

5-day Hands-on Workshop on:

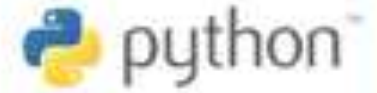
Python for Scientific Computing and TensorFlow for Artificial Intelligence

By Dr Stephen Lynch FIMA SFHEA

Holder of Two Patents

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

STEM Ambassador and Speaker for Schools

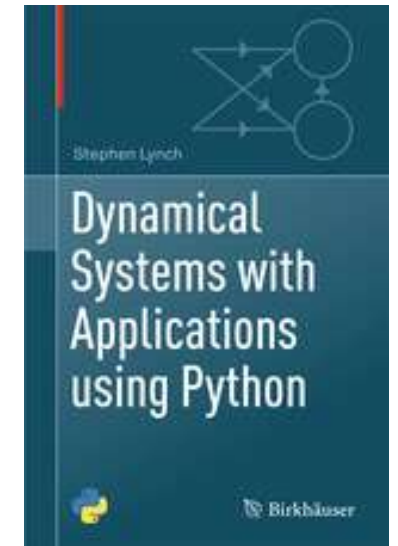


s.lynch@mmu.ac.uk

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

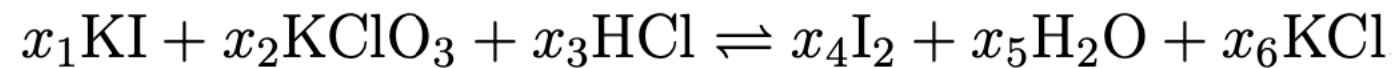
Schedule (Day 3): Start Session 1

Day 3			
Topics	Hours	Topics	Hours
Scientific Computing: Chemical Kinetics	10am-11am	Scientific Computing: Engineering	1pm-2pm
Scientific Computing: Fractals and Multifractals	11am-12pm	Scientific Computing: Physics	2pm-3pm



Chemistry is the branch of science that deals with properties, composition and structure of elements and compounds, how they change, and the energy that is released or absorbed when they change.

Balancing chemical reactions using the matrix null space method:



Element	KI	KClO ₃	HCl	I ₂	H ₂ O	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

Table: Chemical Composition Table.

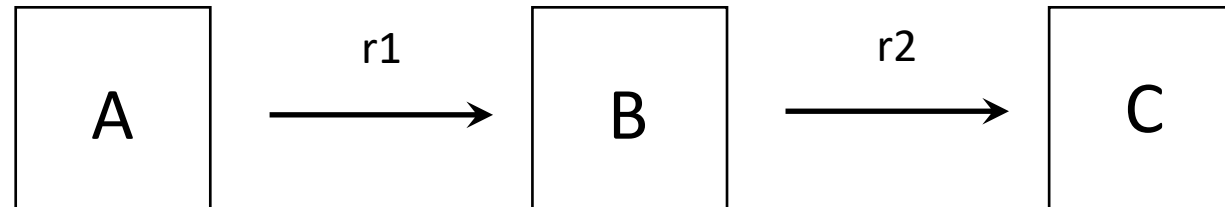
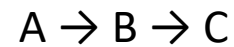
Balancing Chemical Equations

```
# Program New Book: 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.
```

Chemical Kinetics: A Simple Model

Definition: *The Chemical Law of Mass Action* – The rates at which the concentrations of the various chemical species change with time are proportional to their concentrations.

Example: Consider the simple chemical reaction in which A changes to B with respect to the amount of substance A present (r_1 say) and B changes to C with respect to the amount of B present (r_2 say):



Chemical Kinetics: A Simple Model

```
# Program 02h: Chemical kinetics - conservation of mass.
# See Exercise 7, the reaction: A -> B -> C.
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()
```

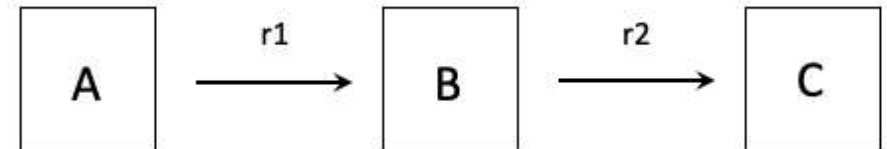
Compartmental Model

$|A| = x, |B| = y, |C| = z$

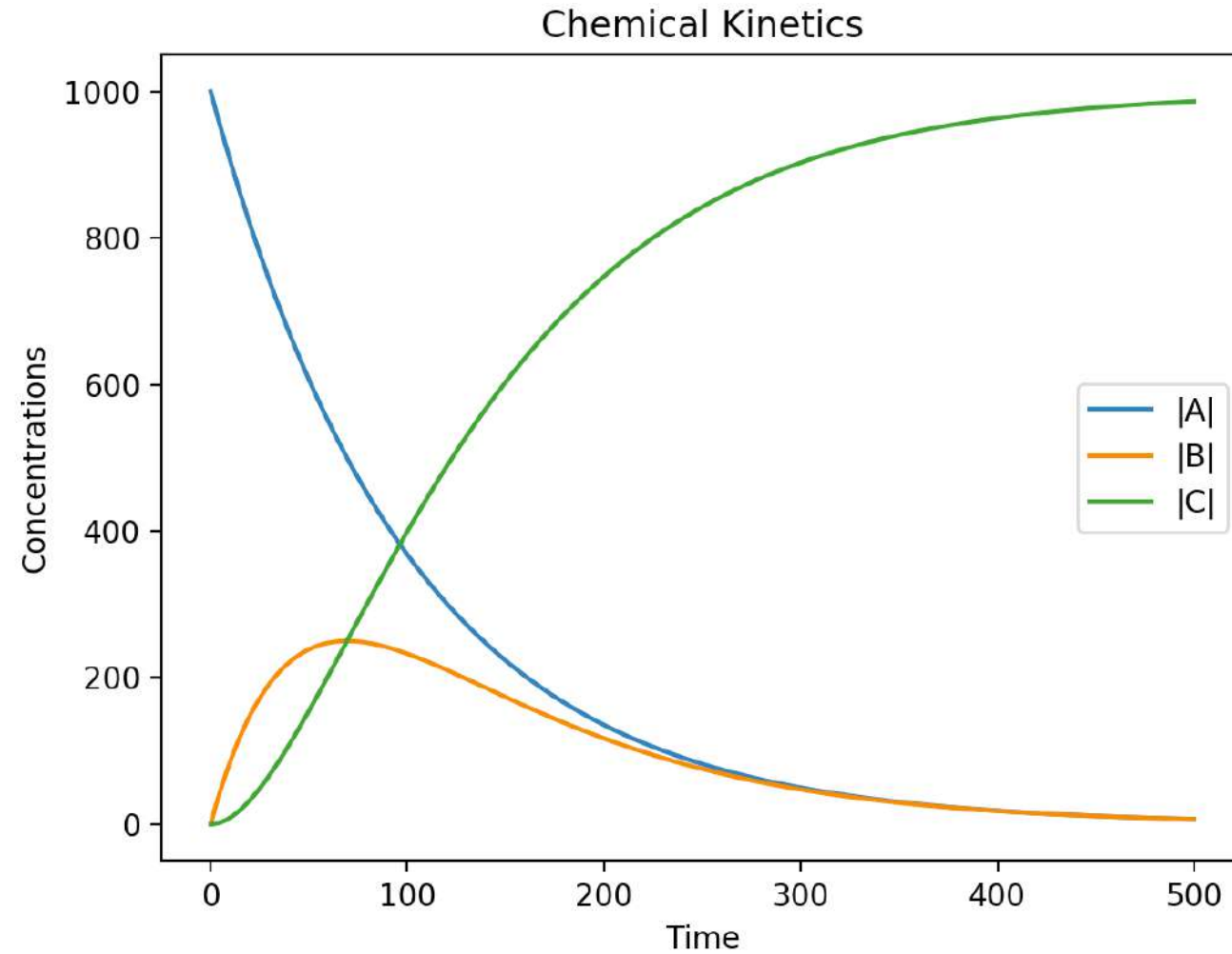
$$\frac{dx}{dt} = -r_1 x$$

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

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

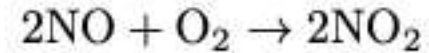


Chemical Kinetics: A Simple Model



Chemical Kinetics: Example 10, Chapter 2

Example 10 The chemical equation for the reaction between nitrous oxide and oxygen to form nitrogen dioxide at $25^{\circ}C$,



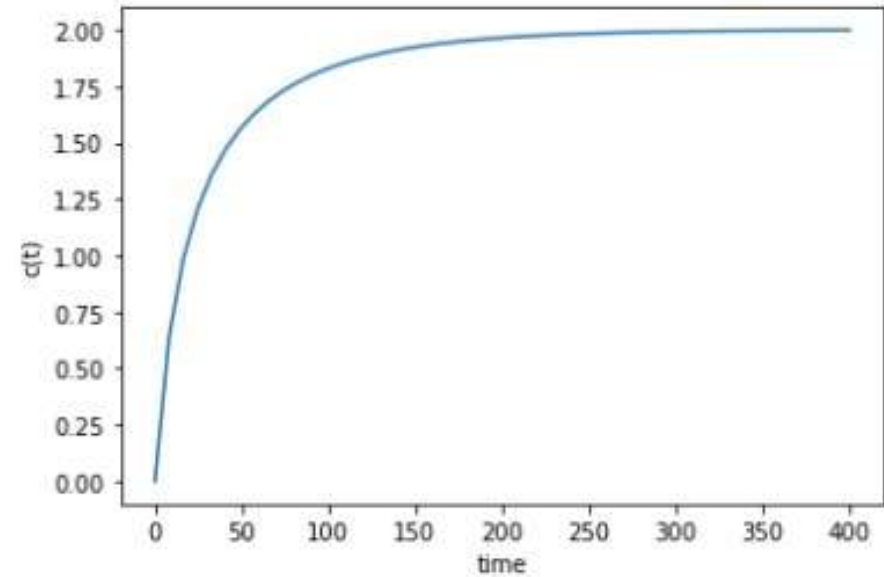
obeys the law of mass action. The rate equation is given by

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

where $c = [\text{NO}_2]$ is the concentration of nitrogen dioxide, k is the rate constant, a_0 is the initial concentration of NO, and b_0 is the initial concentration of O_2 . Find the concentration of nitrogen dioxide after time t given that $k = 0.00713 \text{ l}^2 \text{ M}^{-2} \text{ s}^{-1}$, $a_0 = 4 \text{ M l}^{-1}$, $b_0 = 1 \text{ M l}^{-1}$, and $c(0) = 0 \text{ M l}^{-1}$.

Chemical Kinetics: A Simple Example

```
1  # Production of Nitrogen Dioxide.
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from scipy.integrate import odeint
5  k, a0, b0, c0 = 0.00713, 4, 1, 0
6  def ode(c, t):
7      dcdt = k * (a0 - c)**2 * (b0 - c / 2)
8      return dcdt
9  # Solve the ODE numerically.
10 t = np.linspace(0, 400, 401)   # t = [0, 1, 2, ..., 400].
11 c = odeint(ode, c0, t)
12 plt.xlabel('Time')
13 plt.ylabel('c(t)')
14 plt.title('Production of Nitrogen Dioxide')
15 plt.plot(t, c)
16 plt.show()
17 # Find c when t=100.
18 print('c(100)= ', c[100])
```

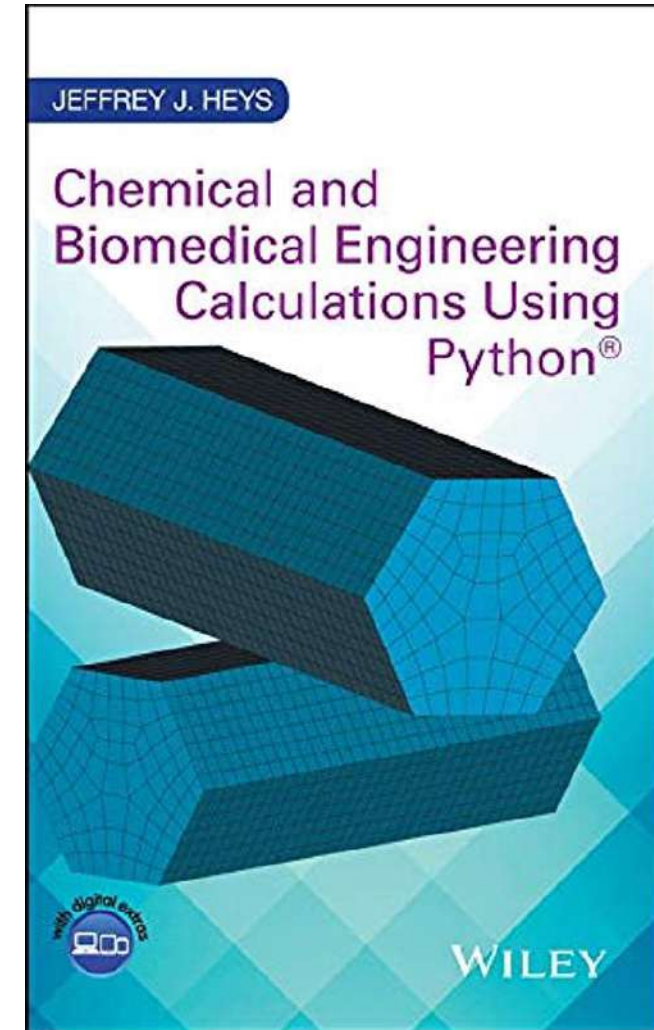


```
In [1]: runfile('/Users/slynch/
Documents/Stephen/MMU Python Workshop/
Slides/Python Programs/
NO_2_Production.py', wdir='/Users/
slynch/Documents/Stephen/MMU Python
Workshop/Slides/Python Programs')
c(100)=  [1.828819]
```

Chemical Kinetics: Examples of ODEs and Book on Chemical Engineering

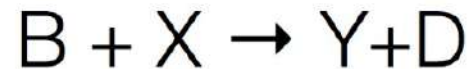
Table 2.1: One of the possible reaction rate equations for each chemical reaction.

Chemical reaction	The reaction rate equation for one species may be expressed as follows:
$A+B \rightarrow C$	$\frac{dc}{dt} = k_f ab = k_f(a_0 - c)(b_0 - c)$
$2A \rightleftharpoons B$	$\frac{db}{dt} = k_f(a_0 - 2b)^2 - k_r b$
$A \rightleftharpoons 2B$	$\frac{db}{dt} = k_f(a_0 - \frac{b}{2}) - k_r b^2$
$A \rightleftharpoons B+C$	$\frac{dc}{dt} = k_f(a_0 - c) - k_r(b_0 + c)(c_0 + c)$
$A+B \rightleftharpoons C$	$\frac{dc}{dt} = k_f(a_0 - c)(b_0 - c) - k_r c$
$A+B \rightleftharpoons C+D$	$\frac{dc}{dt} = k_f(a_0 - c)(b_0 - c) - k_r(c_0 + c)(d_0 + c)$



Chemical Kinetics: The Brusselator Autocatalytic Reaction

The Reactions

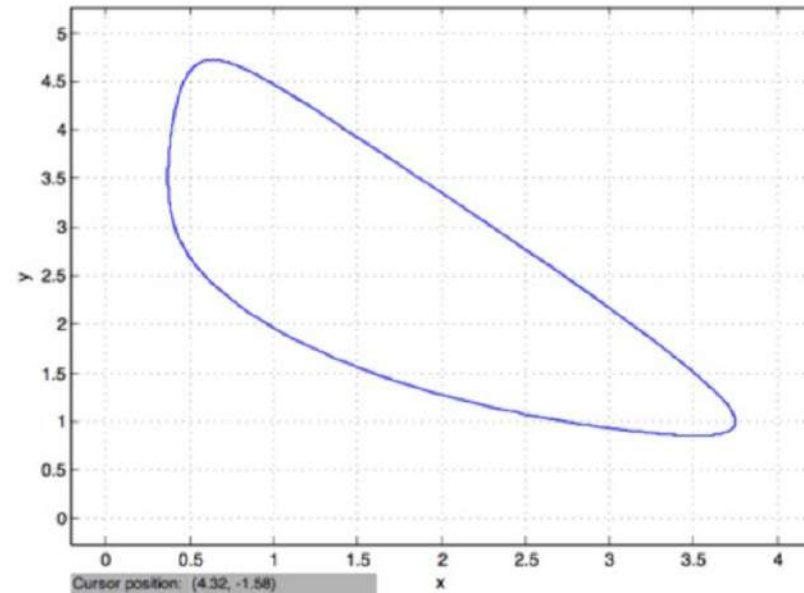


- * A and B are assumed to be constant.
- * D and E are removed from the system as they are produced.
- * Reaction rates are all set to 1.

Scaled Equations

$$\dot{x} = 1 - (1 + b)x + ax^2y$$

$$\dot{y} = bx - ax^2y$$



Chemical Kinetics: The Belousov-Zhabotinski Reaction: End Session 1

00:00

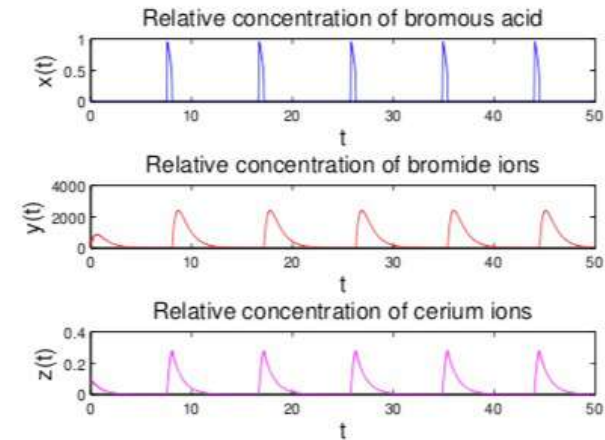


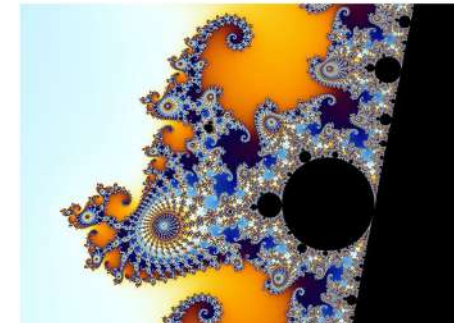
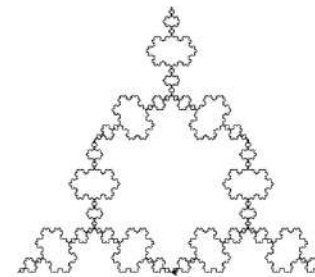
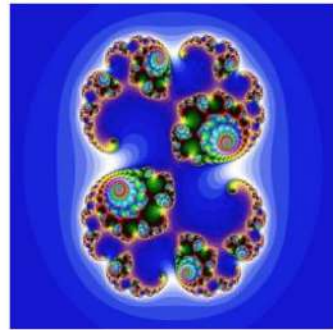
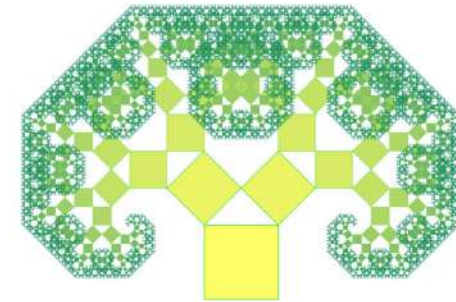
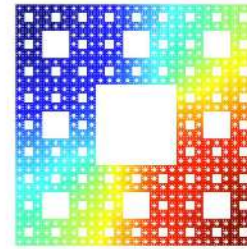
Figure 8.16: [Python] Periodic behaviour for the stiff ODE (8.11) when $X(0) = 0$, $Y(0) = 0$, $Z(0) = 0.1$, $\epsilon_1 = 0.0099$, $\epsilon_2 = 2.4802\text{e-}5$, $q = 3.1746\text{e-}5$, and $C = 1$. Note that the direct physical significance is lost and the graph shows relative concentrations of each of the concentrations of ions.

Chemical Oscillation - Belousov-Zhabotinsky reaction

Fractals and Multifractals: Start Session 2

Definition: A fractal is an image repeated on an ever reduced scale.

Definition: A fractal is an object with non-integer dimension.

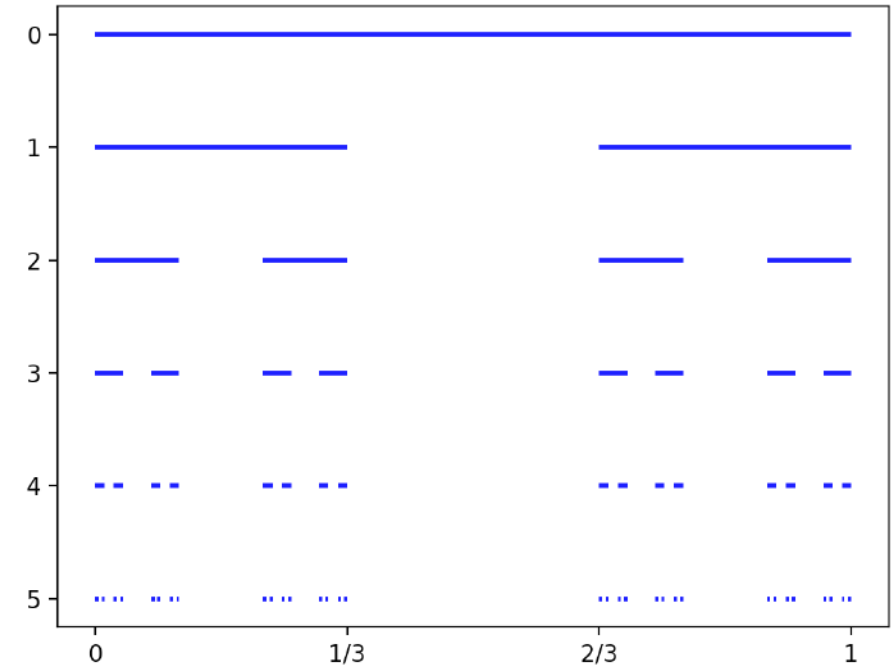


Fractals in Nature

Mathematical Fractals

Fractals: The Cantor Set

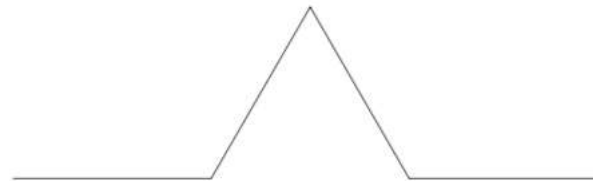
```
1  # The Cantor Set.
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  line, stage = [0, 1], 5
6  def cantor(line, level = 0):
7      plt.plot(line, [level, level], color = "b", lw = 2, \
8              solid_capstyle = "butt")
9      if level < stage:
10         segment = np.linspace(line[0], line[1], 4)
11         cantor(segment[:2], level + 1)
12         cantor(segment[2:], level + 1)
13
14  cantor(line)
15  plt.gca().invert_yaxis()
16  plt.xticks([0, 1/3, 2/3, 1], ['0', '1/3', '2/3', '1'])
17  plt.show()
```



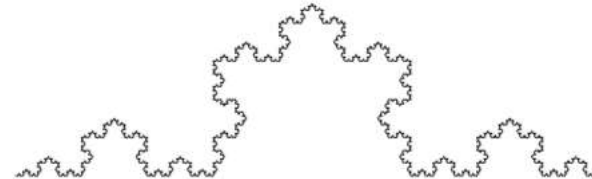
Problem: Edit the program to plot a Cantor set where the two middle fifth segments are removed at each stage.

Fractals: The Koch Curve Fractal

```
1 # Programs 17a: Plotting the Koch curve.
2 # See Figure 17.2.
3
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from math import floor
7
8 k=6
9 n_lines = 4**k
10 h = 3**(-k);
11 x = [0]*(n_lines+1)
12 y = [0]*(n_lines+1)
13 x[0], y[0] = 0, 0
14
15 segment=[0] * n_lines;
16
17 # The angles of the four segments.
18 angle=[0, np.pi/3, -np.pi/3, 0]
19 for i in range(n_lines):
20     m=i
21     ang=0
22     for j in range(k):
23         segment[j] = np.mod(m, 4)
24         m = floor(m / 4)
25         ang = ang + angle[segment[j]]
26
27     x[i+1] = x[i] + h*np.cos(ang)
28     y[i+1] = y[i] + h*np.sin(ang)
29
30 plt.axis('equal')
31 plt.plot(x,y)
32 plt.show()
```

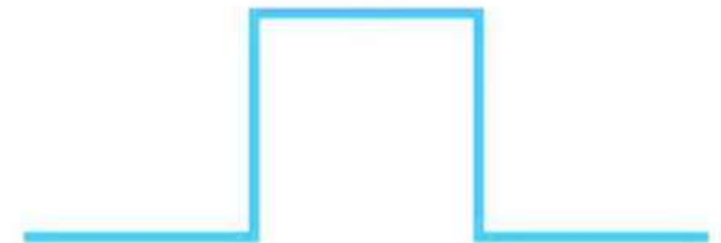


Stage 1



Stage 5

Problem: Edit the program to plot a Koch square fractal.



MATLAB Program to Plot Barnsley's Fern

```
1 # Program 17c: Barnsley's fern.
2 # See Figure 17.7.
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import matplotlib.cm as cm
6
7 # The transformation T
8 f1 = lambda x, y: (0.0, 0.2*y)
9 f2 = lambda x, y: (0.85*x + 0.05*y, -0.04*x + 0.85*y + 1.6)
10 f3 = lambda x, y: (0.2*x - 0.26*y, 0.23*x + 0.22*y + 1.6)
11 f4 = lambda x, y: (-0.15*x + 0.28*y, 0.26*x + 0.24*y + 0.44)
12 fs = [f1, f2, f3, f4]
13
14 num_points = 60000
15
16 width = height = 300
17 fern = np.zeros((width, height))
18
19 x, y = 0, 0
20 for i in range(num_points):
21     # Choose a random transformation
22     f = np.random.choice(fs, p=[0.01, 0.85, 0.07, 0.07])
23     x, y = f(x,y)
24     # Map (x,y) to pixel coordinates
25     # Center the image
26     cx, cy = int(width / 2 + x * width / 10), int(y * height / 10)
27     fern[cy, cx] = 1
28
29 fig, ax=plt.subplots(figsize=(8,8))
30 plt.imshow(fern[::-1,:], cmap=cm.Greens)
31 ax.axis('off')
32 plt.show()
```



Multifractal Cantor Set

The Cantor multifractal: $N(\varepsilon) = 2, \varepsilon = \frac{1}{3}$.

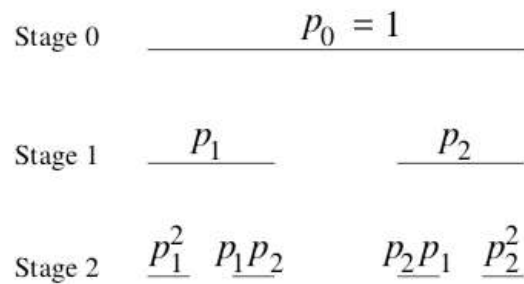


Figure 17.14: The weight distribution on a Cantor multifractal set up to stage 2.

$$\tau = \frac{\ln(p_1^q + p_2^q)}{\ln(3)}, \quad \alpha = -\frac{d\tau}{dq}, \quad f(\alpha) = \alpha q + \tau.$$

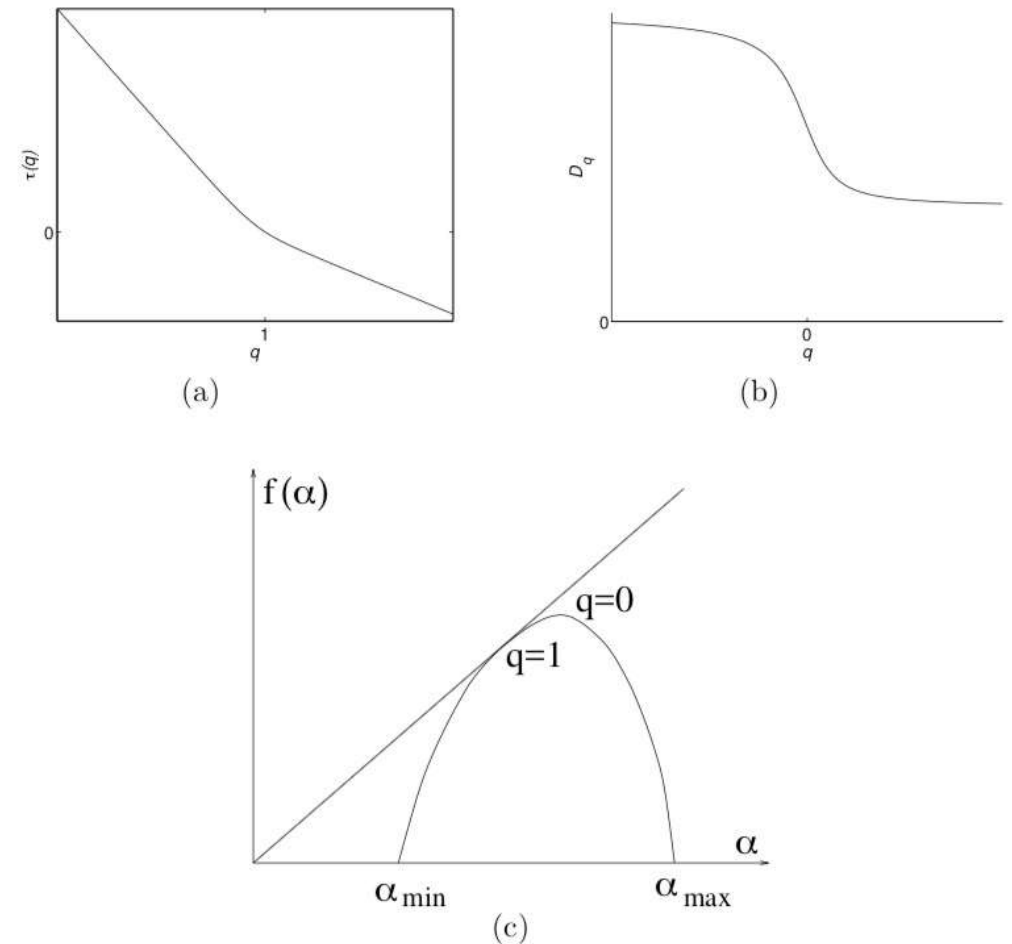
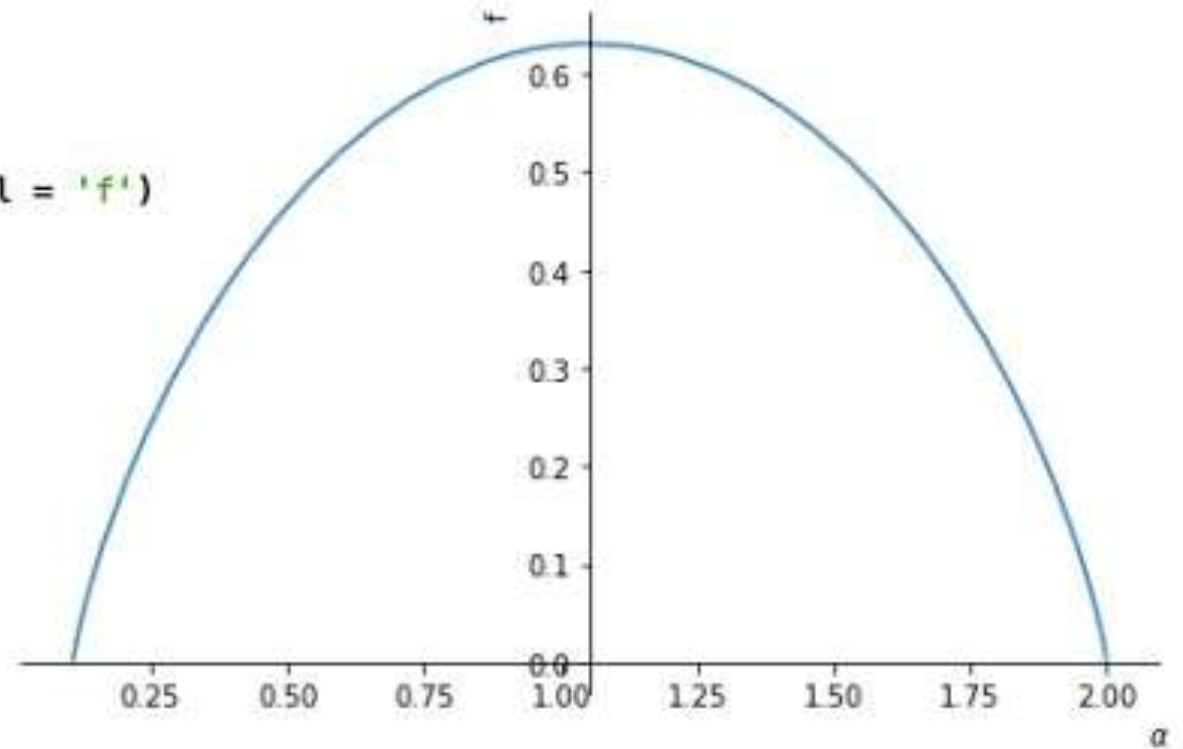


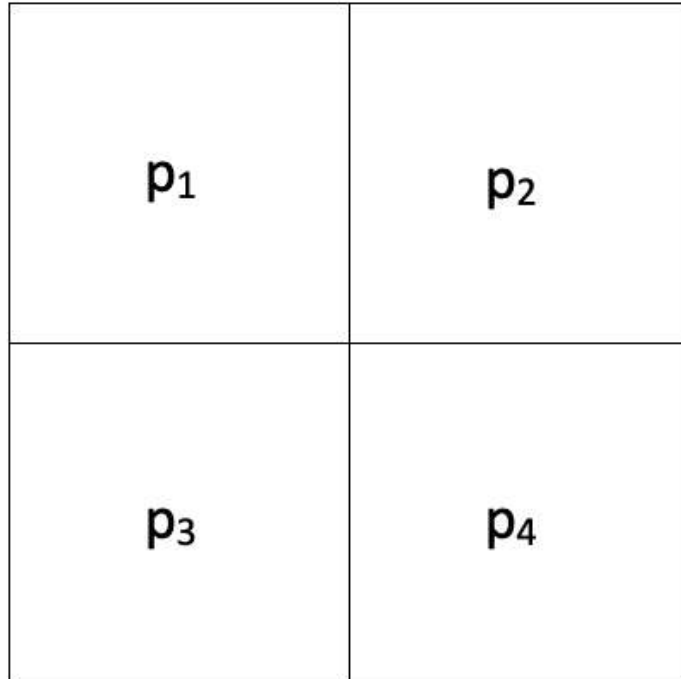
Figure 17.13: Typical curves of (a) the $\tau(q)$ function, (b) the D_q spectrum, and (c) the $f(\alpha)$ spectrum. In case (c), points on the curve near α_{\min} correspond to values of $q \rightarrow \infty$, and points on the curve near α_{\max} correspond to values of $q \rightarrow -\infty$.

Multifractals: $f(\alpha)$ Curve

```
1  # Multifractal Cantor Set
2  from sympy import log, symbols, diff
3  from sympy.plotting import plot_parametric
4  p1, p2 = 1/9, 8/9
5  q = symbols('q')
6  alpha = symbols('alpha')
7  tau = log(p1**q + p2**q) / log(3)
8  alpha = -diff(tau, q)
9  f = alpha * q + tau
10 plot_parametric(alpha, f, xlabel = r'$\alpha$', ylabel = 'f')
```



Multifractal Grid

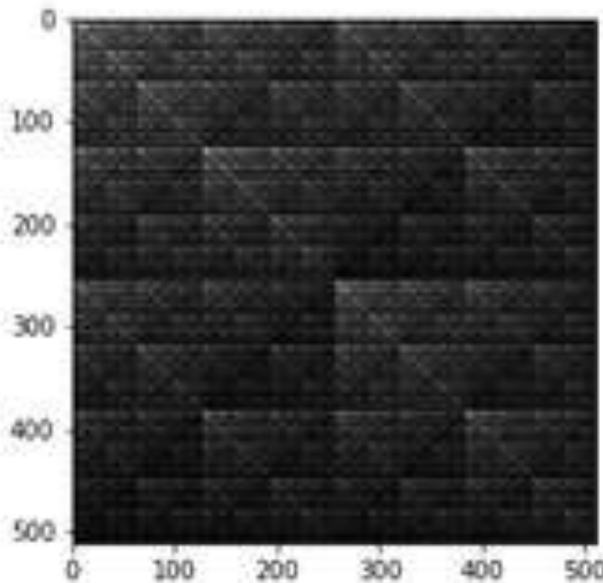
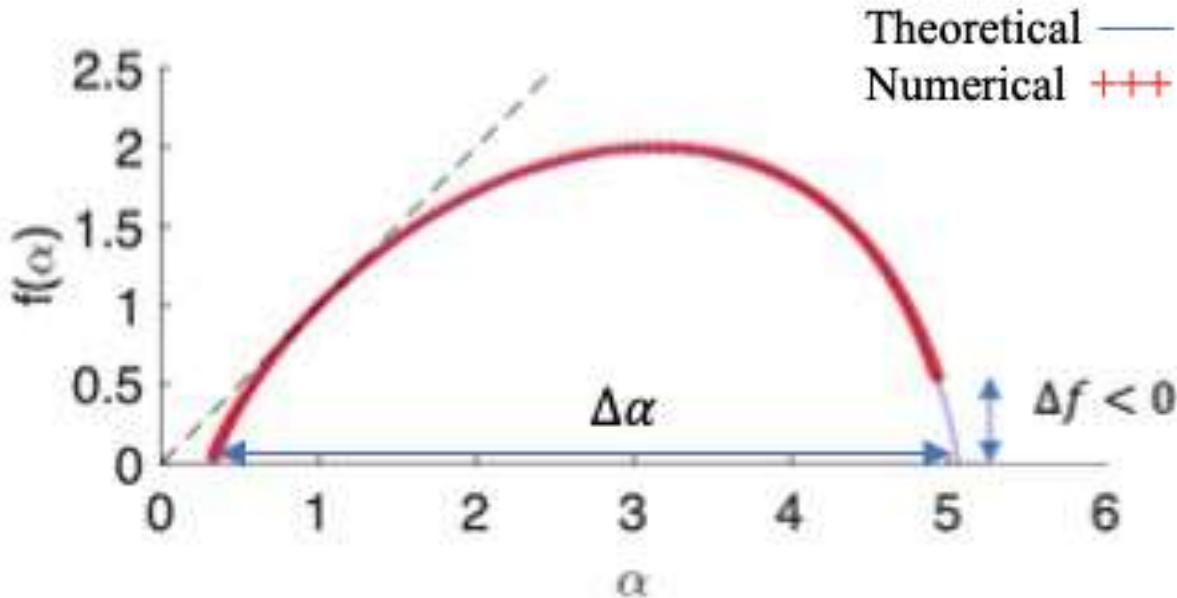


The grid multifractal: $N(\varepsilon) = 4, \varepsilon = \frac{1}{2}$.

```
1 # Multifractal of a 2x2 grid.
2 from sympy import log, symbols, diff
3 from sympy.plotting import plot_parametric
4 p1, p2, p3, p4 = 0.8, 0.1, 0.03, 0.07
5 q = symbols('q')
6 alpha = symbols('alpha')
7 tau = log(p1**q + p2**q + p3**q + p4**q) / log(2)
8 alpha = -diff(tau, q)
9 f = alpha * q + tau
10 plot_parametric(alpha, f, xlabel = r'$\alpha$', ylabel = 'f')
```

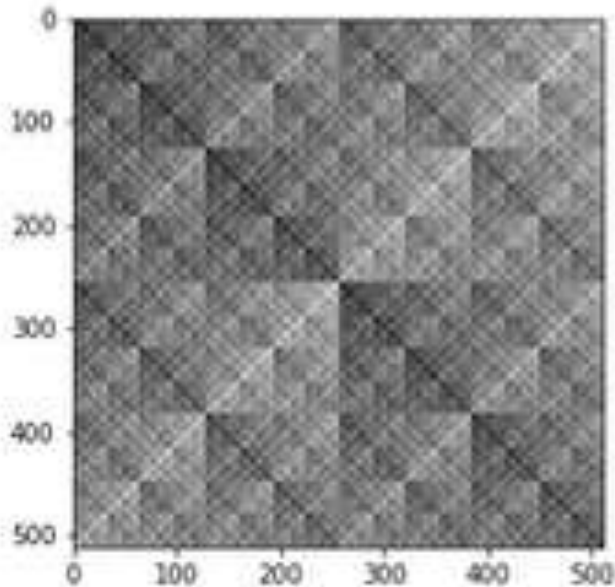
$$\tau = \frac{\ln(p_1^q + p_2^q + p_3^q + p_4^q)}{\ln(2)}, \quad \alpha = -\frac{d\tau}{dq}, \quad f(\alpha) = \alpha q + \tau.$$

Multifractals to Measure Dispersion and Clustering

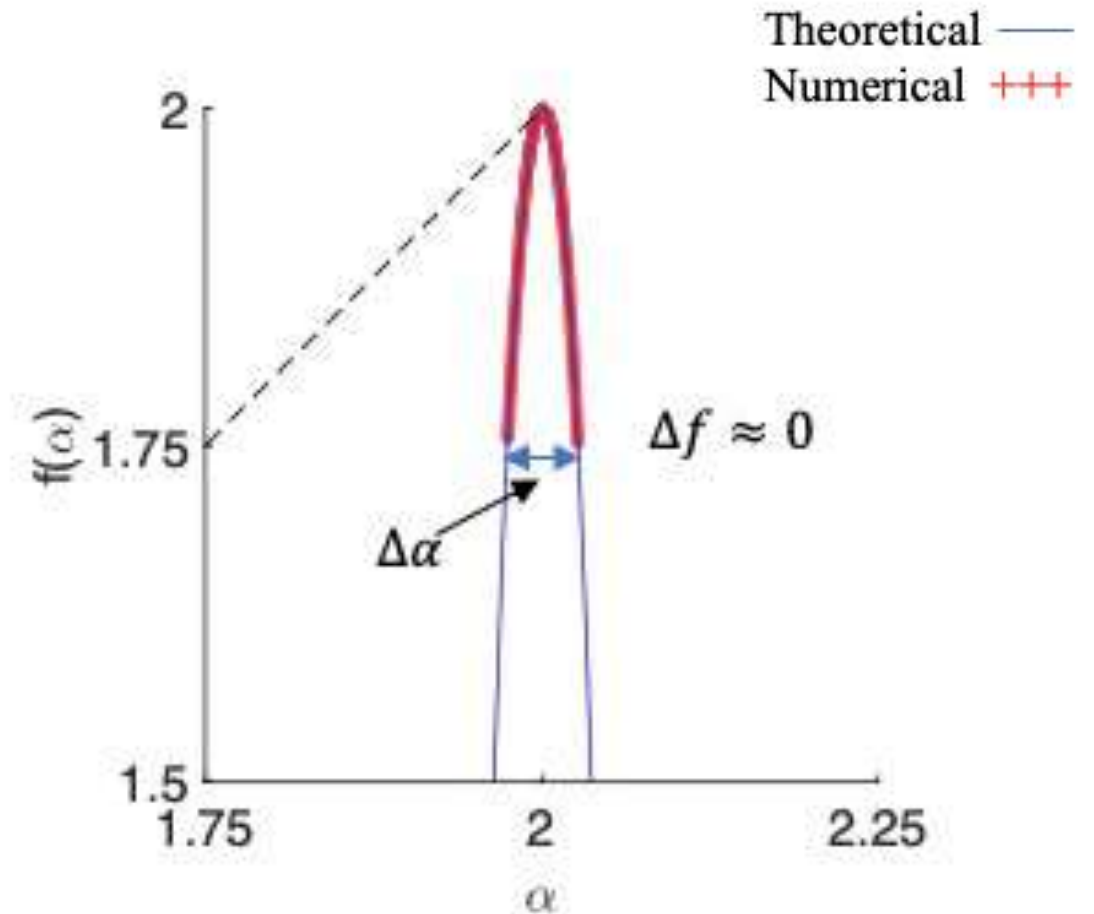
Multifractal motif	Representative figure	Multifractal f - α curve				
<table><tr><td>0.8</td><td>0.1</td></tr><tr><td>0.03</td><td>0.07</td></tr></table>	0.8	0.1	0.03	0.07	 <p>Clusters of dark pixels.</p>	 <p>The f-α curve is skewed left and $\Delta f < 0$.</p>
0.8	0.1					
0.03	0.07					

Multifractals to Measure Dispersion and Clustering

0.24	0.26
0.255	0.245



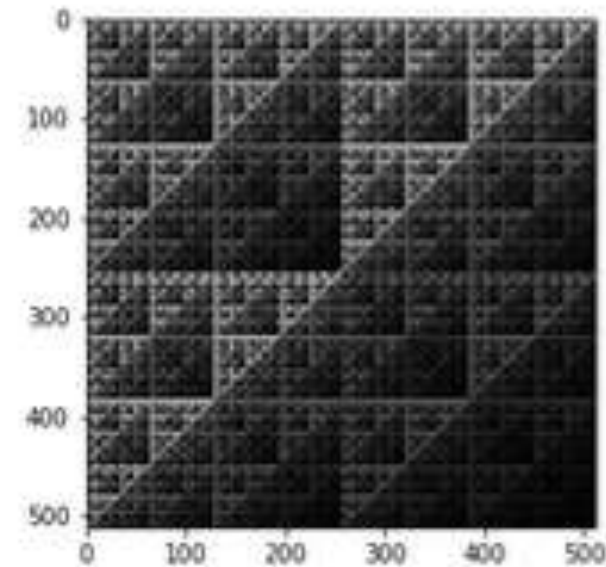
Homogeneous – no clusters.



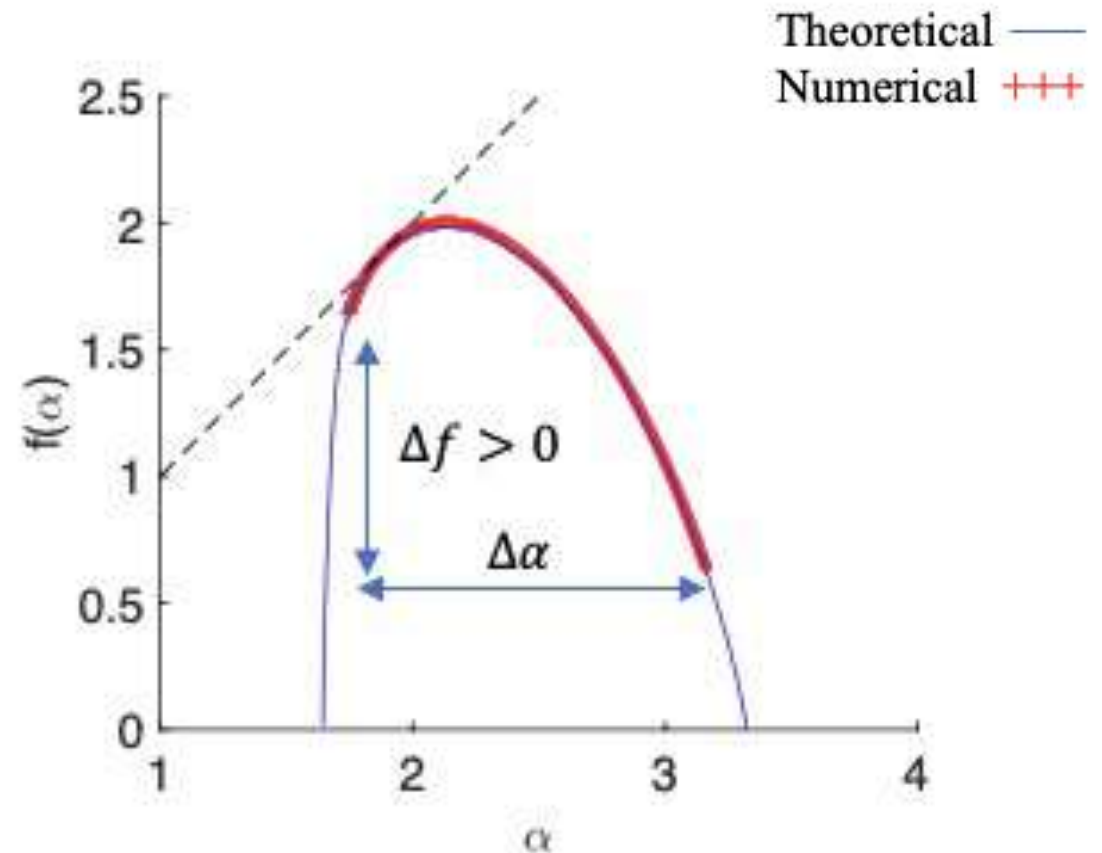
The f - α curve is not skewed and $\Delta f \approx 0$.

Multifractals to Measure Dispersion and Clustering

0.3	0.33
0.27	0.05



Clusters of bright pixels.

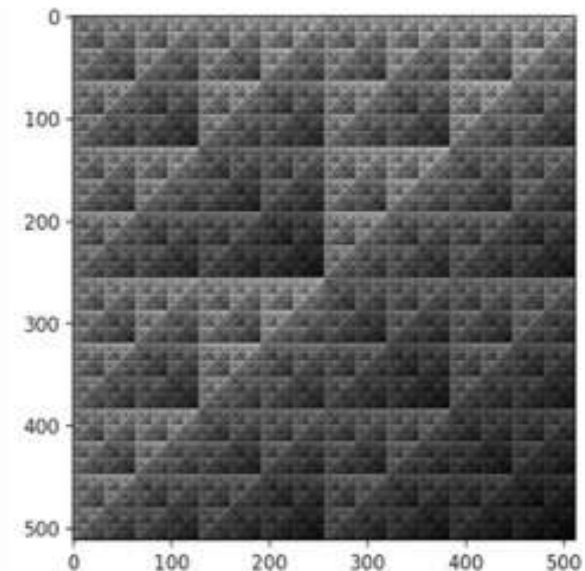


The f - α curve is skewed right and $\Delta f > 0$.

Chapter 18: Generating a Multifractal Image

```
1 # Program 18a: Generating a multifractal image.
2 # Save the image.
3 # See Figure 18.1(b).
4
5 import numpy as np
6 import matplotlib.pyplot as plt
7 from skimage import exposure, io, img_as_uint
8
9 p1, p2, p3, p4 = 0.3, 0.4, 0.25, 0.05
10 p = [[p1, p2], [p3, p4]]
11 for k in range(1, 9, 1):
12     M = np.zeros([2 ** (k + 1), 2 ** (k + 1)])
13     M.tolist()
14     for i in range(2**k):
15         for j in range(2**k):
16             M[i][j] = p1 * p[i][j]
17             M[i][j + 2**k] = p2 * p[i][j]
18             M[i + 2**k][j] = p3 * p[i][j]
19             M[i + 2**k][j + 2**k] = p4 * p[i][j]
20     p = M
21
22 # Plot the multifractal image.
23 M = exposure.adjust_gamma(M, 0.2)
24 plt.imshow(M, cmap='gray', interpolation='nearest')
25
26 # Save the image as a portable network graphics (png) image.
27 im = np.array(M, dtype='float64')
28 im = exposure.rescale_intensity(im, out_range='float')
29 im = img_as_uint(im)
30 io.imsave('Multifractal.png', im)
31 io.show()
```

0.3	0.4
0.25	0.05



Multifractals in the Real World: End Session 2

Whitehead KA, El Mohtadi M, **Lynch S**, Liauw CM, Amin M, Deisenroth T, Preuss A and Verran J (2021) Diverse surface properties reveal that substratum roughness affects fungal spore binding. *iScience* 24(4), 102333.

Slate AJ, Whitehead KA, **Lynch S**, Foster CW and Banks CE (2020) Electrochemical decoration of additively manufactured graphene macro electrodes with MoO₂ nanowire: An approach to demonstrate the surface morphology, *J. of Physical Chemistry C*, 124(28) 15377-15385.

Wickens D, **Lynch S**, Kelly P, West G, Whitehead K, and Verran J, (2014) Quantifying the pattern of microbial cell dispersion, density and clustering on surfaces of differing chemistries and topographies using multifractal analysis, *Journal of Microbiological Methods*, 104, 101-108.

Mills SL, Lees G, Liauw C and **Lynch S** (2004) An improved method for the dispersion assessment of flame retardent filler/polymer systems based on the multifractal analysis of SEM images, *Macromolecular Materials and Engineering*, **289**(10), 864-871.

Drozd S, Kowalski R, Oswiecimka P, Rak R and Gebarowski R (2018) Dynamical variety of shapes in financial multifractality, *Complexity* 7015721, 1-13.

Models in Engineering: Start Session 3

Periodically forced pendulum $\ddot{x} + k\dot{x} + (x^3 - x) = \Gamma \cos(\omega t),$

$$\dot{x} = y \quad \dot{y} = x - ky - x^3 + \Gamma \cos(\omega t)$$

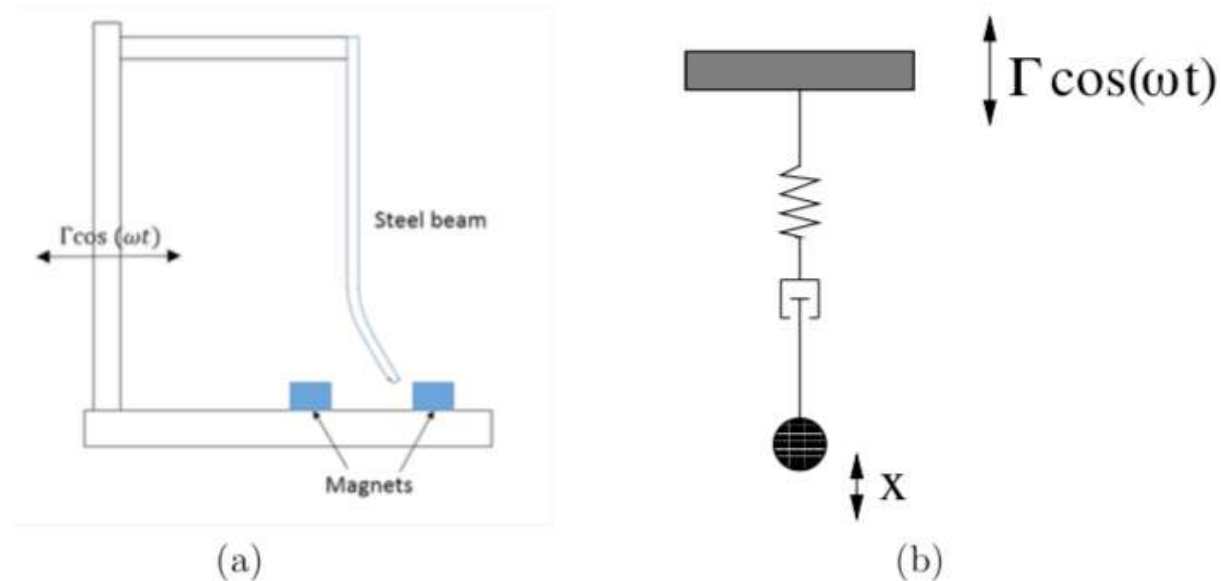


Figure 9.8: (a) A steel beam between two magnets. (b) A periodically driven pendulum.

Hysteresis in the Periodically Forced Pendulum

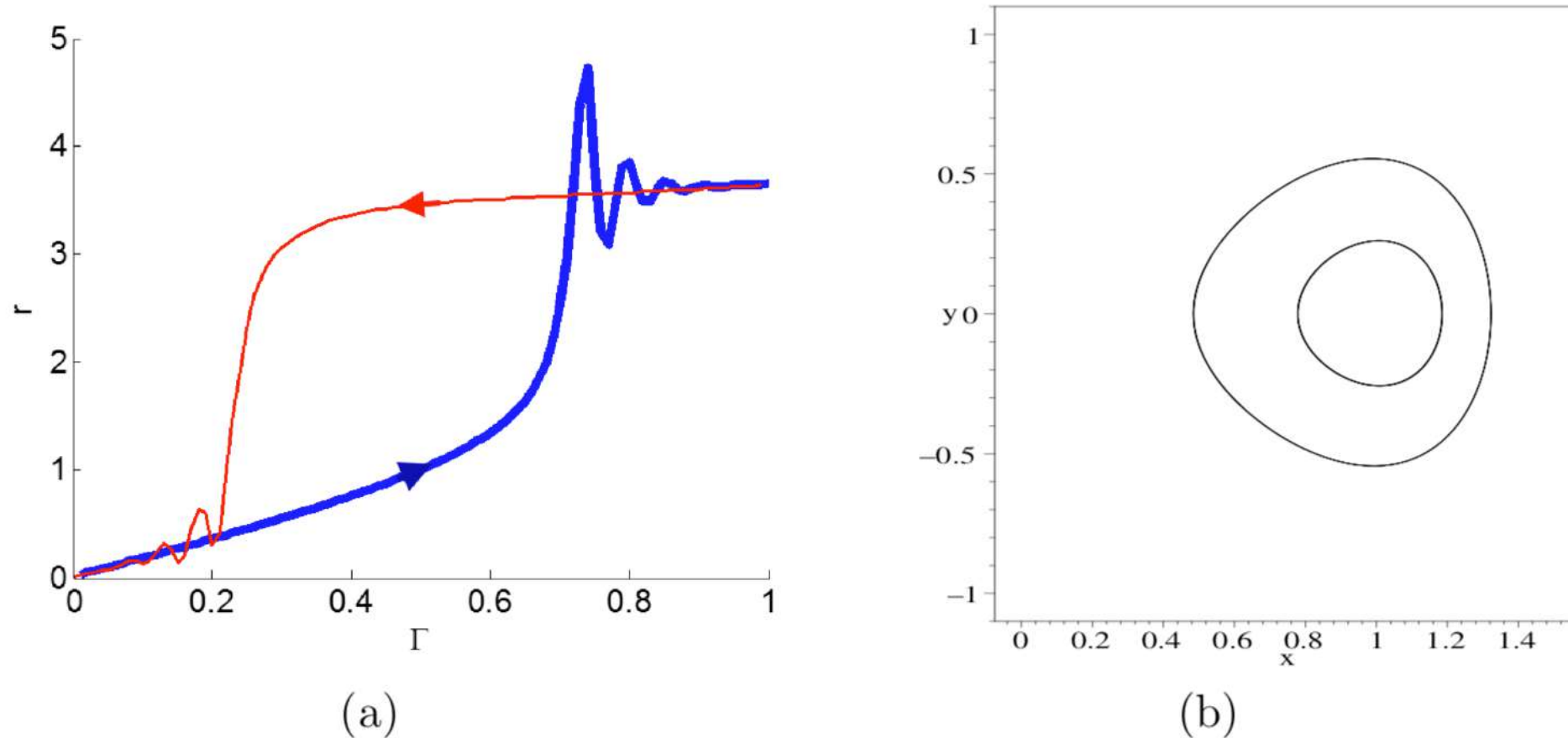


Figure 23.2: (a) Bifurcation diagram. (b) Multistable behavior.

Periodically Driven Pendulum: Program 9c.py

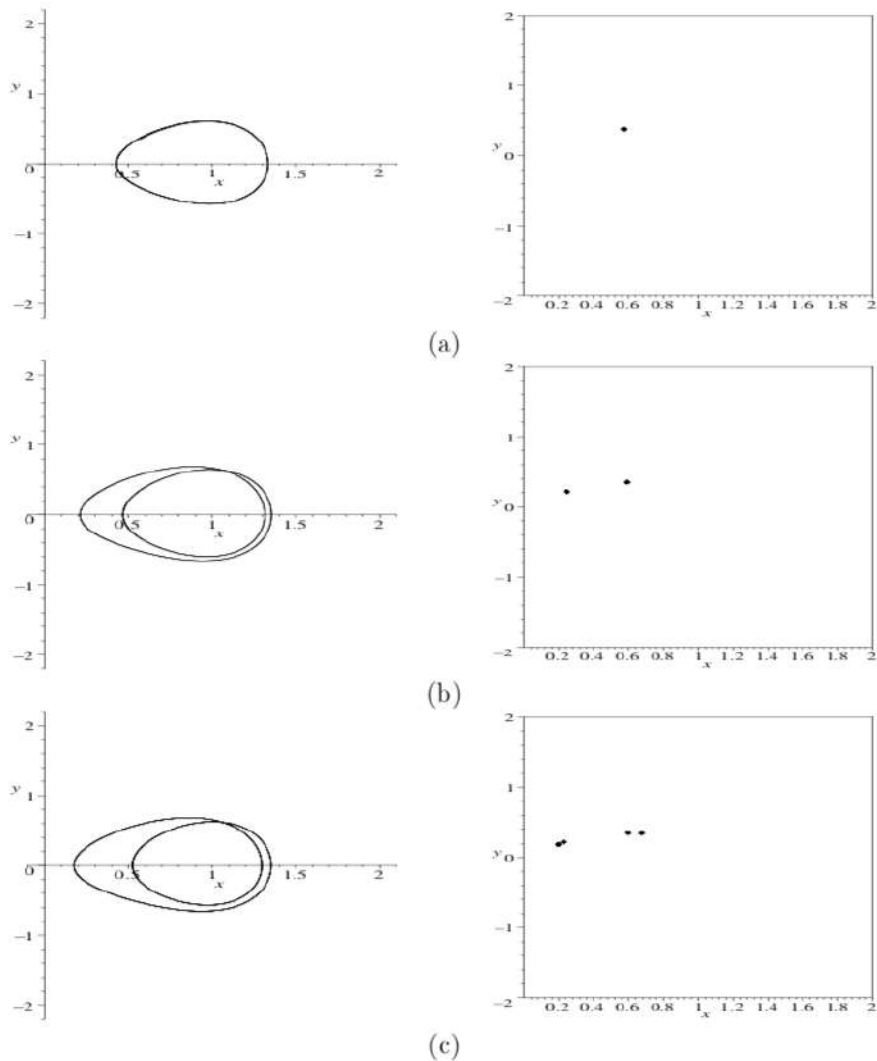


Figure 9.10: A gallery of phase portraits and Poincaré maps for system (9.10) when $\alpha = 1, \beta = -1, k = 0.3$ and $\omega = 1.25$: (a) $\Gamma = 0.2$ (forced period one), (b) $\Gamma = 0.3$ (a period-two subharmonic), and (c) $\Gamma = 0.31$ (a period-four subharmonic).

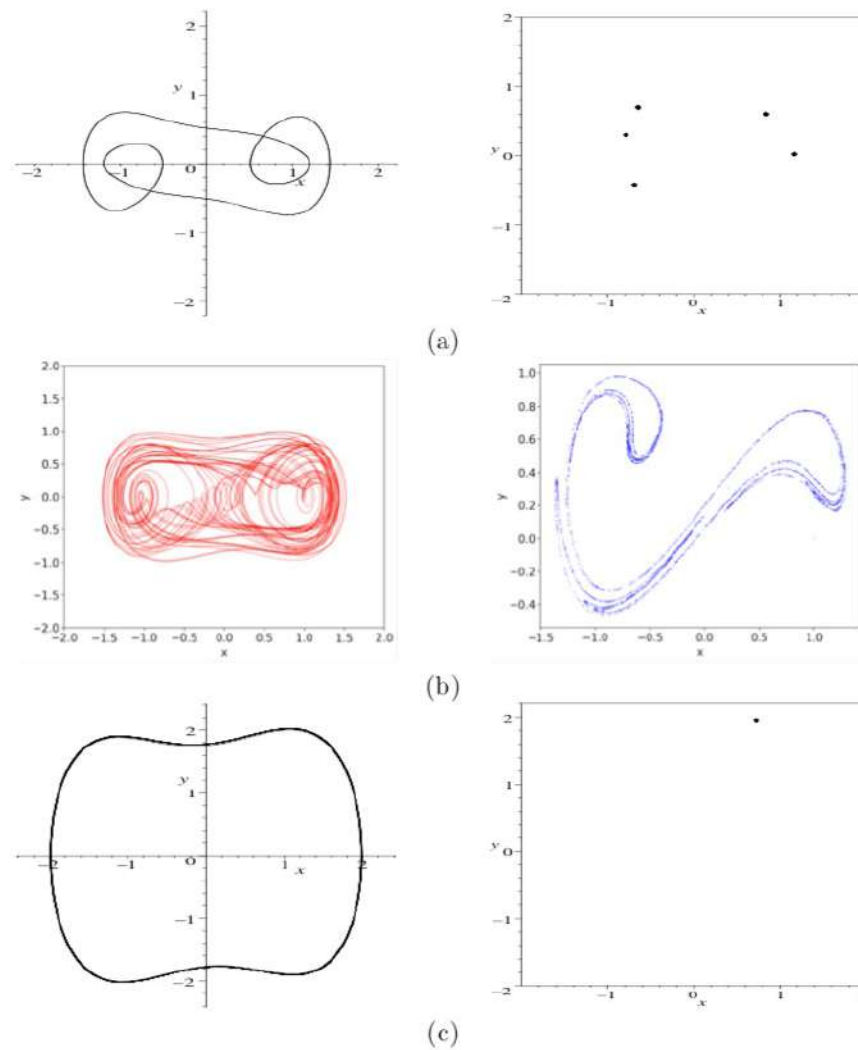


Figure 9.11: [Python] A gallery of phase portraits and Poincaré maps for system (9.10) when $\alpha = 1, \beta = -1, k = 0.3$ and $\omega = 1.25$: (a) $\Gamma = 0.37$ (a period-five subharmonic); (b) $\Gamma = 0.5$ (chaos), 4000 points are plotted; (c) $\Gamma = 0.8$ (forced period one).

Hysteresis in the Periodically Driven Two-Bar Linkage

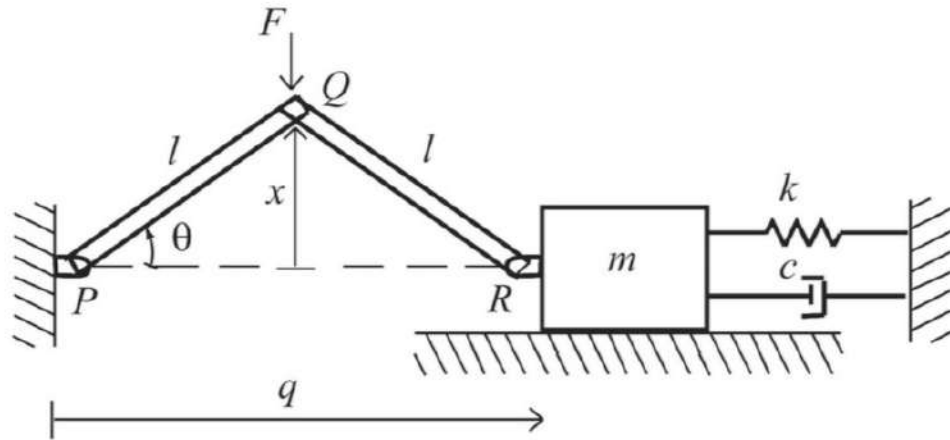
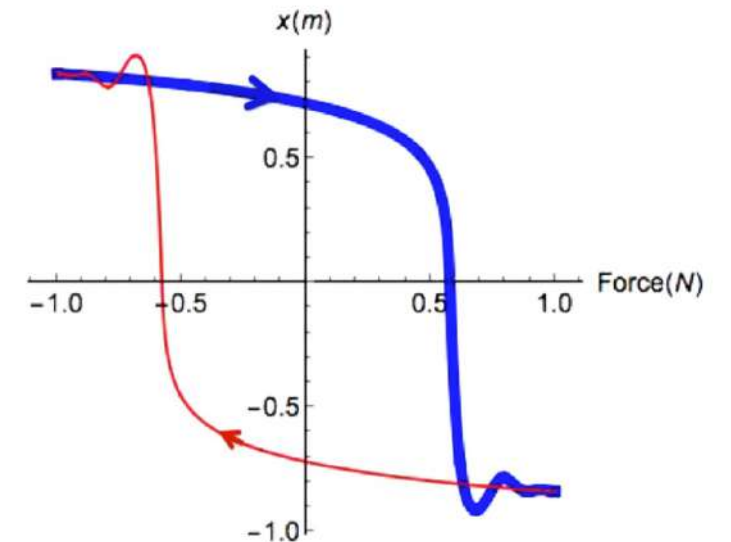
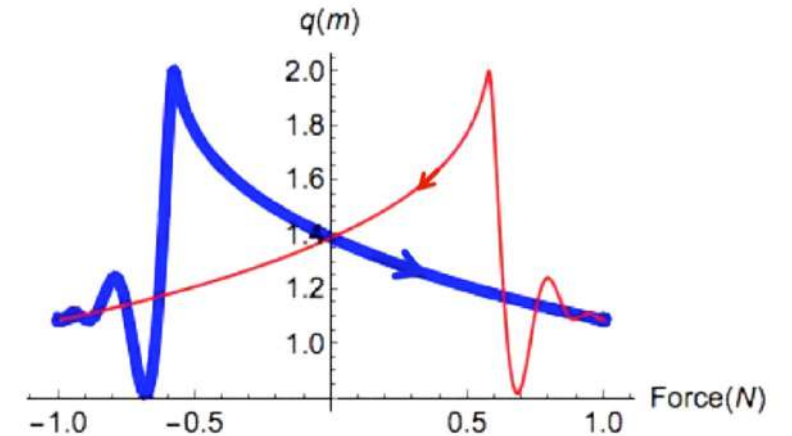


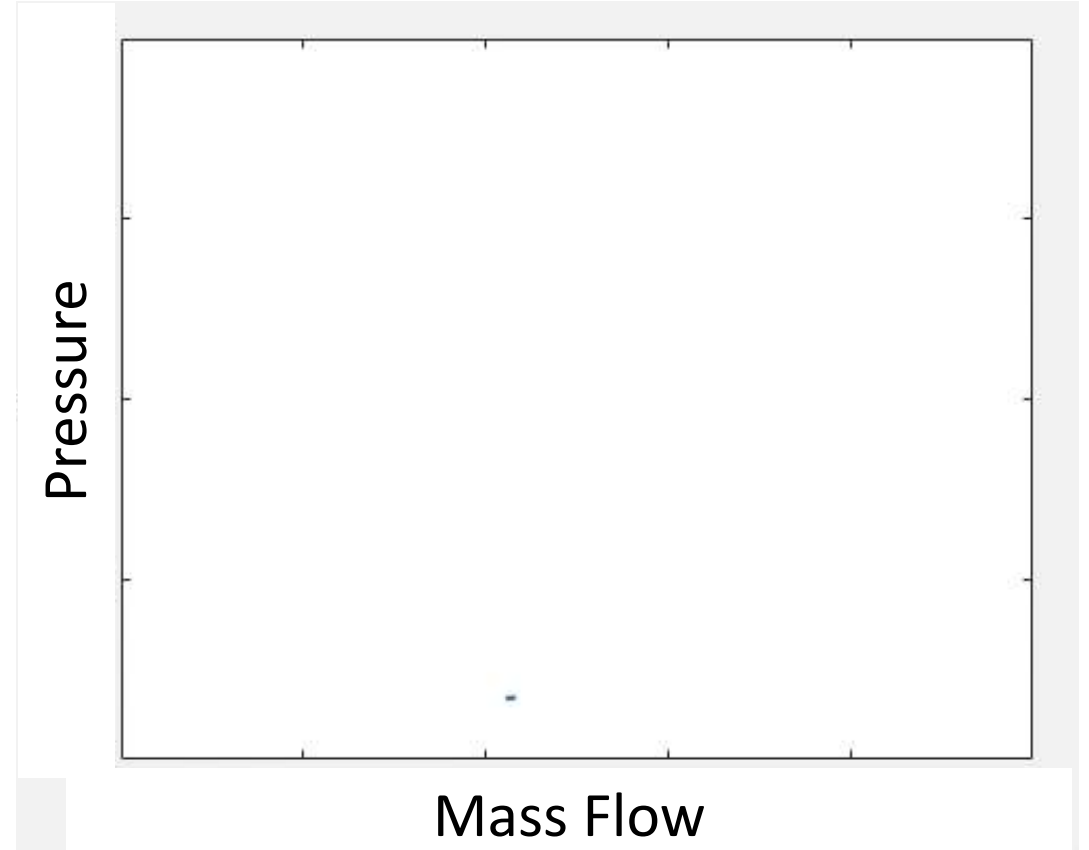
Figure 22.2: The preloaded two-bar linkage with a periodic force F acting at the joint Q . As the point Q moves vertically up and down, the mass m moves horizontally left and right.



Limit Cycles: Surge in Jet Engines

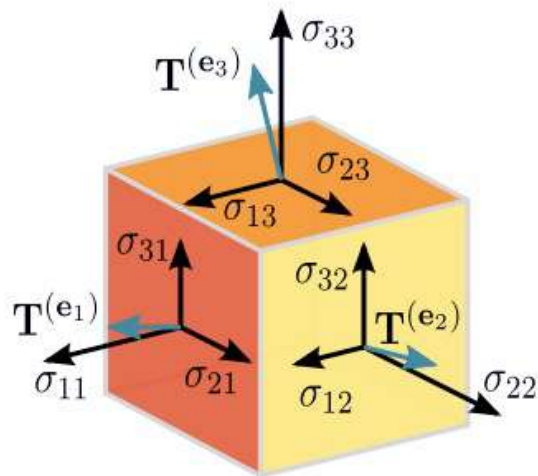


Animation: The nonlinear characteristics of a jet engine. This is a hard or dangerous bifurcation of one limit cycle of small amplitude to a large amplitude limit cycle.



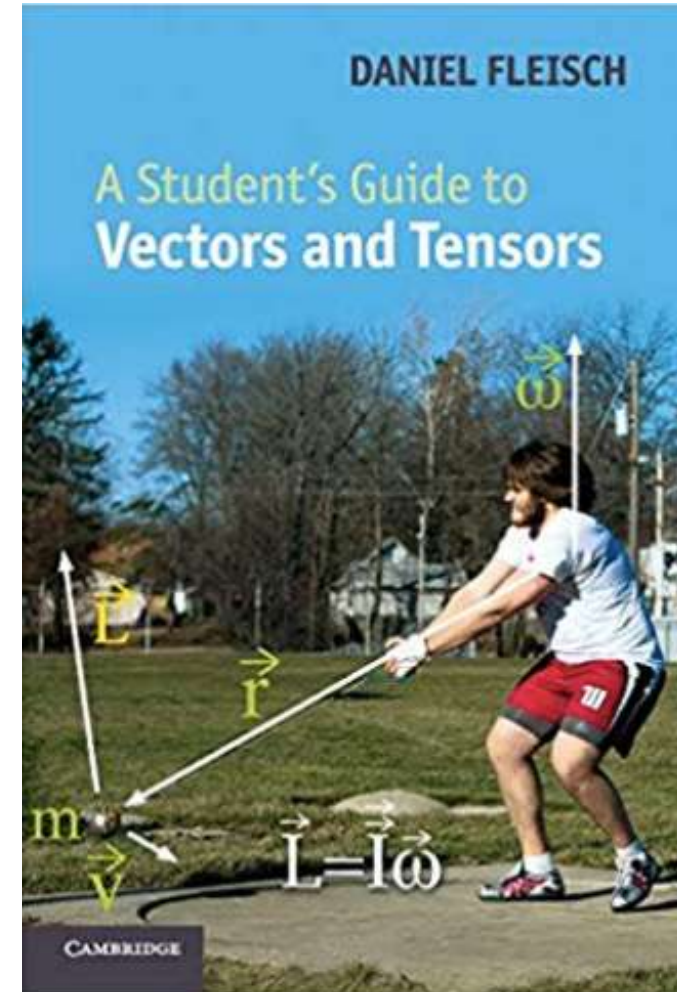
Tensors

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

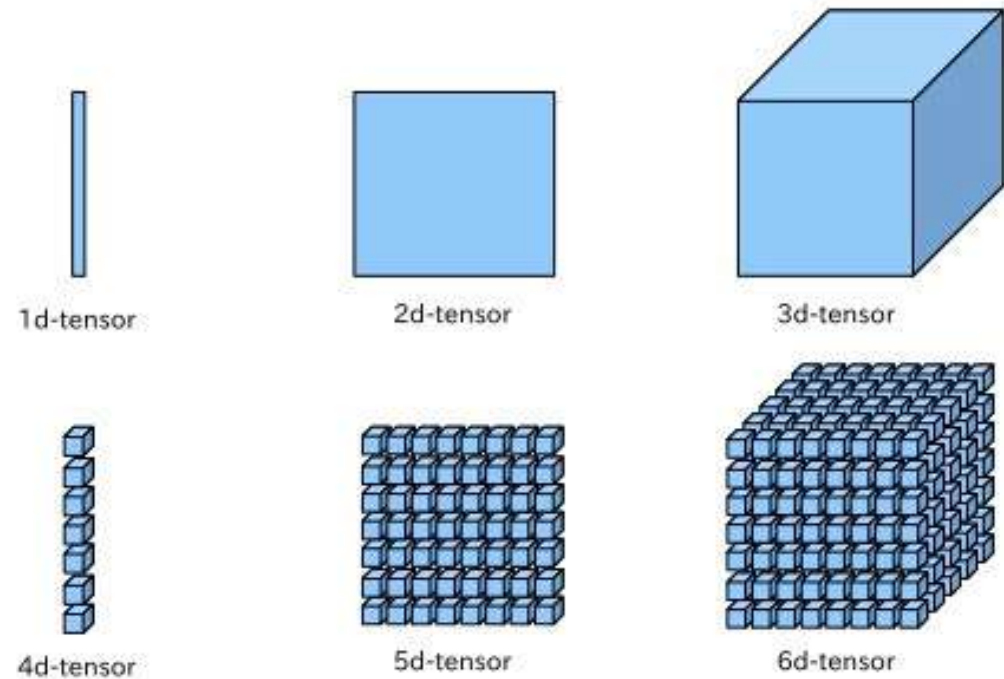
[[1,2,3],[4,5,6],[7,8,9]] is an array and rank 2 tensor

[[[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

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
```

Linear Electric Circuits: Resistor-Inductor-Capacitor

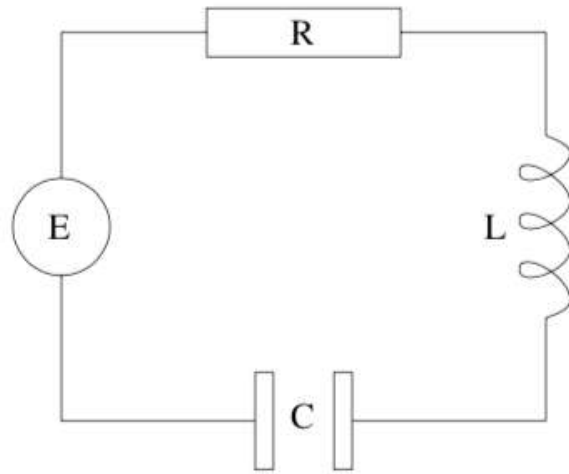
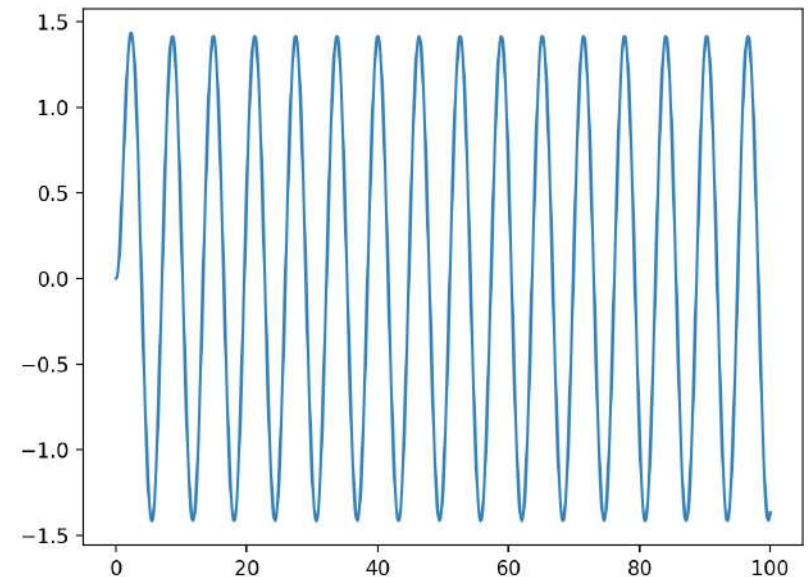


Figure 2.8: Schematic of a simple RLC series circuit.

$$\frac{d^2 I}{dt^2} + 5 \frac{dI}{dt} + 6I = 10 \sin(t).$$

$$I(t) = 2e^{-2t} - e^{-3t} + \sin(t) - \cos(t).$$



Nonlinear Electric Circuits

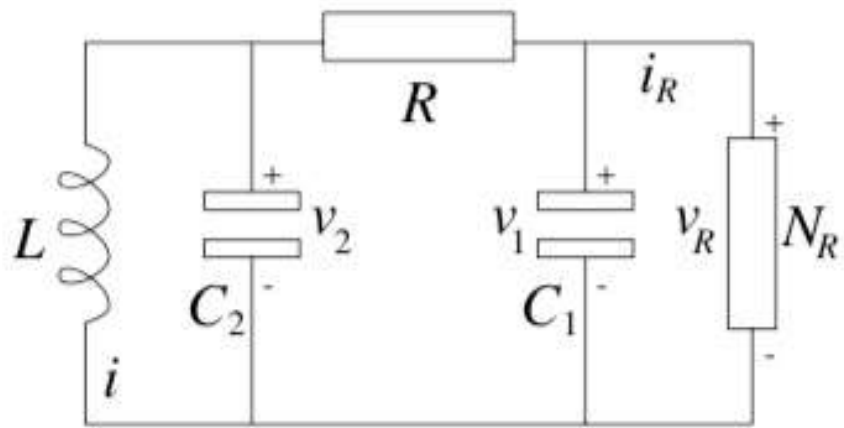


Figure 8.13: Chua's electric circuit.

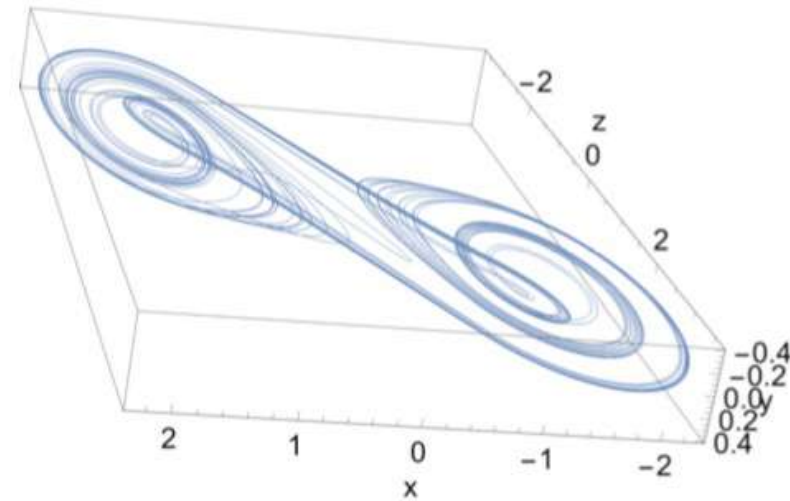
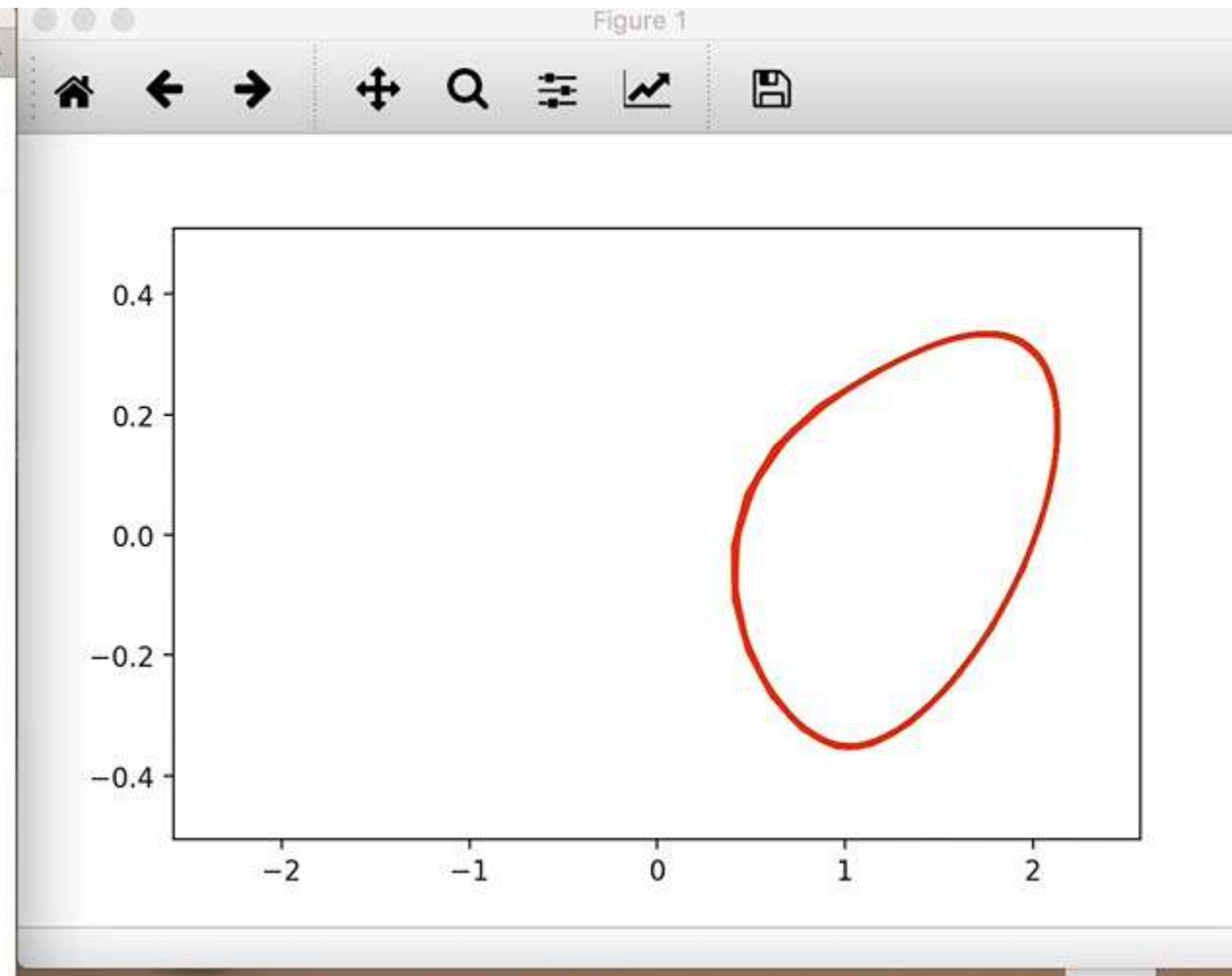
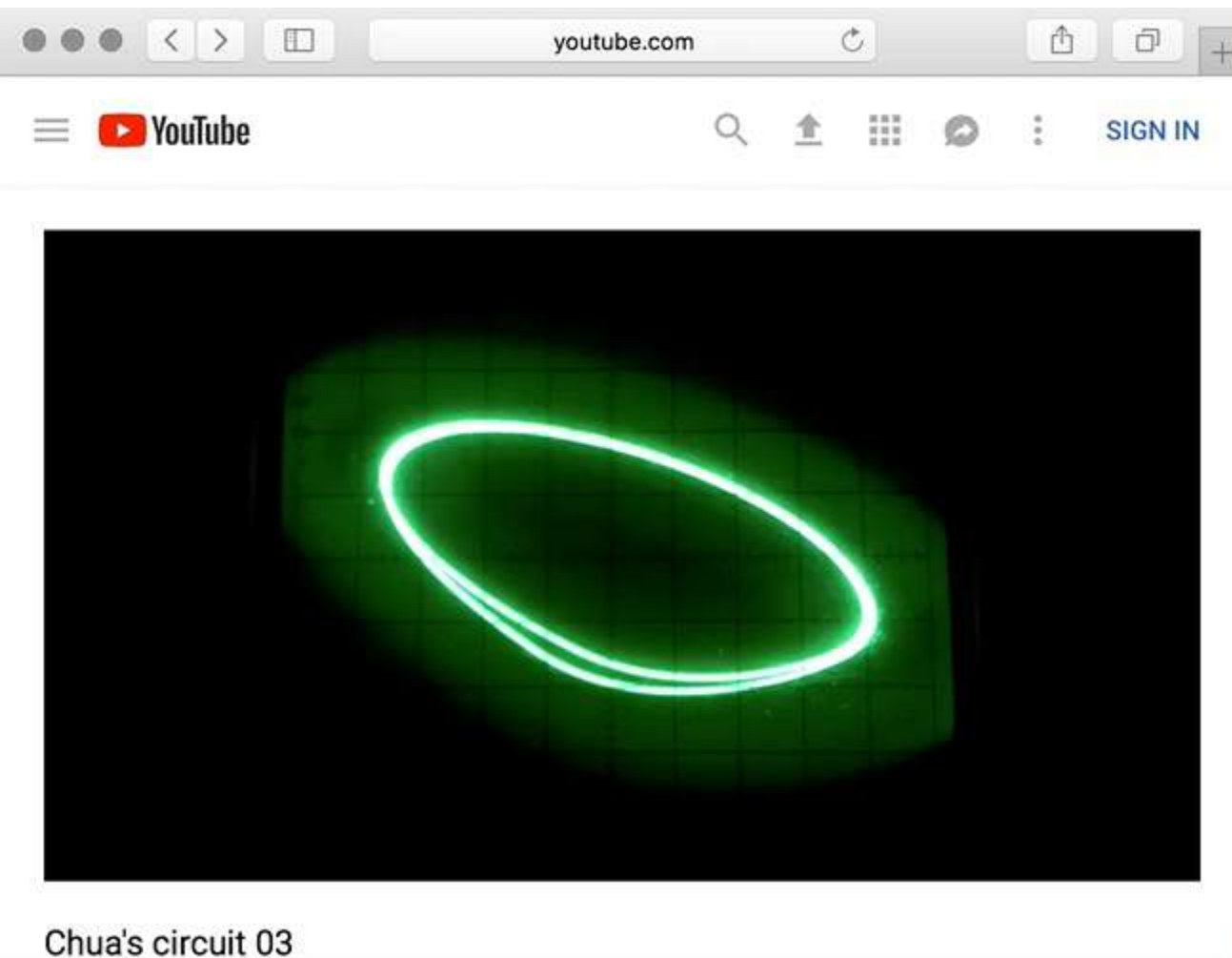


Figure 8.14: [Python animation] Chua's double-scroll attractor: Phase portrait for system (8.9) when $a = 15$, $b = 25.58$, $c = -5/7$, and $d = -8/7$. The initial conditions are $x(0) = -1.6$, $y(0) = 0$, and $z(0) = 1.6$.

Chua Circuit Animation

```
Program_08d.py*
1 # Program 08d: Animation of a Chua circuit bifurcation.
2 # Type %matplotlib qt5 in the Console window to run animation.
3 # You can watch a YouTube animation on the web.
4 # Search for Chua circuit animated bifurcation.
5
6 from matplotlib import pyplot as plt
7 from matplotlib.animation import ArtistAnimation
8 import numpy as np
9 from scipy.integrate import odeint
10 fig = plt.figure()
11 m0, m1, tmax = -1/7, 2/7, 100
12 def chua(x, t):
13     return [a * (x[1] - (m1*x[0] + (m0-m1) / 2 * (np.abs(x[0] + 1) - np.abs(x[0] - 1)))),
14            x[0] - x[1] + x[2],
15            -15*x[1]]
16 time = np.arange(0, tmax, 0.1)
17 x0 = [1.96, -0.0519, -3.077]
18 myimages = []
19 for a in np.arange(8, 11, 0.05):
20     xs = odeint(chua, x0, time)
21     imgplot = plt.plot(xs[:, 0], xs[:, 1], 'r-')
22     myimages.append(imgplot)
23 my_anim = ArtistAnimation(fig, myimages, interval=500, blit=False, repeat_delay=500)
24 plt.show()
```


Chua Circuit Animation: End Session 3



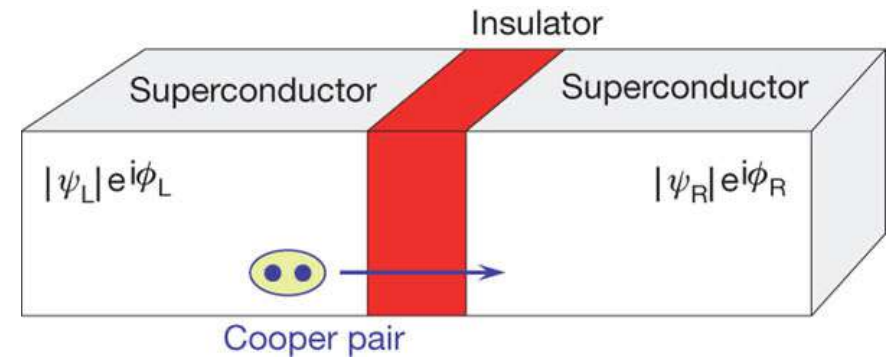
Models in Physics: Josephson Junctions (JJs): Start Session 4

Mathematical Model of a JJ. A JJ with two superconducting layers sandwiching an insulating layer will be investigated. The differential equation used to model the resistively shunted JJ is written as

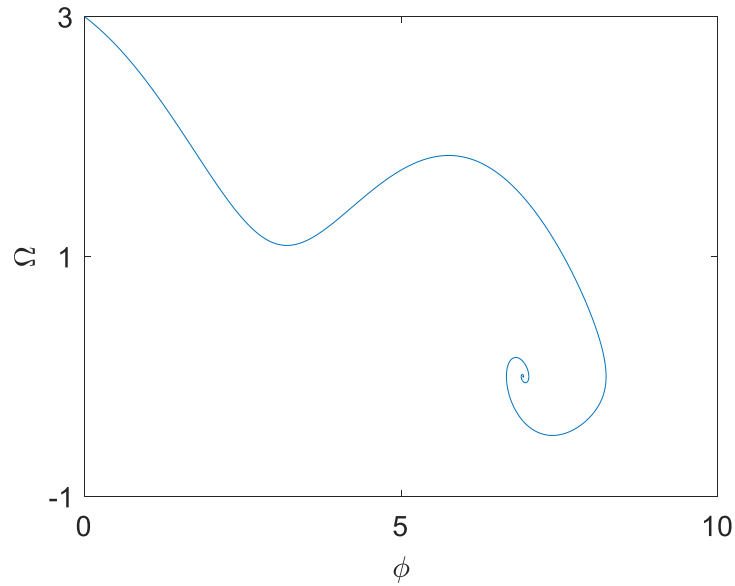
$$\frac{d^2\phi}{d\tau^2} + \beta_J \frac{d\phi}{d\tau} + \sin \phi = \kappa, \quad (21.12)$$

where ϕ is a phase difference, β_J is a parameter inversely related to the Josephson plasma frequency ω_J , κ is related to the total current across the junction and

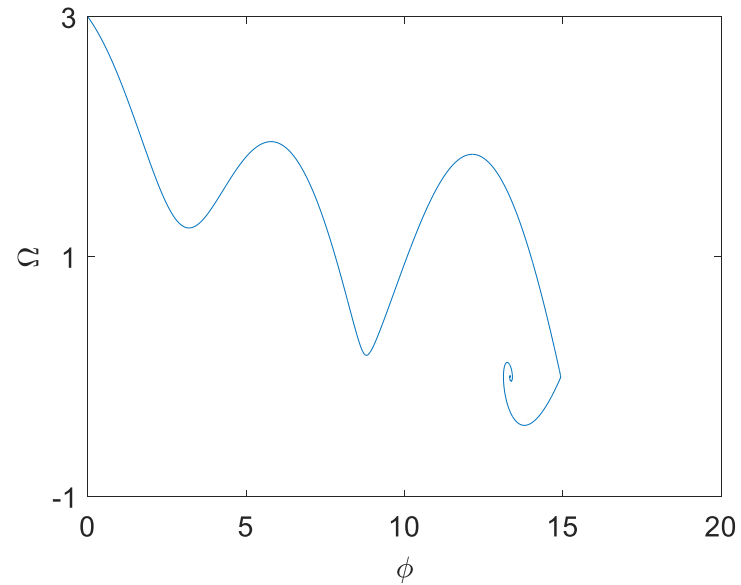
$$\frac{d\phi}{dt} = \omega_J \frac{d\phi}{d\tau}.$$



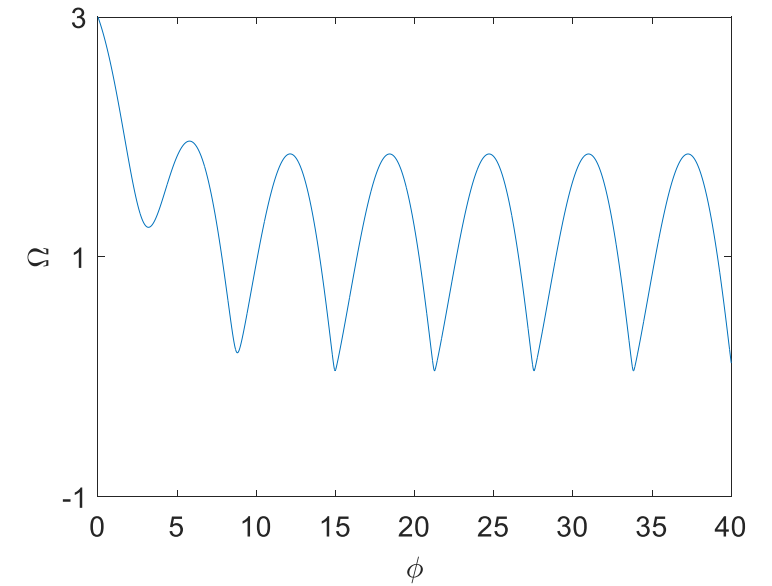
Josephson Junctions: Threshold Oscillators



Below threshold, no oscillation.



Below threshold, no oscillation.



Above threshold, oscillation.

Just like starting a petrol lawn mower!



Josephson Junctions

```
1 # Program 21c: Josephson junction limit cycle.
2 # See Figure 21.9.
3
4 from matplotlib import pyplot as plt
5 import numpy as np
6 from scipy.integrate import odeint
7
8 fig = plt.figure()
9
10 bj = 1.2
11 tmax = 100
12 kappa = 1.4
13
14 def jj_ode(x, t):
15     return [x[1], kappa - bj*x[1] - np.sin(x[0])]
16
17 time = np.arange(0, tmax, 0.1)
18 x0=[0.1,0.1]
19 xs = odeint(jj_ode, x0, time)
20 imgplot = plt.plot(np.sin(xs[:, 0]), xs[:, 1], 'r-')
21
22 plt.xlabel(r'$\sin(\phi)$', fontsize=15)
23 plt.ylabel(r'$\Omega$', fontsize=15)
24 plt.tick_params(labelsize=15)
25 plt.show()
```

Josephson Junctions: Hysteresis

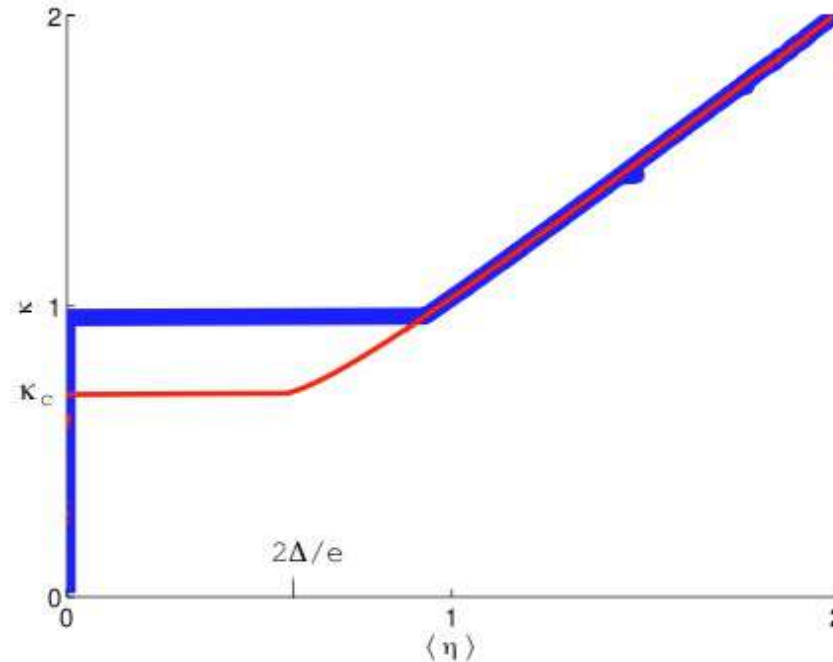


Figure 21.10: A typical I-V characteristic curve usually observed in a tunneling JJ. The blue curve shows the current for increasing voltage and the red curve depicts current for decreasing voltage. There is a clockwise hysteresis cycle. Note that $\kappa = \frac{I}{I_c}$ and the average voltage, $\langle \eta \rangle = \beta_J \langle \Omega \rangle$.

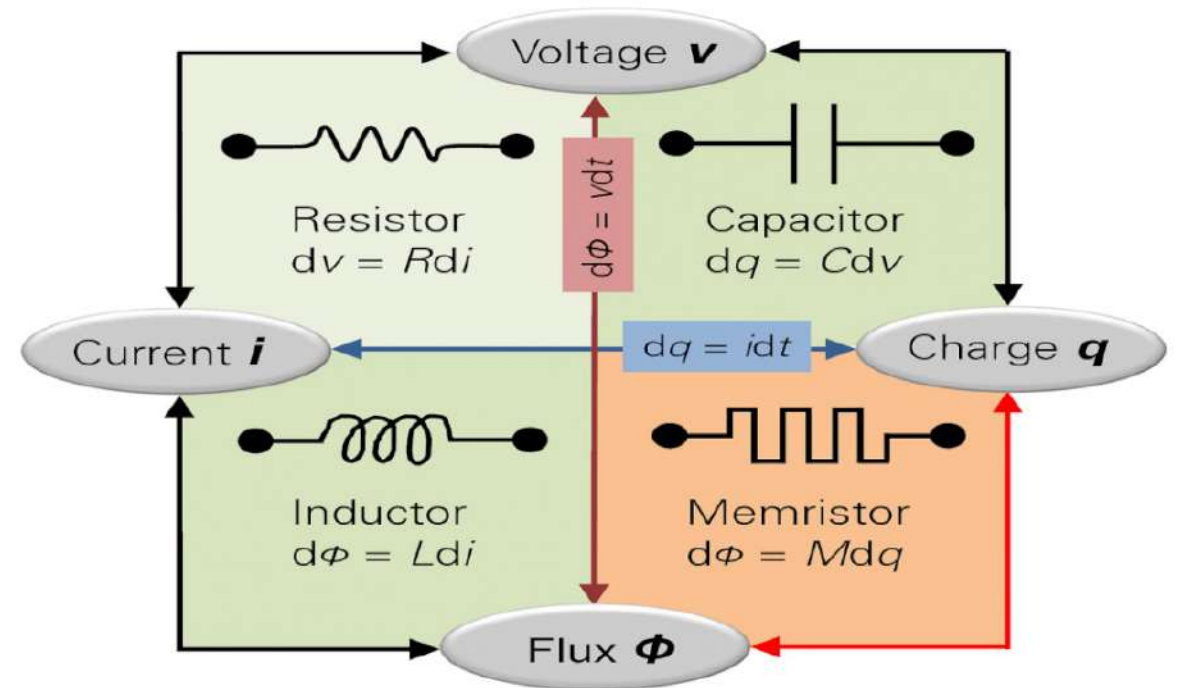
Memristors

Memristor – the Missing Circuit Element

In 1971, Leon Chua mathematically proved the existence of the memristor.



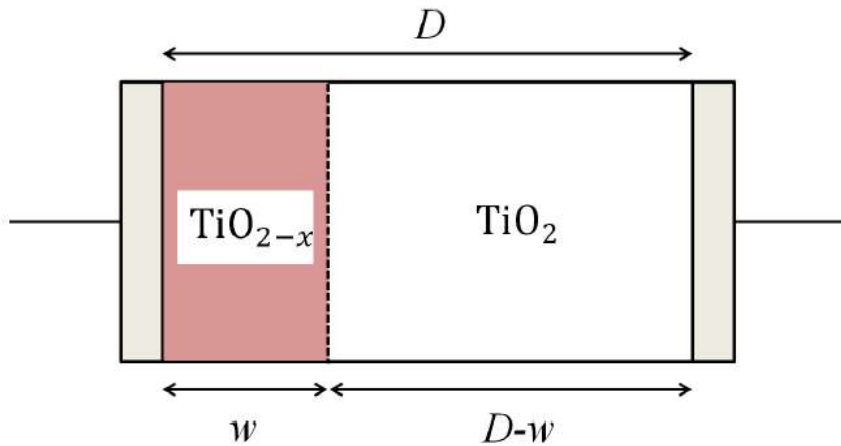
Steve Furber, Jon Borresen & Leon Chua.



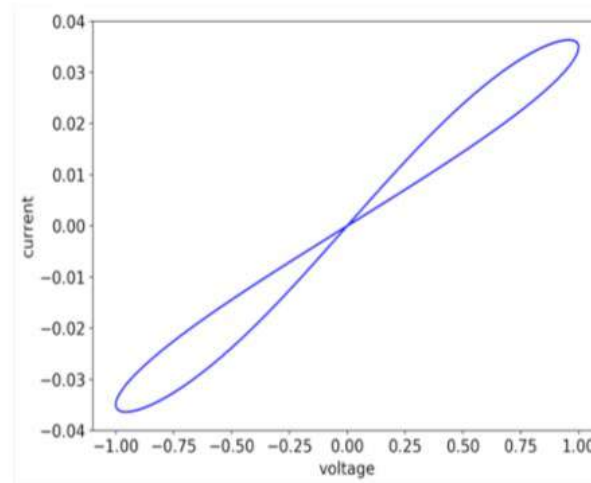
Fundamental circuit variable relationships.

Memristors

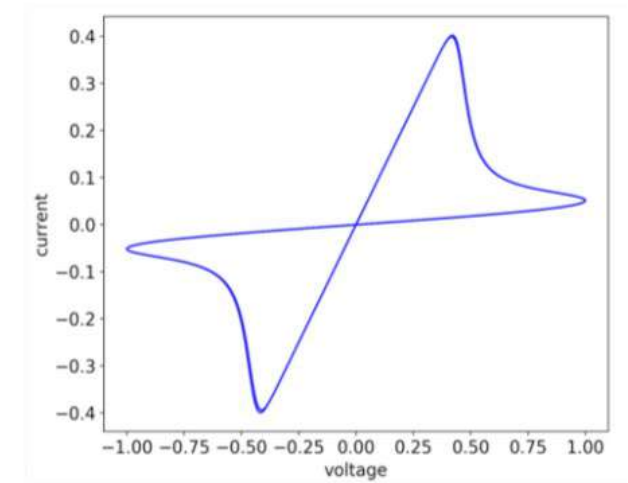
The Memristor and Pinched Hysteresis



HP Labs titanium dioxide memristor.



(a)



(b)

Pinched hysteresis loops. There remains some controversy about memristors.

Memristors: Pinched Hysteresis

```
Program_21e.py*
1 # Program 21e: Pinched hysteresis in a memristor.
2 # See Figure 21.12.
3
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from scipy.integrate import odeint
7 # Constants
8 eta, L, Roff, Ron, p, T, w0 = 1.0, 1.0, 70.0, 1.0, 10.0, 20.0, 0.5
9 t=np.arange(0.0, 40.0, 0.01)
10 # Set up the ODEs, see equations (21.3)
11 def memristor(X, t):
12     w = X
13     dwdt = ((eta * (1 - (2*w - 1) ** (2*p)) * np.sin(2*np.pi * t/T))
14             / (Roff - (Roff - Ron) * w))
15     return dwdt
16 X = odeint(memristor, [w0], t, rtol=1e-12)
17 w = X[:, 0]
18 plt.plot(np.sin(2*np.pi * t/T), np.sin(2*np.pi * t/T)
19          / (Roff - (Roff - Ron) * X[:, 0]), 'b')
20 plt.xlabel('voltage', fontsize=15)
21 plt.ylabel('current', fontsize=15)
22 plt.tick_params(labelsize=15)
23 plt.show()
```

Chapter 16: Electromagnetic (EM) Waves and Optical Resonators

Chapter 16

- 16.1 Maxwell's Equations and EM Waves
- 16.2 Historical Background
- 16.3 The Nonlinear SFR
- 16.4 Chaotic Attractors and Hysteresis
- 16.5 Linear Stability Analysis

$$E_{n+1} = A + BE_n \exp[i(|E_n|^2 + \phi_L)].$$

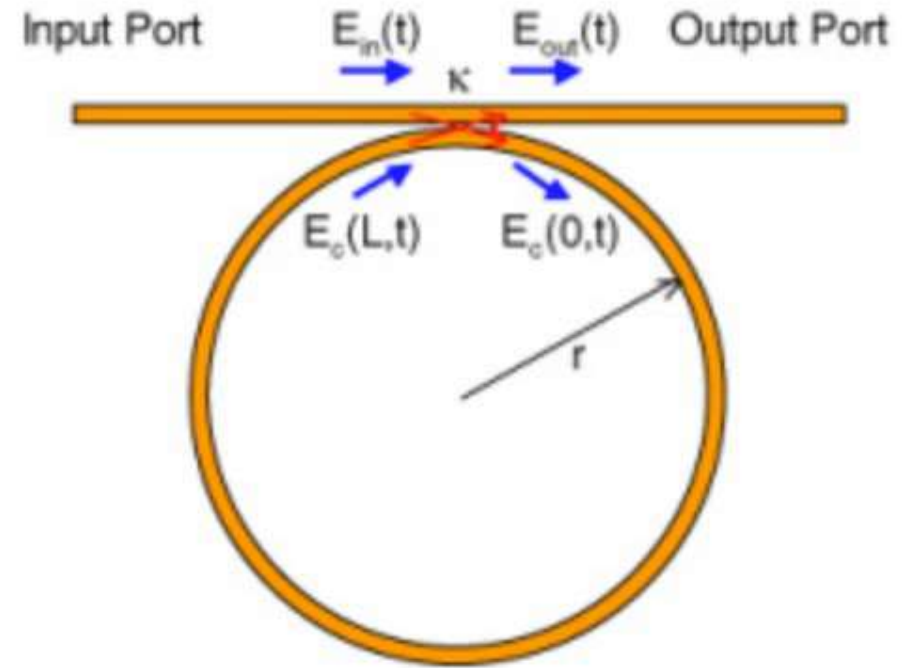


Fig. The Simple Fiber Ring (SFR) Resonator.

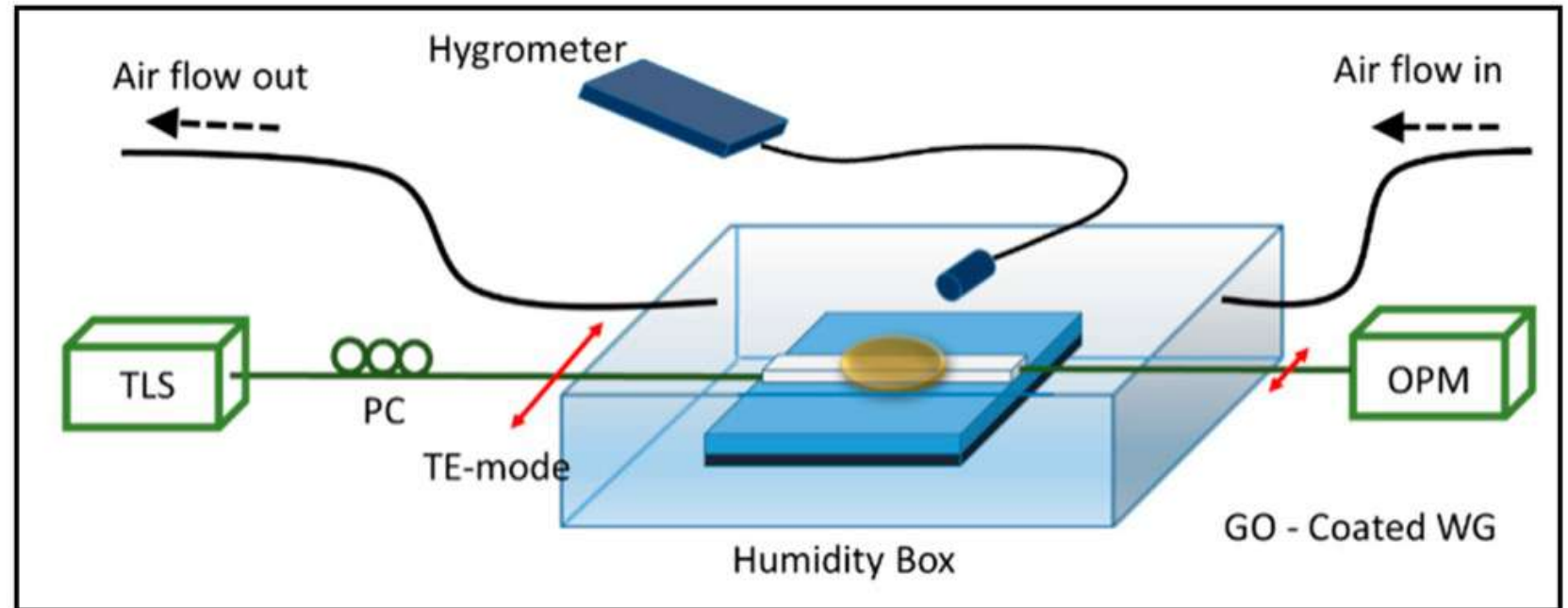
Applications: Sensors

Optical Switching

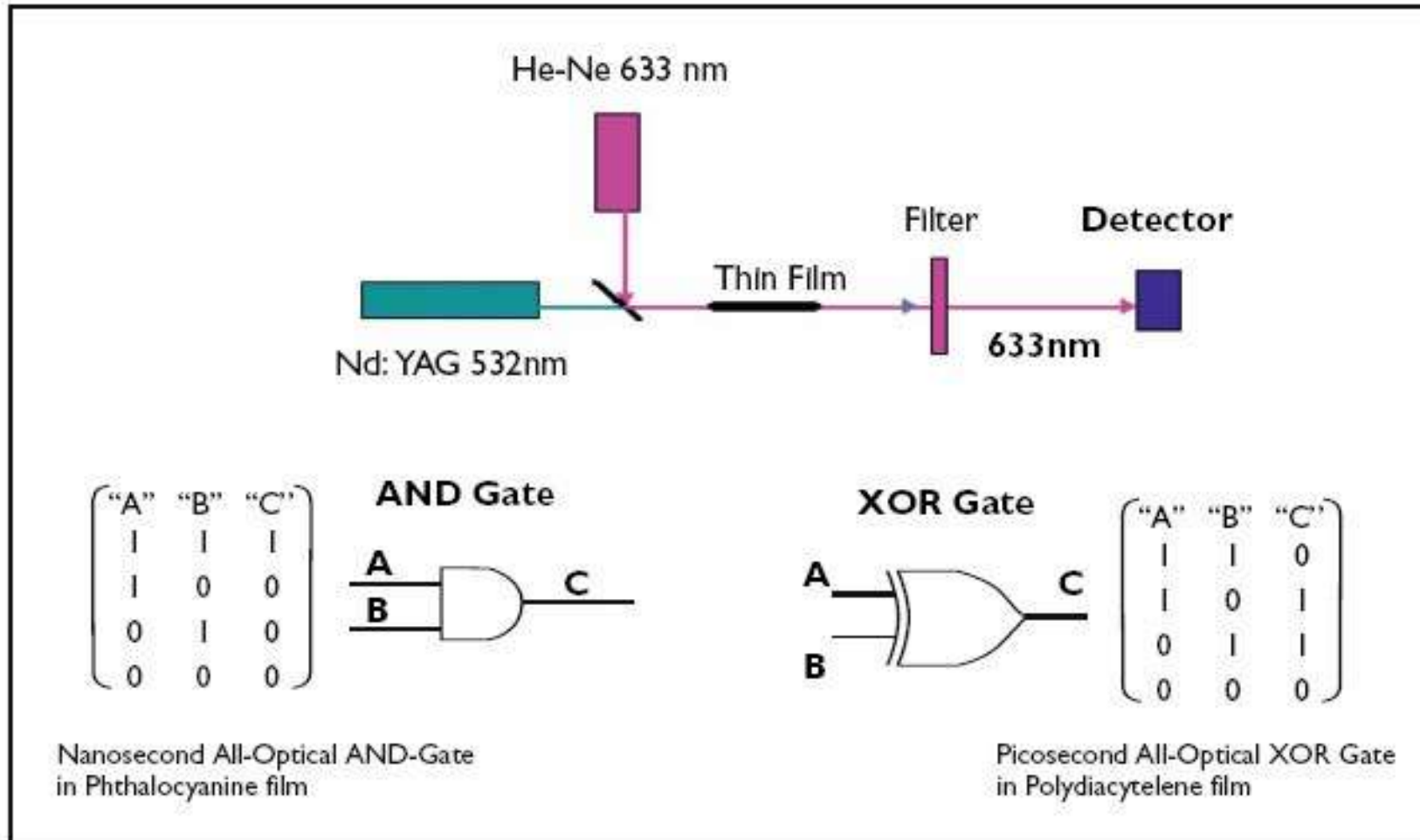
Optical Signal Processing

Optical Computing

Optical Sensors



Applications: All-Optical Computer



The Cavity Ring (CR) Resonator

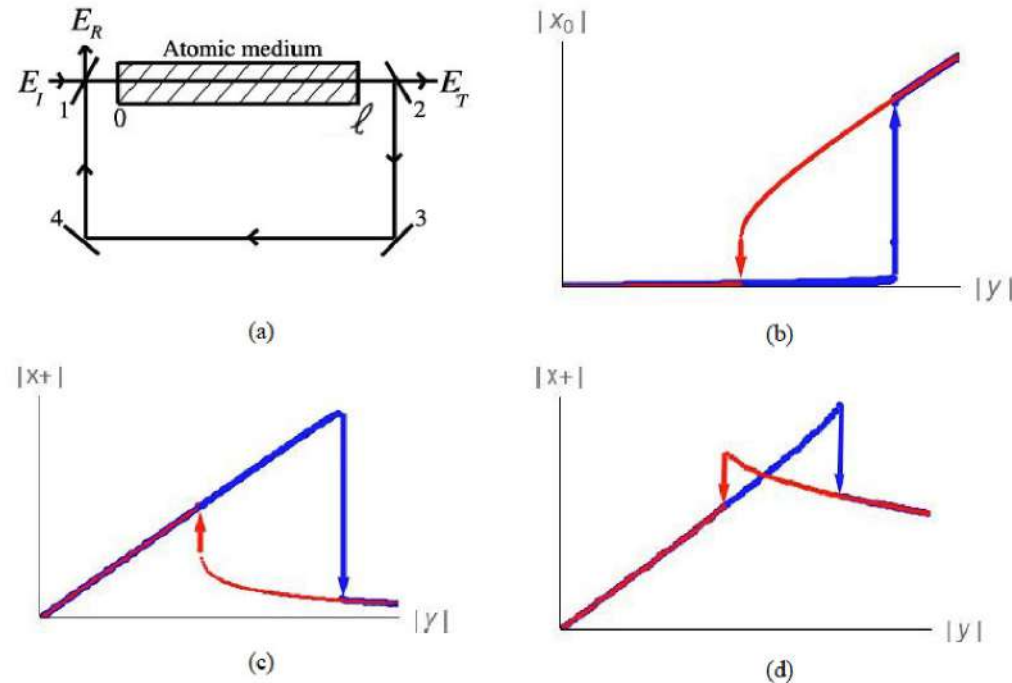


Fig. (a) A CR resonator; (b) a counterclockwise hysteresis cycle in the fundamental harmonic; (c) a clockwise hysteresis cycle in the first harmonic and (d) a butterfly hysteresis cycle.

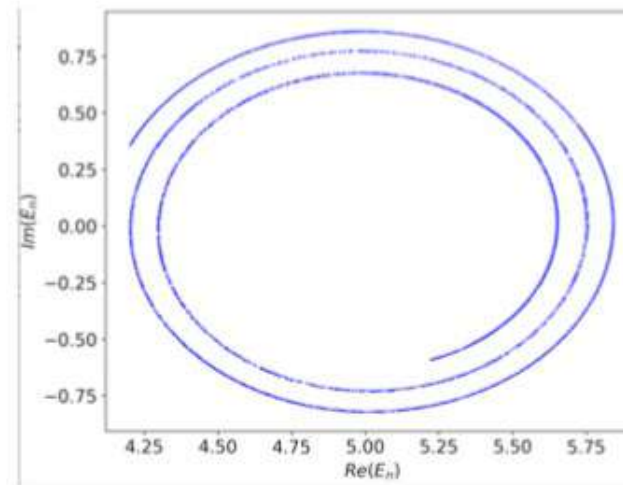
Lynch S, Alharbey RA, Hassan SS, & Batarfi HA (2015) Delayed-dynamical bistability within and without rotating wave approximation, Journal of Nonlinear Optical Physics and Materials, 24(3), 1550037.

The SFR Resonator Iterative Equation

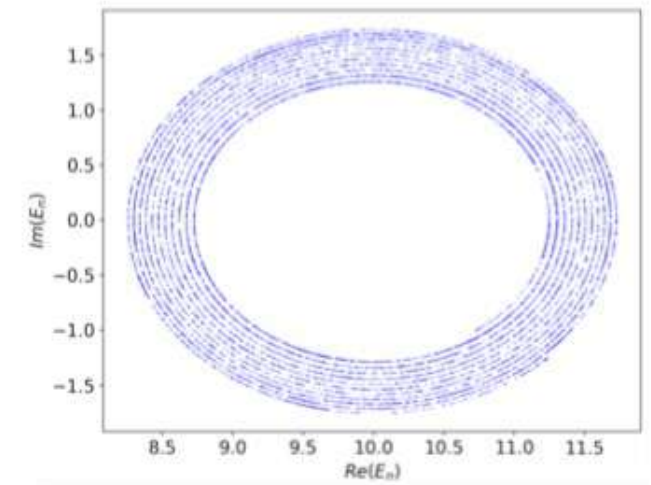
The SFR Resonator Equation

In the late 1970's, Ikeda simplified the DDEs used to model a CR to obtain the much simpler complex iterative equation:

$$E_{n+1} = A + BE_n \exp[i(|E_n|^2 + \phi_L)]$$



(a)

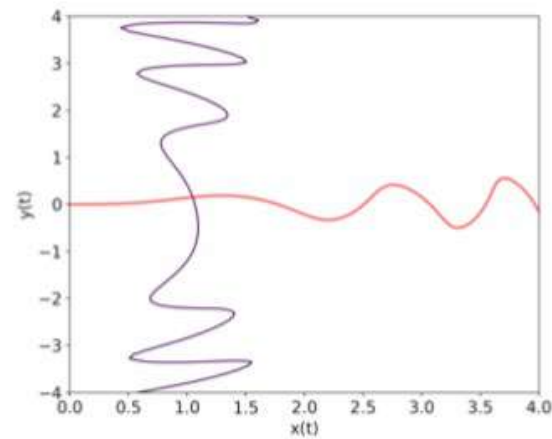


(b)

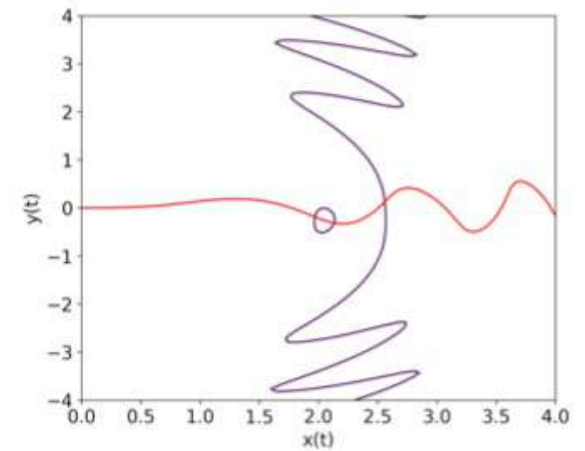
Figure 16.11: [Python] The chaotic attractors when (a) $A = 5$ (5000 iterates) and (b) $A = 10$ (5000 iterates).

SFR Python Programs: Finding Fixed Points

```
1 # Program 16a: Intersection of implicit curves.
2 # See Figure 16.10(b).
3
4 import numpy as np
5 import matplotlib.pyplot as plt
6
7 A, B = 2.2, 0.15
8
9 x, y = np.mgrid[0:4:100j, -4:4:100j]
10 z1 = A + B*x*np.cos(x**2 + y**2) - B*y*np.sin(x**2 + y**2) - x
11 z2 = B*x*np.sin(x**2 + y**2) + B*y*np.cos(x**2 + y**2) - y
12
13 fig, ax = plt.subplots()
14 plt.contour(x, y, z1, levels=[0])
15 plt.contour(x, y, z2, levels=[0], colors='r')
16 ax.set_xlabel('x(t)', fontsize=15)
17 ax.set_ylabel('y(t)', fontsize=15)
18 plt.tick_params(labelsize=15)
19 plt.show()
```



(a)

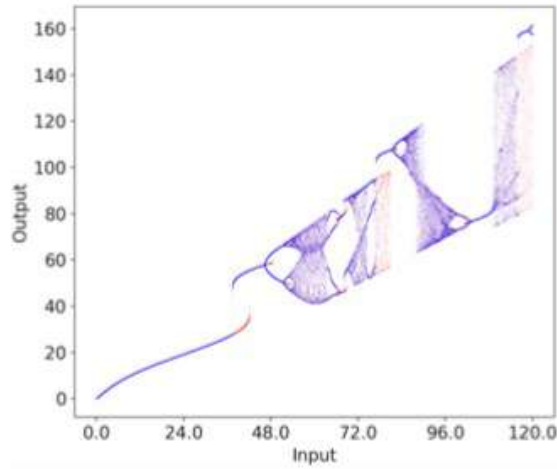


(b)

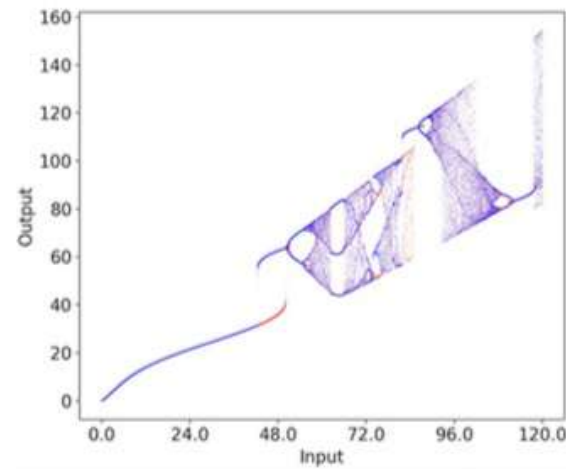
The SFR Resonator Iterative Equation: Chaotic Attractor

```
Program_16b.py*
1 # Program 16b: Iteration of the Ikeda map.
2 # See Figure 16.11(b).
3 from matplotlib import pyplot as plt
4 import numpy as np
5 # Parameters
6 A, B = 10, 0.15
7 def ikeda(X):
8     x, y = X
9     xn = A + B*x*np.cos(x**2 + y**2) - B*y*np.sin(x**2 + y**2)
10    yn = B*x*np.sin(x**2 + y**2) + B*y*np.cos(x**2 + y**2)
11    return (xn, yn)
12 X0 = [A, 0]
13 X, Y = [], []
14 for i in range(10000):
15     xn, yn = ikeda(X0)
16     X, Y = X + [xn], Y + [yn]
17     X0 = [xn, yn]
18 fig, ax = plt.subplots(figsize=(10,10))
19 ax.scatter(X, Y, color='blue', s=0.1)
20 plt.xlabel('$Re(E_n)$', fontsize=15)
21 plt.ylabel('$Im(E_n)$', fontsize=15)
22 plt.tick_params(labelsize=15)
23 plt.show()
```

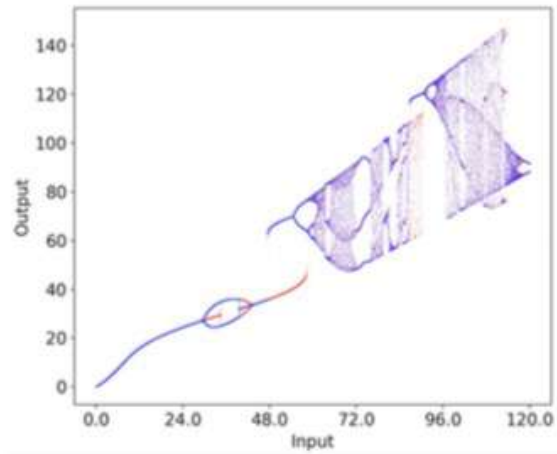

Bifurcation Diagrams of the SFR and Book on Physics of Waves



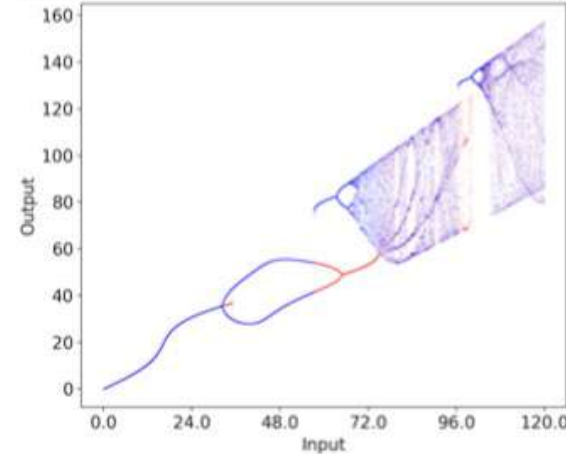
(a)



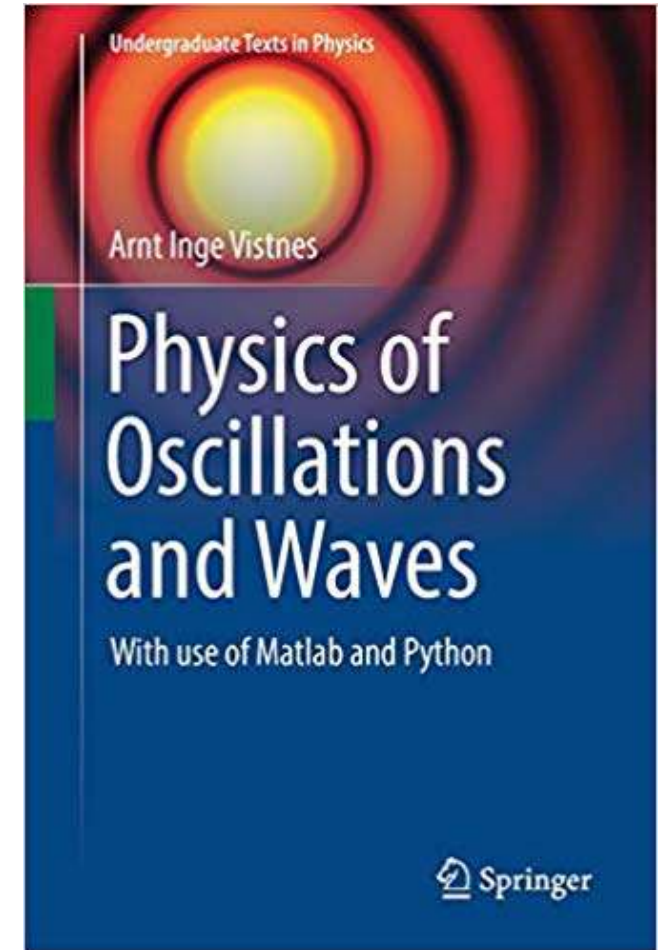
(b)



(c)



(d)



```

1  # Program_16c.py: Bifurcation diagram of the Ikeda map.
2  from matplotlib import pyplot as plt
3  import numpy as np
4  B , phi , Pmax , En = 0.15 , 0 , 10 , 0  # phi is a linear phase shift.
5  half_N = 99999
6  N = 2*half_N + 1
7  N1 = 1 + half_N
8  esqr_up, esqr_down = [], []
9  ns_up = np.arange(half_N)
10 ns_down = np.arange(N1, N)
11 # Ramp the power up
12 for n in ns_up:
13     En = np.sqrt(n * Pmax / N1) + B * En * np.exp(1j*((abs(En))**2 - phi))
14     esqr1 = abs(En)**2
15     esqr_up.append([n, esqr1])
16 esqr_up = np.array(esqr_up)
17 # Ramp the power down
18 for n in ns_down:
19     En = np.sqrt(2 * Pmax - n * Pmax / N1) + \
20         B*En* np.exp(1j*((abs(En))**2 - phi))
21     esqr1 = abs(En)**2
22     esqr_down.append([N-n, esqr1])
23 esqr_down=np.array(esqr_down)
24 fig, ax = plt.subplots()
25 xtick_labels = np.linspace(0, Pmax, 6)
26 ax.set_xticks([x / Pmax * N1 for x in xtick_labels])
27 ax.set_xticklabels(["{:0.1f}".format(xtick) for xtick in xtick_labels])
28 plt.plot(esqr_up[:, 0], esqr_up[:, 1], "r.", markersize=0.1)
29 plt.plot(esqr_down[:, 0], esqr_down[:, 1], "b.", markersize=0.1)
30 plt.xlabel("Input Power", fontsize=15)
31 plt.ylabel("Output Power", fontsize=15)
32 plt.tick_params(labelsize=15)
33 plt.show()

```

```

1  # Program 16d: Animation of the Ikeda Chaotic Attractor.
2  from matplotlib import pyplot as plt
3  from matplotlib.animation import ArtistAnimation
4  import numpy as np
5  fig=plt.figure()
6  plt.title('Animation Ikeda Chaotic Attractor')
7  plt.axis([0, 14, -2, 2])
8  B = 0.15
9  def ikeda(X):
10     x, y = X
11     xn = A + B*x*np.cos(x**2 + y**2) - B*y*np.sin(x**2 + y**2)
12     yn = B*x*np.sin(x**2 + y**2) + B*y*np.cos(x**2 + y**2)
13     return (xn, yn)
14  myimages = []
15  for A in np.arange(2, 10, 0.1):
16     X0 = [A, 0]
17     X, Y = [], []
18     for i in range(5000):
19         xn, yn = ikeda(X0)
20         X, Y = X + [xn], Y + [yn]
21         X0 = [xn, yn]
22     myimages.append(plt.plot(X, Y, 'b.', markersize=0.2))
23  my_anim = ArtistAnimation(fig, myimages, blit=False, interval=100)
24  plt.show()

```

End Day 3 Summary

Day 3			
Topics	Hours	Topics	Hours
Scientific Computing: Chemical Kinetics	10am-11am	Scientific Computing: Engineering	1pm-2pm
Scientific Computing: Fractals and Multifractals	11am-12pm	Scientific Computing: Physics	2pm-3pm

