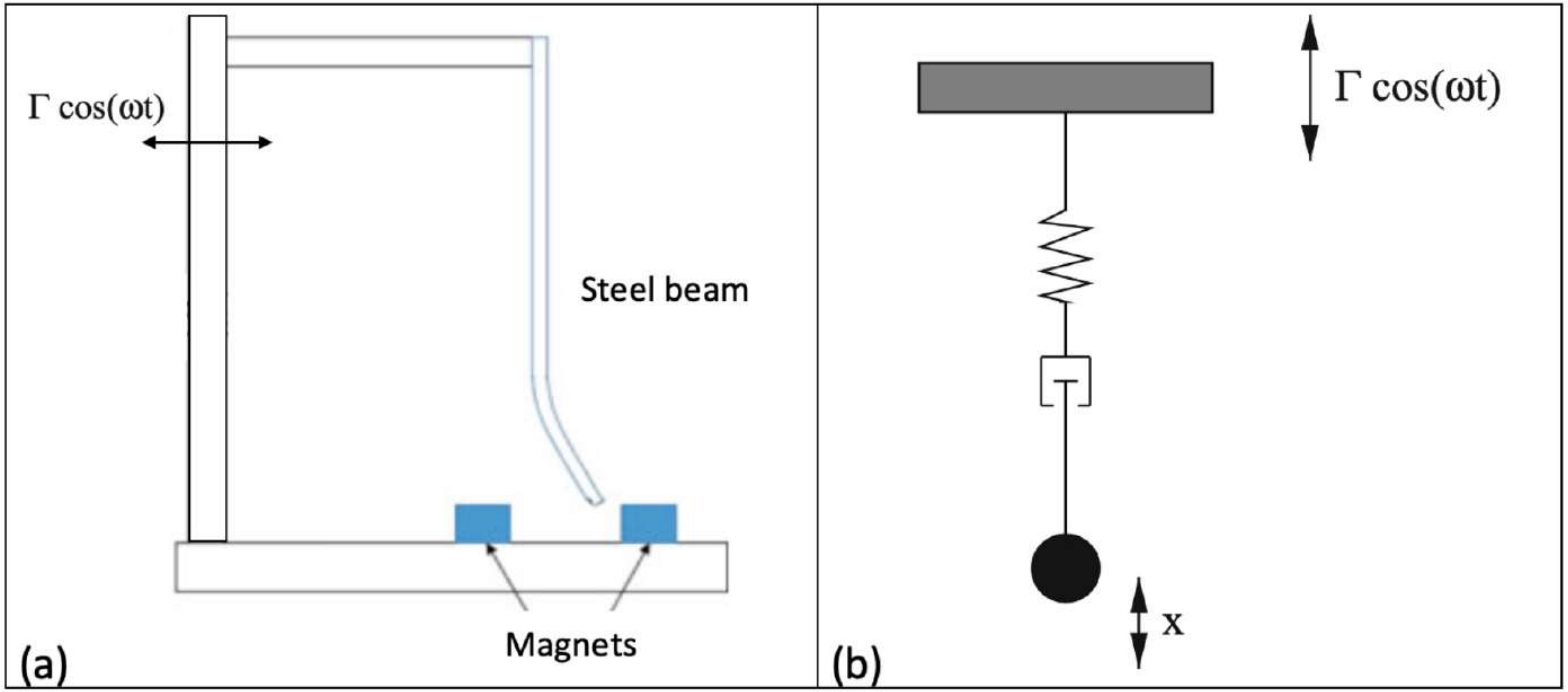


Engineering: Coupled Oscillators: Program_10d.py

$$\dot{x}_1 = y_1,$$
$$\dot{y}_1 = -\frac{1}{m_1}\left(k_1 x_1 + k_2\left(x_1 - x_2\right)\right),$$
$$\dot{x}_2 = y_2,$$
$$\dot{y}_2 = \frac{1}{m_2}\left(k_2\left(x_2 - x_1\right) + k_3 x_2\right),$$



Engineering: Periodically Forced Systems: Program_10e.py



Engineering: Periodically Forced Systems: Bifurcation Diagram

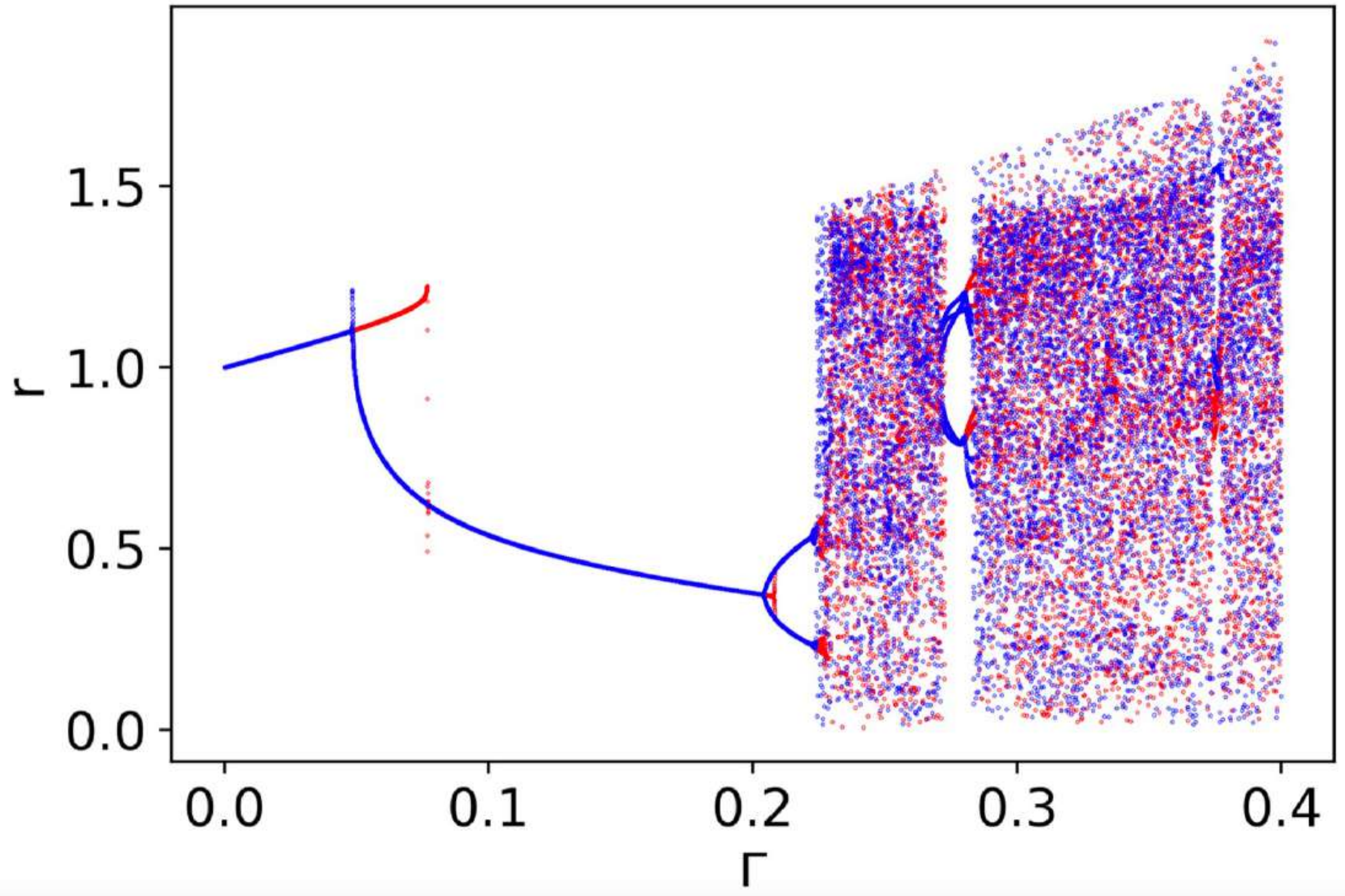


Fig. Bifurcation diagram showing a clockwise hysteresis loop as the amplitude of forcing is increased and decreased.

Hysteresis in the Periodically Driven Two-Bar Linkage

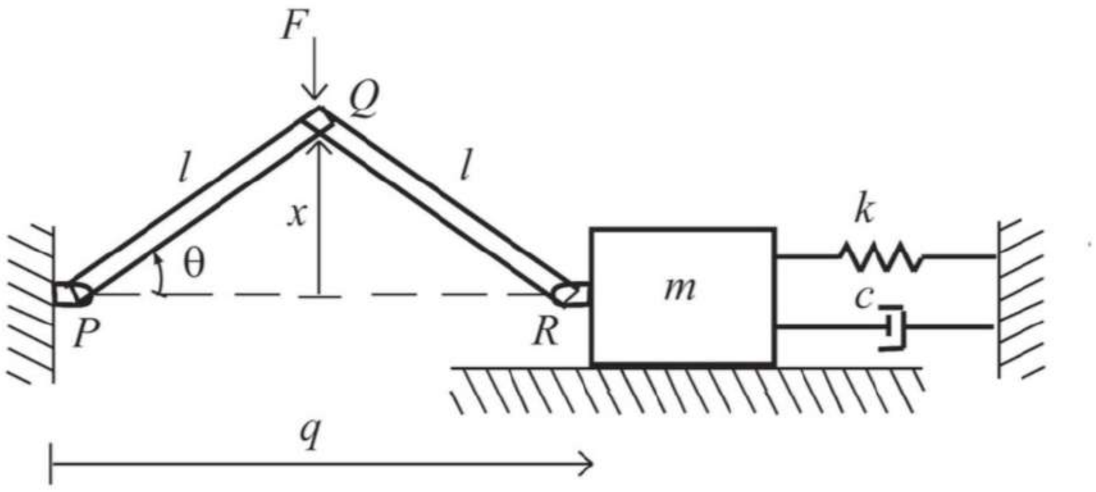
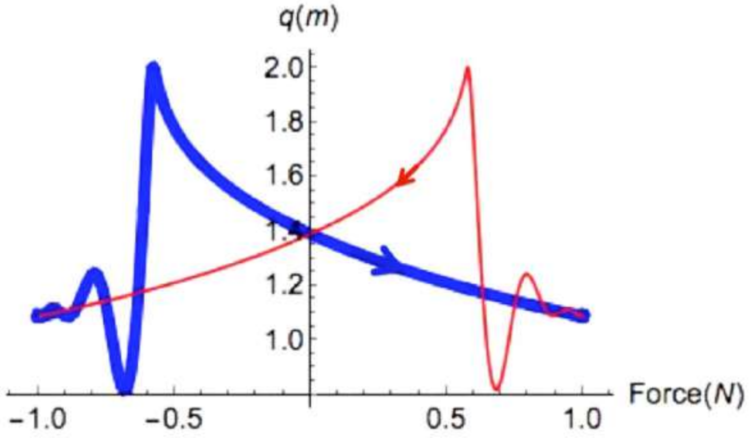
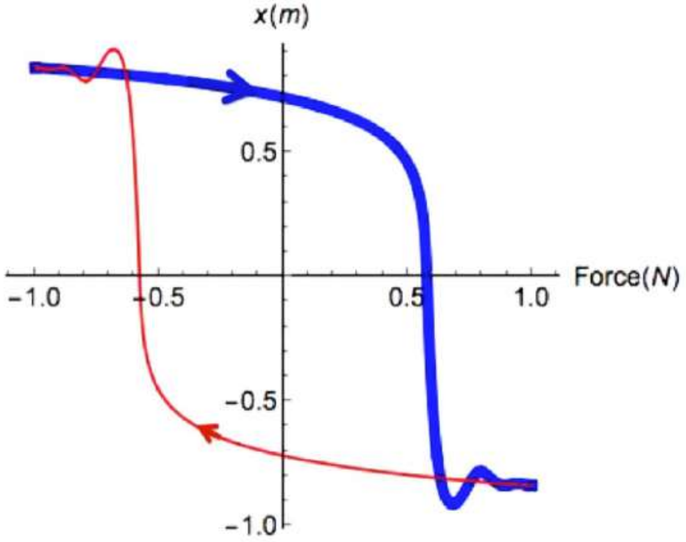


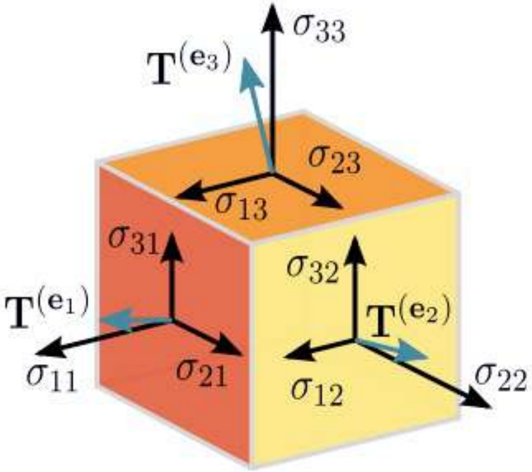
Figure: The preloaded two-bar linkage with a periodic force F acting at the joint Q. As the point Q moves vertically up and down, the mass m moves horizontally left and right.



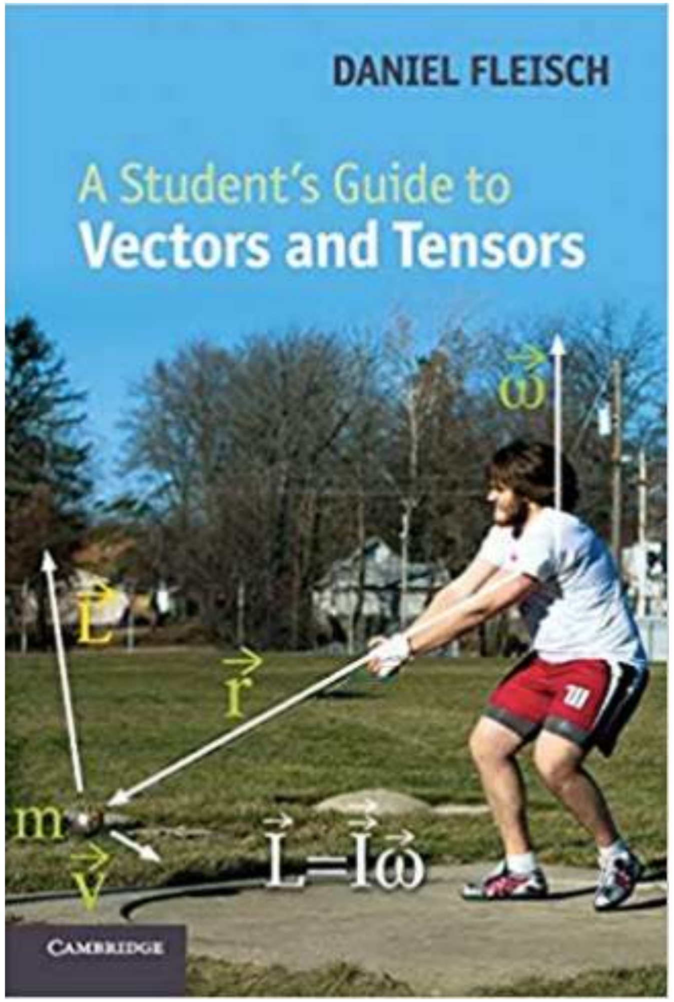


Tensors

Definition: An n'th rank tensor in m-dimensional space is a mathematical object that has n indices and m^n components and obeys certain transformation rules.



Cauchy stress tensor.



Tensors have applications in Riemannian geometry, mechanics, elasticity, theory of relativity, electromagnetic theory and artificial intelligence, for example.

Tensors in Artificial Intelligence

Definition: An n'th rank tensor in m-dimensional space is a mathematical object that has n indices and m^n components and obeys certain transformation rules.

3 is a scalar and rank 0 tensor

[1,2,3] is a vector and rank 1 tensor

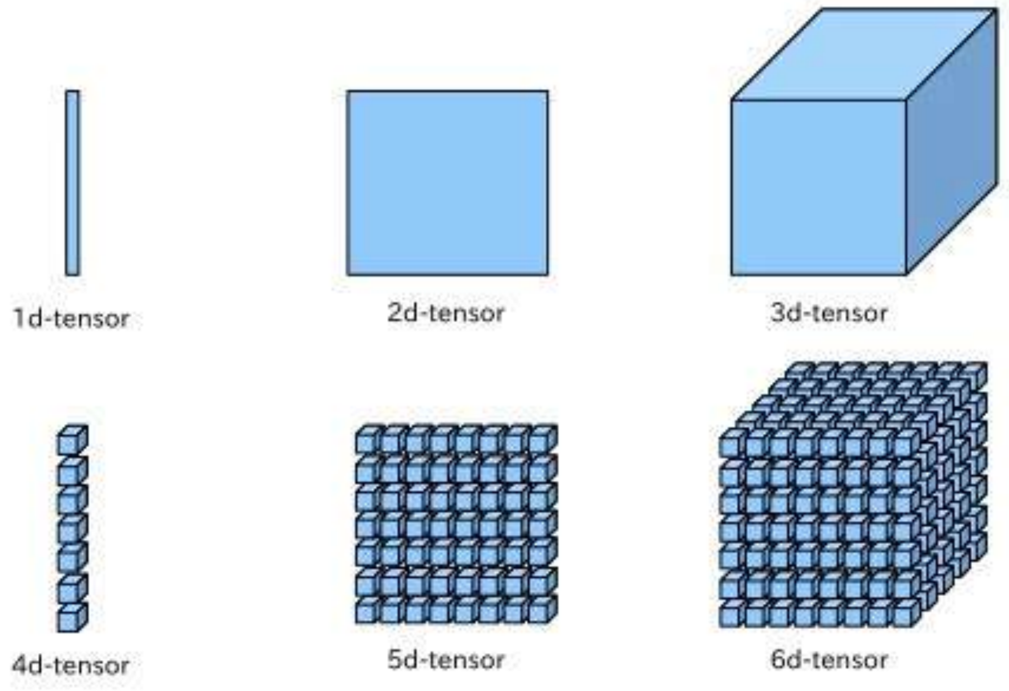
[[1,2,3],[4,5,6],[7,8,9]] is an array and rank 2 tensor

[[[1,2],[3,4]],[[5,6],[7,8]]] is an array and rank 3 tensor

In Deep Learning:

Images are represented by tensors of rank 4

Videos are represented by tensors of rank 5



Tensors and Python

```
In [1]: import numpy as np

In [2]: A = np.array([1,2,3,4])

In [3]: A.shape
Out[3]: (4,)

In [4]: A.ndim
Out[4]: 1
```

```
In [1]: import numpy as np

In [2]: T = np.array([
   ...: [[1,2,3],[4,5,6],[7,8,9]],
   ...: [[11,12,13], [14,15,16], [17,18,19]],
   ...: [[21,22,23], [24,25,26], [27,28,29]],
   ...:   ])

In [3]: T.shape
Out[3]: (3, 3, 3)

In [4]: T.ndim
Out[4]: 3
```

Tensors and Python: End Session 5

There are a number of ways to multiply tensors, for example:

```
In [1]: import numpy as np

In [2]: A = np.array([[1,2],[3,4]])

In [3]: B = np.array([[5,6],[7,8]])

In [4]: A * B
Out[4]:
array([[ 5, 12],
       [21, 32]])


In [5]: C=np.tensordot(A,B, axes=1)

In [6]: print(C)
[[19 22]
 [43 50]]
```

```
In [7]: C=np.tensordot(A,B, axes=0)

In [8]: print(C)
[[[[ 5  6]
   [ 7  8]]

  [[10 12]
   [14 16]]]


 [[[15 18]
   [21 24]]

  [[20 24]
   [28 32]]]]

In [9]: C.shape
Out[9]: (2, 2, 2, 2)

In [10]: C.ndim
Out[10]: 4
```

Day 2 Summary

Day 2			
Jupyter and Colab Notebooks	10am-11am	Economics	2pm-3pm
Biology and Chemistry	11am-12pm	Engineering	3pm-4pm
Data Science	12pm-1pm		

Download all files from GitHub:

https://github.com/proflynch/CRC-Press/

Solutions to the Exercises in Section 2:

https://drstephenlynch.github.io/webpages/Solutions_Section_2.html

python™

Manchester Metropolitan University