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 3)

Day 3			
Fractals and Multifractals	10am-11am	Physics and Statistics	2pm-3pm
Image Processing	11am-12pm	Brain-Inspired Computing	3pm-4pm
Numerical Methods ODEs/PDEs	12pm-1pm		

Download all files from GitHub:

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

Solutions to the Exercises in Section 2:

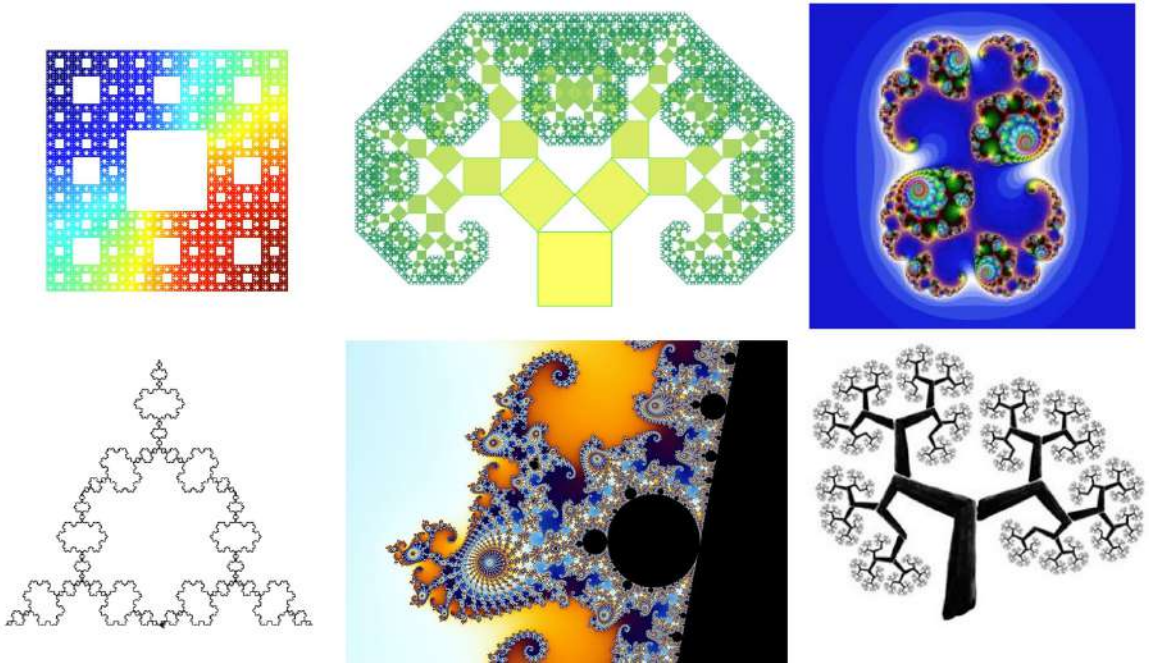
https://drstephenlynch.github.io/webpages/Solutions_Section_2.html

Fractals and Multifractals: Start Session 1

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

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

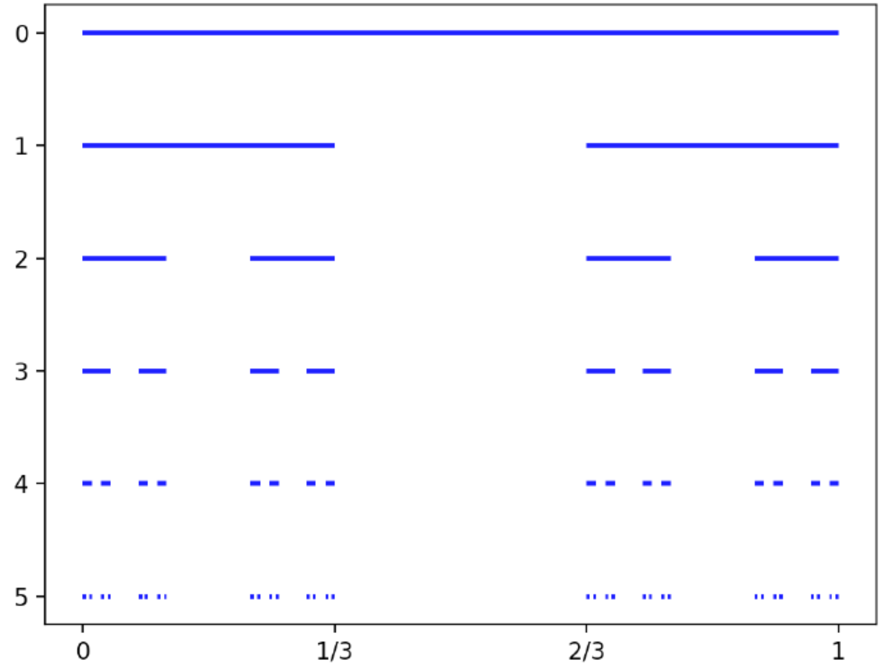
Fractals in Nature



Mathematical Fractals

Fractals: The Cantor Set

```python
# The Cantor Set.
import numpy as np
import matplotlib.pyplot as plt

line, stage = [0 , 1] ,  5
def cantor(line, level = 0):
    plt.plot(line, [level ,level], color = "b", lw = 2,\
             solid_capstyle = "butt")
    if level < stage:
        segment = np.linspace(line[0] , line[1] , 4)
        cantor(segment[:2], level + 1)
        cantor(segment[2:], level + 1)

cantor(line)
plt.gca().invert_yaxis()
plt.xticks([0, 1/3, 2/3, 1],['0', '1/3', '2/3', '1'])
plt.show()
```

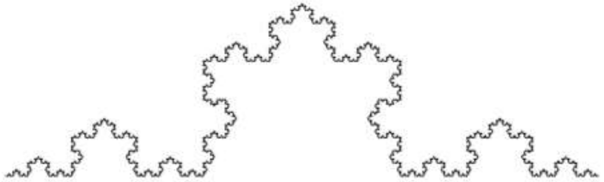


Problem: Edit the program to plot a Cantor set where the two middle fifth segments are removed at each stage.

Fractals: The Koch Curve Fractal

```python
# Programs 17a: Plotting the Koch curve.
# See Figure 17.2.

import numpy as np
import matplotlib.pyplot as plt
from math import floor

k=6
n_lines = 4**k
h = 3**(-k);
x = [0]*(n_lines+1)
y = [0]*(n_lines+1)
x[0], y[0] = 0, 0

segment=[0] * n_lines;

# The angles of the four segments.
angle=[0, np.pi/3, -np.pi/3, 0]
for i in range(n_lines):
    m=i
    ang=0
    for j in range(k):
        segment[j] = np.mod(m, 4)
        m = floor(m / 4)
        ang = ang + angle[segment[j]]

    x[i+1] = x[i] + h*np.cos(ang)
    y[i+1] = y[i] + h*np.sin(ang)

plt.axis('equal')
plt.plot(x,y)
plt.show()
```

Stage 1



Stage 5

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

Barnsley's Fern

```python
# Program 17c: Barnsley's fern.
# See Figure 17.7.
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm

# The transformation T
f1 = lambda x, y: (0.0, 0.2*y)
f2 = lambda x, y: (0.85*x + 0.05*y, -0.04*x + 0.85*y + 1.6)
f3 = lambda x, y: (0.2*x - 0.26*y, 0.23*x + 0.22*y + 1.6)
f4 = lambda x, y: (-0.15*x + 0.28*y, 0.26*x + 0.24*y + 0.44)
fs = [f1, f2, f3, f4]

num_points = 60000

width = height = 300
fern = np.zeros((width, height))

x, y = 0, 0
for i in range(num_points):
    # Choose a random transformation
    f = np.random.choice(fs, p=[0.01, 0.85, 0.07, 0.07])
    x, y = f(x,y)
    # Map (x,y) to pixel coordinates
    # Center the image
    cx, cy = int(width / 2 + x * width / 10), int(y * height / 10)
    fern[cy, cx] = 1

fig, ax=plt.subplots(figsize=(8,8))
plt.imshow(fern[::-1,:], cmap=cm.Greens)
ax.axis('off')
plt.show()
```

Box Counting Dimensions: Program_11d.py

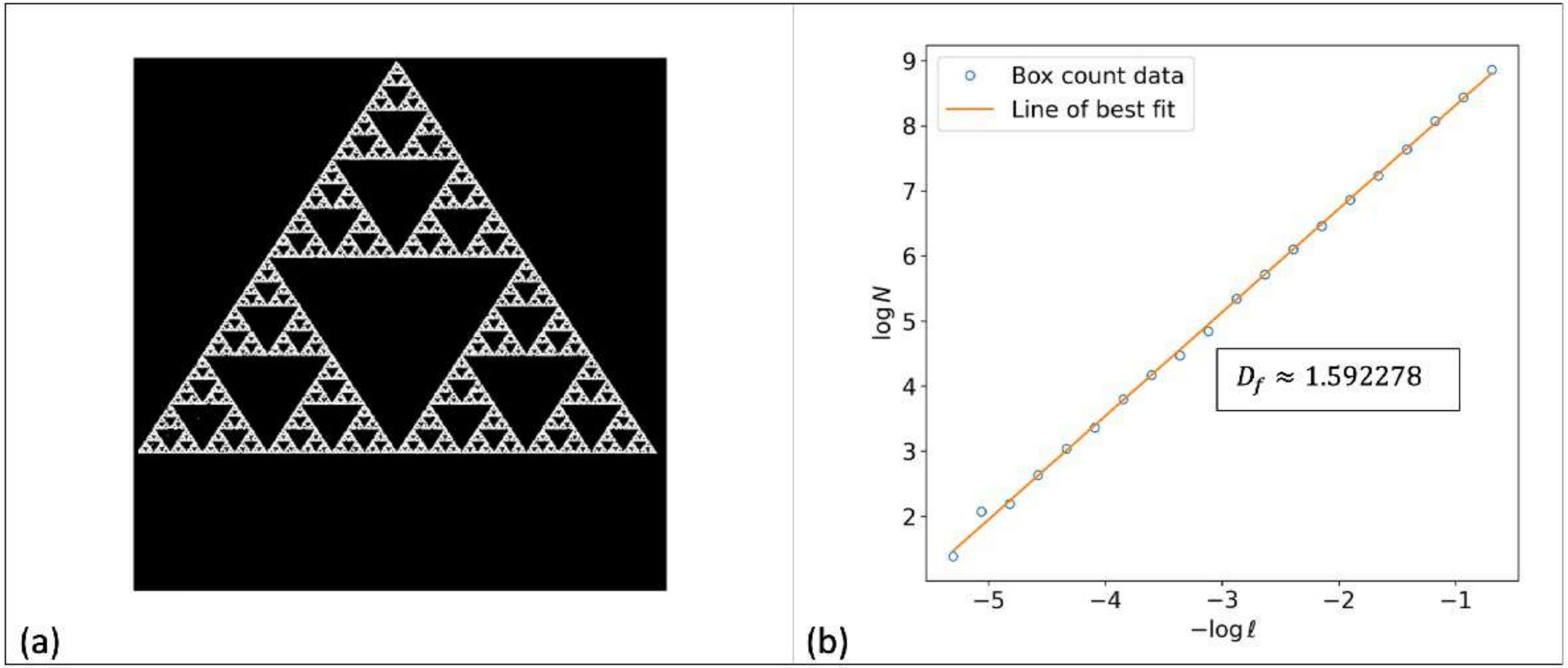


Figure 11.4 (a) A padded binary image of the Sierpinski triangle. (b) The line of best fit for the data points. The gradient of the line gives the fractal dimension, $D_f \approx 1.592278$, of the figure displayed in (a).

Multifractal Cantor Set

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

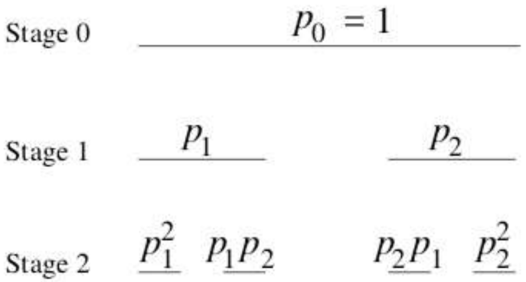


Figure 17.14: The weight distribution on a Cantor multifractal set up to stage 2.

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

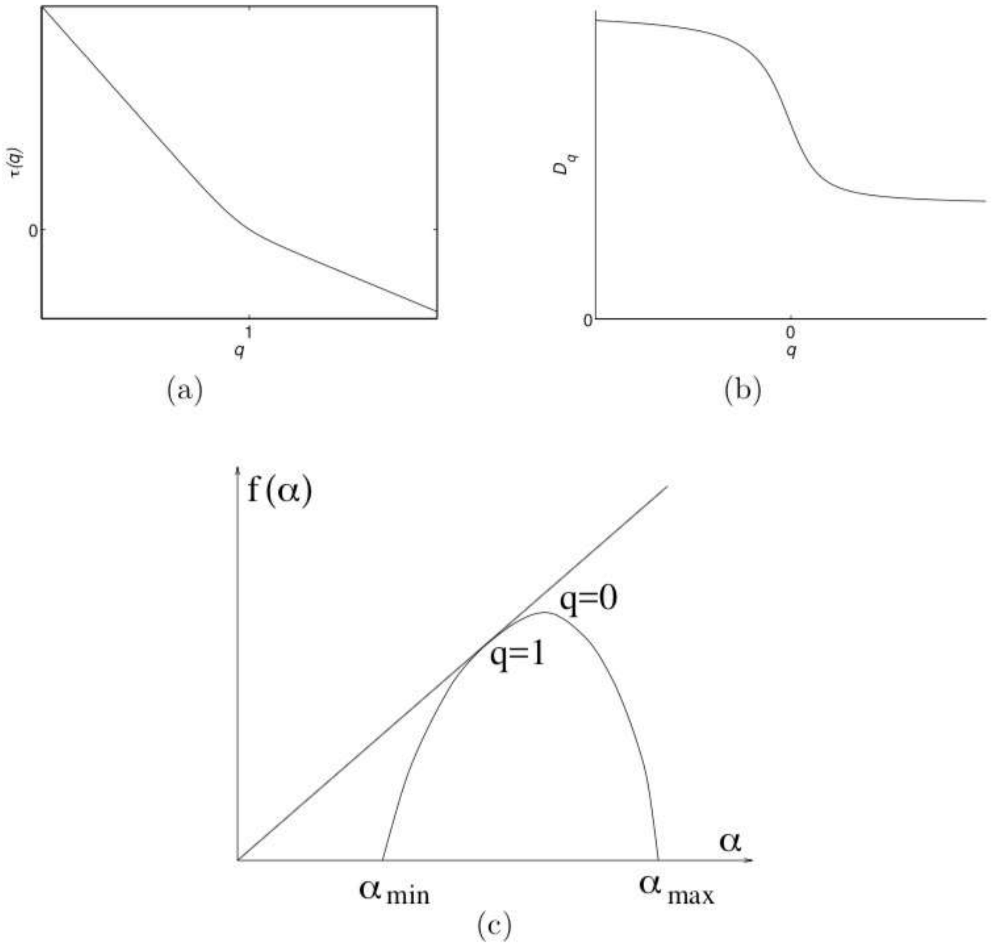
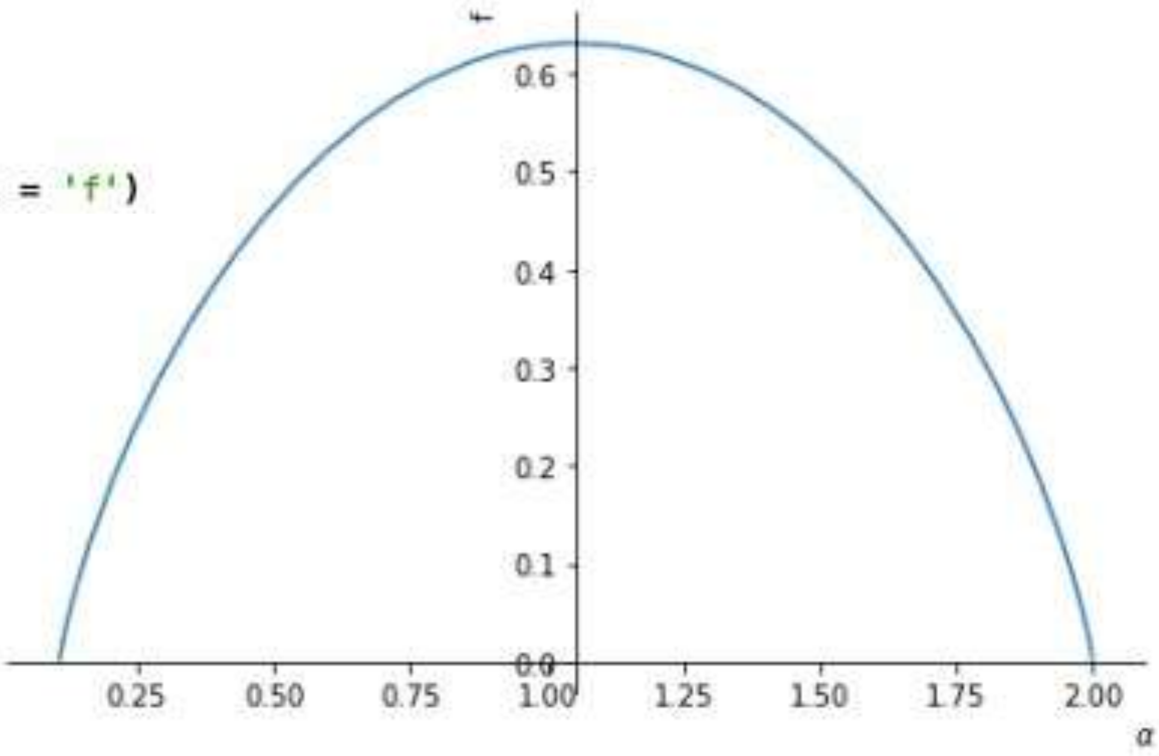


Figure 17.13: Typical curves of (a) the $\tau(q)$ function, (b) the D_q spectrum, and (c) the $f(\alpha)$ spectrum. In case (c), points on the curve near $\alpha_{\min}$ correspond to values of $q \to \infty$, and points on the curve near $\alpha_{\max}$ correspond to values of $q \to -\infty$.

Multifractals: $f(\alpha)$ $Curve$

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



Manchester Metropolitan University

Multifractal Grid

p_1	p_2
p_3	p_4

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

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

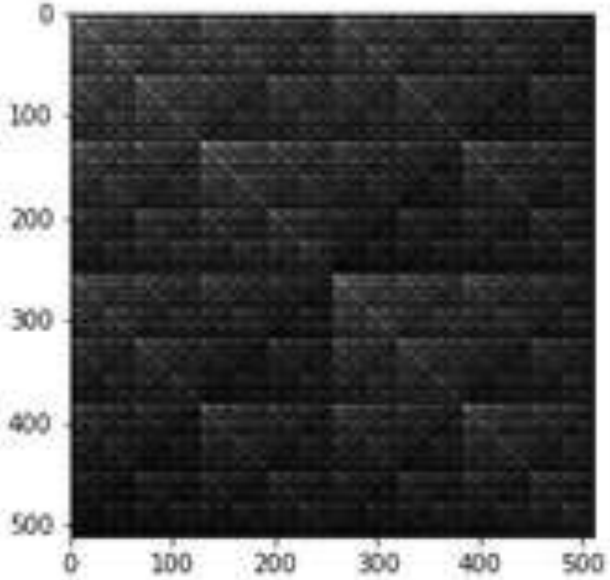
$$\tau = \frac{ln\left(p_1^q + p_2^q + p_3^q + p_4^q\right)}{ln(2)}, \ \alpha = -\frac{d\tau}{dq}, \ f(\alpha) = \alpha q + \tau .$$

Multifractals to Measure Dispersion and Clustering

Multifractal motif

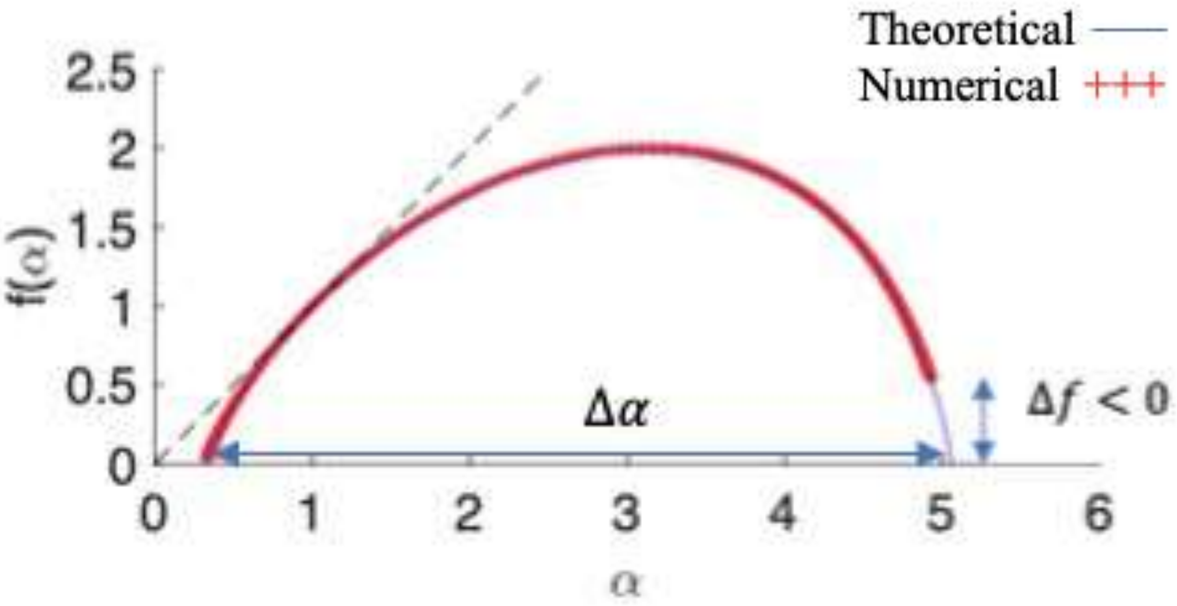
0.8	0.1
0.03	0.07

Representative figure



Clusters of dark pixels.

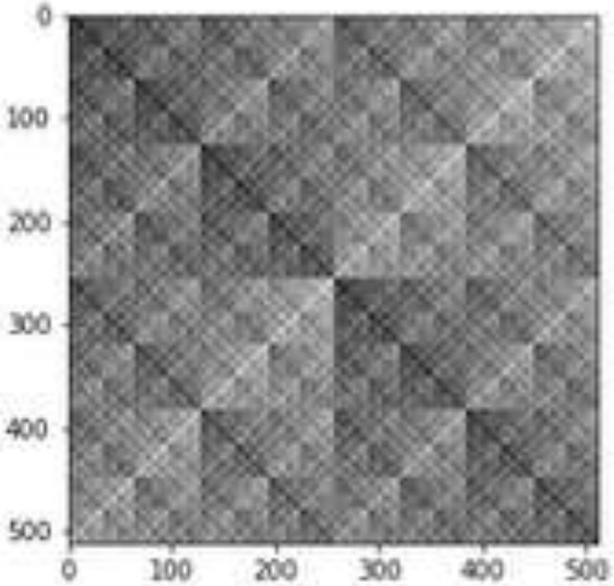
Multifractal f-α curve



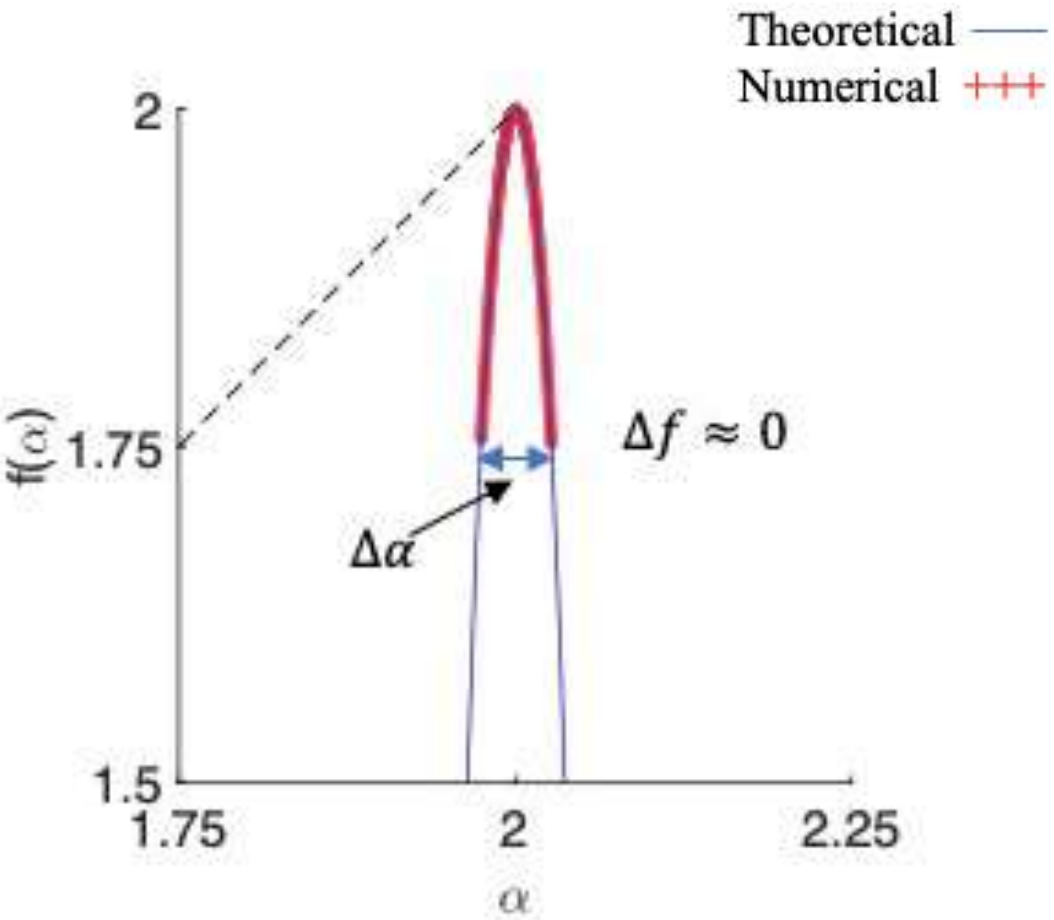
The f-α curve is skewed left and $\Delta f < 0$.

Multifractals to Measure Dispersion and Clustering

0.24	0.26
0.255	0.245



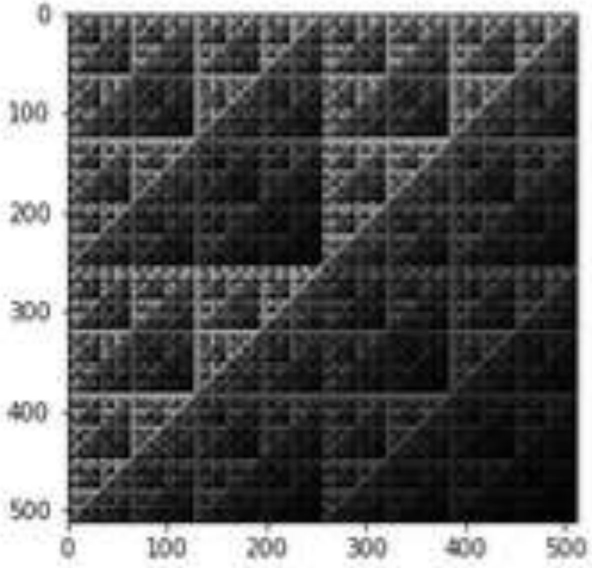
Homogeneous – no clusters.



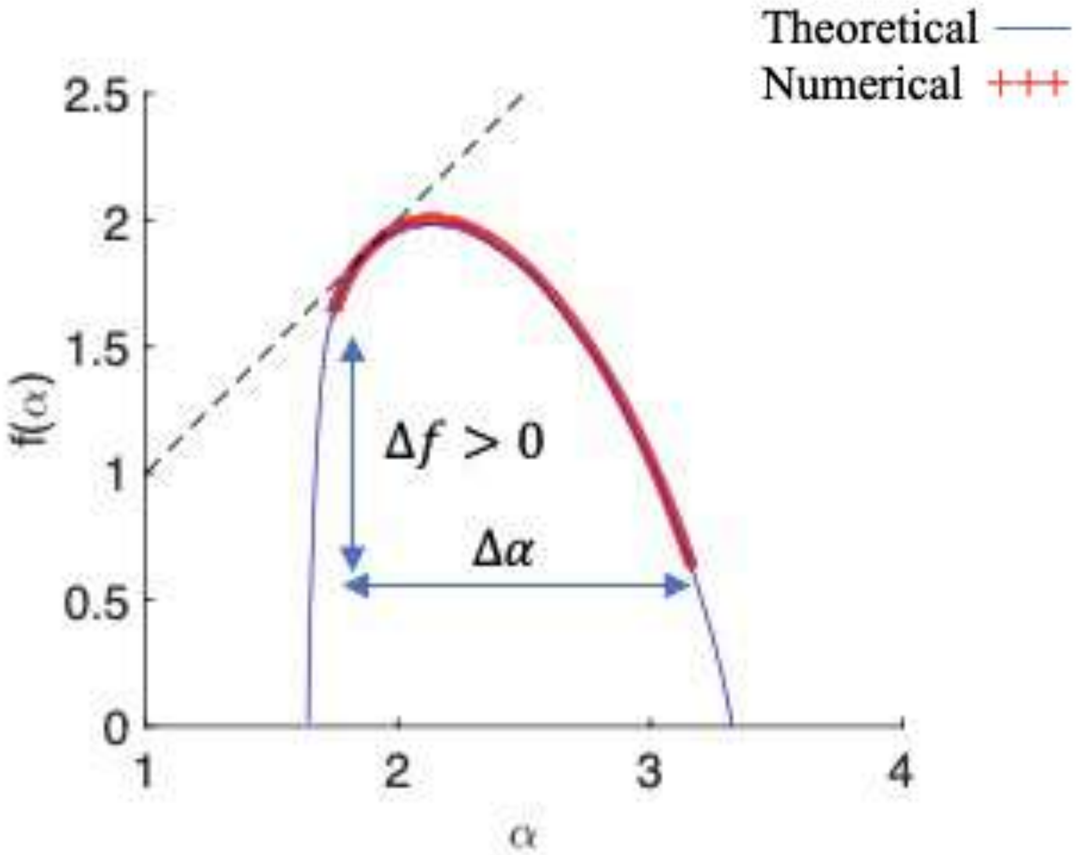
The f-α curve is not skewed and $\Delta f \approx 0$.

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 $\Delta f > 0$.

Multifractals in the Real World: End Session 1

Evans A, Slate AJ, Tobin M, **Lynch S**, Wilson-Nieuwenhuis J, Verran J, Kelly P and Whitehead KA (2022) Multifractal analysis to determine the effect of surface topography on the distribution, density, dispersion and clustering of differently organised coccal shaped bacteria. Antibiotics 11(5) 11050551.

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.

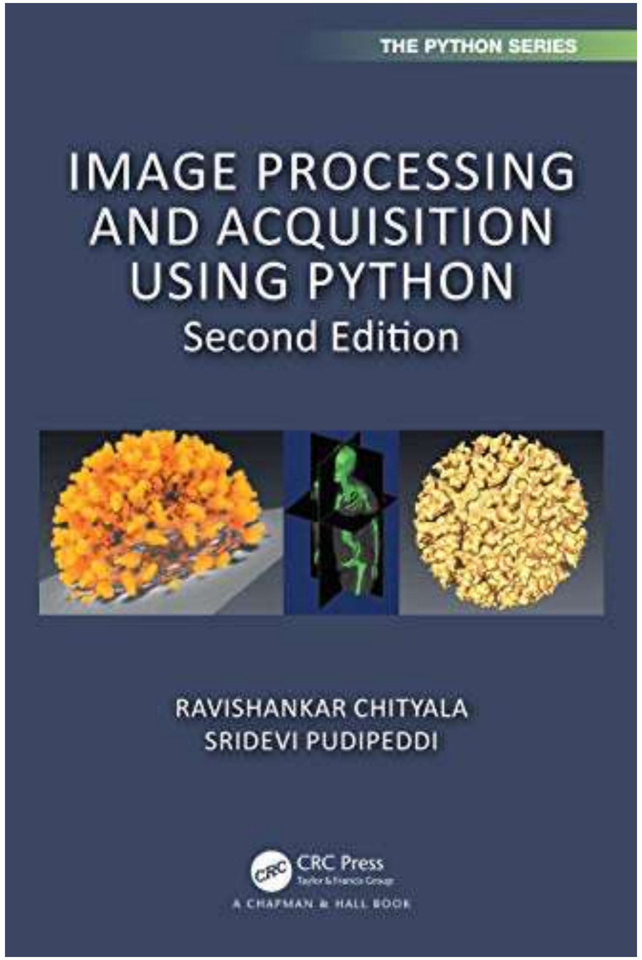
Drozdz S, Kowalski R, Oswiecimka P, Rak R and Gebarowski R (2018) Dynamical variety of shapes in financial multifractality, Complexity 7015721, 1-13.

Image Processing: Session 2

scikit-image

image processing in python

https://scikit-image.org



Manchester Metropolitan University

Image Processing in Python: scikit-image

scikit-image
image processing in python

Installation Gallery Documentation Community Source

Search documentation ...

Docs for 0.19.0
All versions

User Guide

- Getting started
- A crash course on NumPy for images
  - NumPy indexing
  - Color images
  - Coordinate conventions
  - Notes on the order of array dimensions
  - A note on the time dimension
- Image data types and what they mean
  - Input types
  - Output types
  - Working with OpenCV
  - Image processing pipeline
  - Rescaling intensity values
  - Note about negative values
  - References
- I/O Plugin Infrastructure
- Handling Video Files
  - A Workaround: Convert the Video to an Image Sequence
  - PyAV
  - Adding Random Access to PyAV
  - MoviePy
  - Imageio
  - OpenCV
- Data visualization
  - Matplotlib
  - Plotly
  - Mayavi
  - Napari
- Image adjustment: transforming image content
  - Color manipulation
  - Contrast and exposure
- Geometrical transformations of images
  - Cropping, resizing and rescaling images
  - Projective transforms (homographies)
- Tutorials
  - Image Segmentation
  - How to parallelize loops
- Getting help on using `skimage`
  - Examples gallery
  - Search field
  - API Discovery
  - Docstrings
  - Mailing-list
- Image Viewer
  - Quick Start

Image Processing: Generating a Multifractal Image

```python
# Program 18a: Generating a multifractal image.
# Save the image.
# See Figure 18.1(b).

import numpy as np
import matplotlib.pyplot as plt
from skimage import exposure, io, img_as_uint

p1, p2, p3, p4 = 0.3, 0.4, 0.25, 0.05
p = [[p1, p2], [p3, p4]]
for k in range(1, 9, 1):
    M = np.zeros([2 ** (k + 1), 2 ** (k + 1)])
    M.tolist()
    for i in range(2**k):
        for j in range(2**k):
            M[i][j] = p1 * p[i][j]
            M[i][j + 2**k] = p2 * p[i][j]
            M[i + 2**k][j] = p3 * p[i][j]
            M[i + 2**k][j + 2**k] = p4 * p[i][j]
    p = M

# Plot the multifractal image.
M = exposure.adjust_gamma(M, 0.2)
plt.imshow(M, cmap='gray', interpolation='nearest')

# Save the image as a portable network graphics (png) image.
im = np.array(M, dtype='float64')
im = exposure.rescale_intensity(im, out_range='float')
im = img_as_uint(im)
io.imsave('Multifractal.png', im)
io.show()
```

0.3	0.4
0.25	0.05

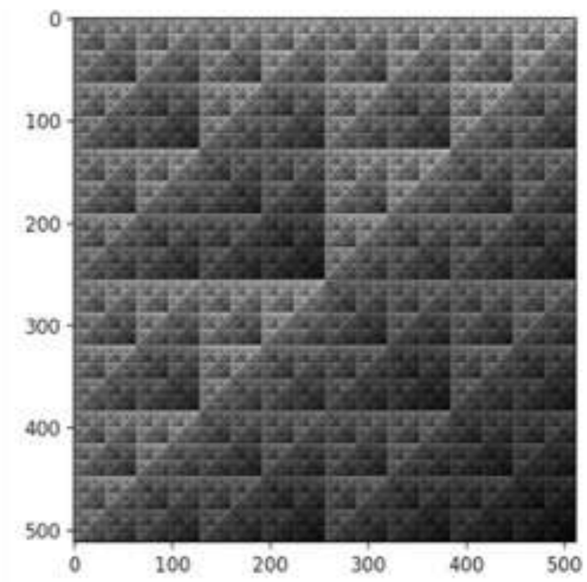
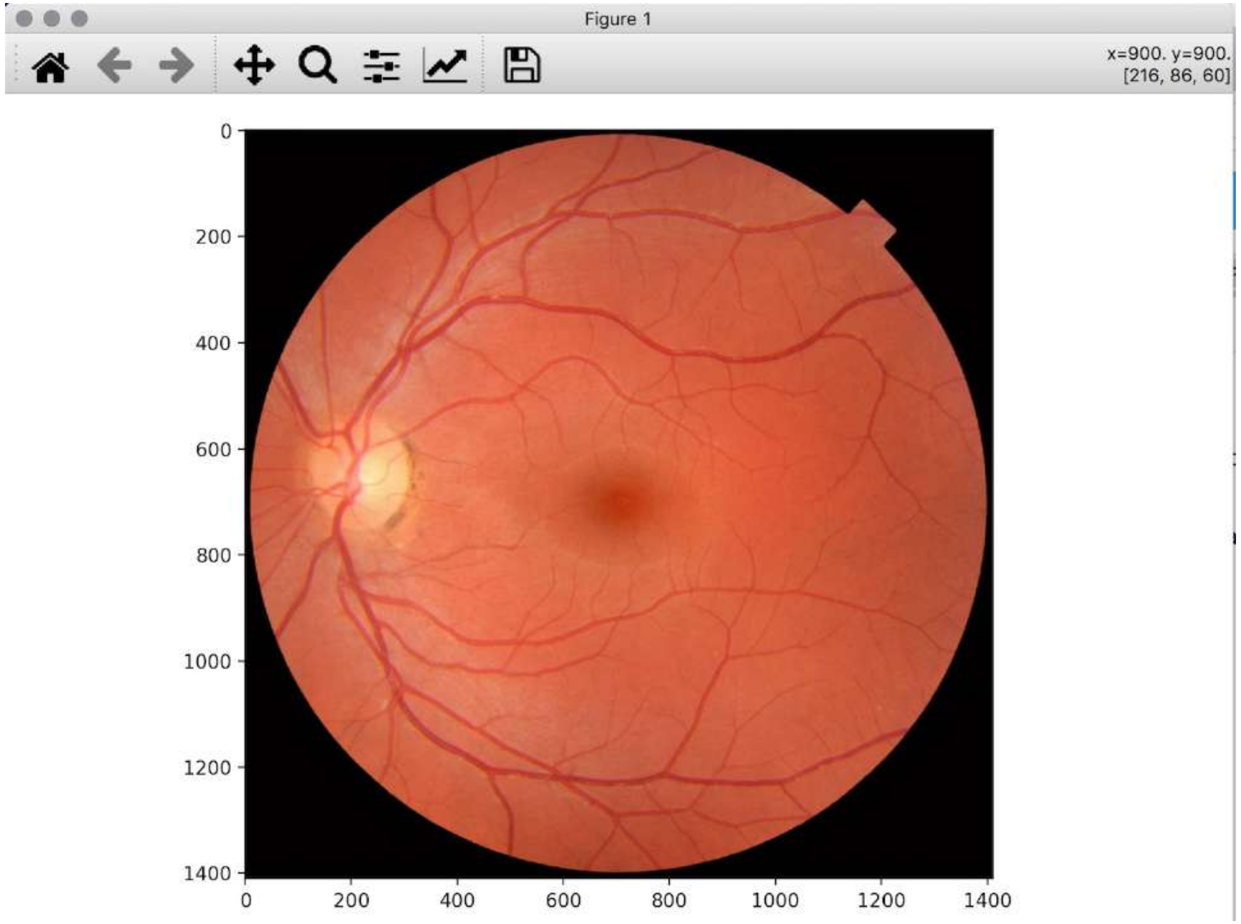


Image Processing: Colour Image

Console 4/A

```
Python 3.9.7 (default, Sep 16 2021, 08:50:36)
Type "copyright", "credits" or "license" for more
information.

IPython 7.29.0 -- An enhanced Interactive Python.

In [1]: from skimage import data, io

In [2]: retina = data.retina()

In [3]: io.imshow(retina)
Out[3]: <matplotlib.image.AxesImage at 0x12ec9a6d0>

In [4]: retina.shape
Out[4]: (1411, 1411, 3)

In [5]: retina.dtype
Out[5]: dtype('uint8')

In [6]: retina[900 , 900]
Out[6]: array([216,  86,  60], dtype=uint8)
```

IPython console History



Manchester Metropolitan University

Image Processing: Binarize a Color Image

```
# Program_12b.py: Counting colored pixels.
from skimage import io
import numpy as np
import matplotlib.pyplot as plt
peppers = io.imread("peppers.jpeg")
plt.figure(1)
io.imshow(peppers)
print("Image Dimensions=" , peppers.shape)
print("peppers[100,100]=",peppers[400,400])
Red = np.zeros((700,700))
for i in range(700):
    for j in range(700):
        if peppers[j,i,0]>190 and peppers[j,i,1]<120 \
            and peppers[j,i,2]<170:
            Red[j,i]=1
        else:
            Red[j,i]=0
plt.figure(2)
plt.imshow(Red,cmap="gray")
pixel_count = int(np.sum(Red))
print("There are {:,} red pixels".format(pixel_count))
```

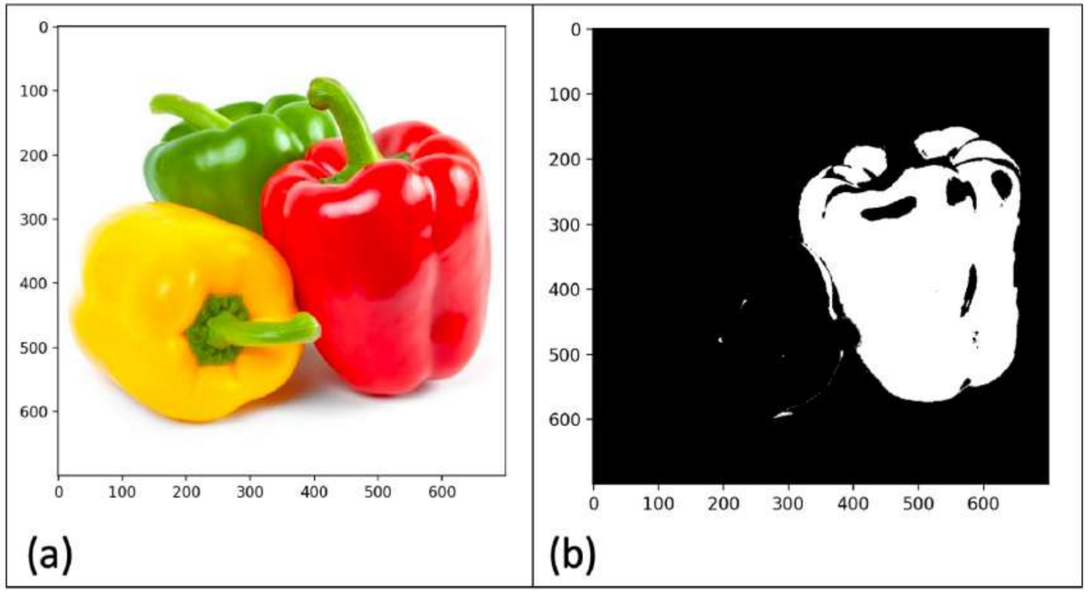


Image Processing: Statistical Analysis on an Image of Microbes

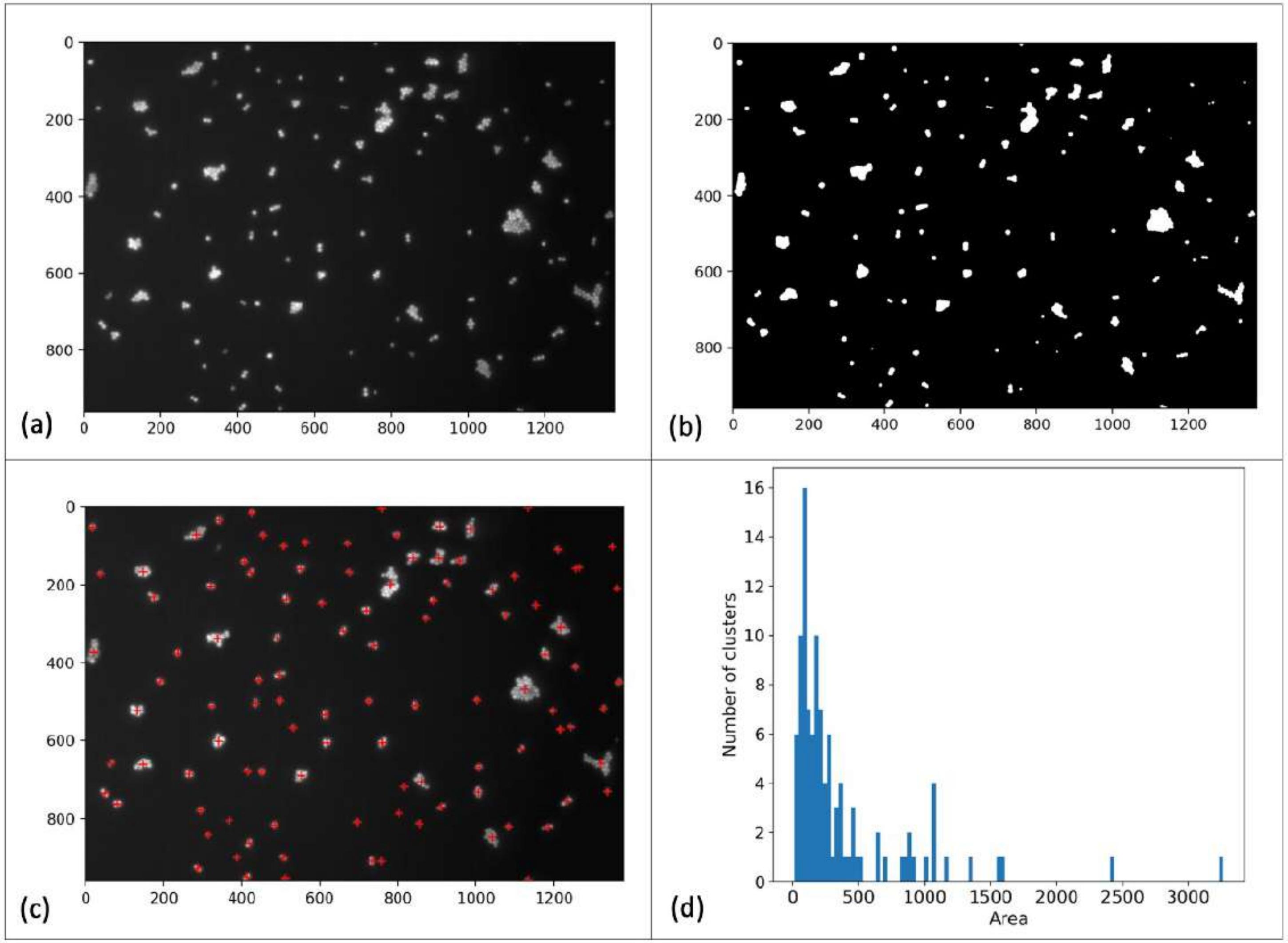


Fig. (a) Original image; (b) binarized image; (c) centroids of clusters; (d) histogram, clusters against areas.

Image Processing: Program_12c.py

```
# Program_12c.py: Statistical Analysis on Microbes.
import matplotlib.pyplot as plt
from skimage import io , measure
import numpy as np
from skimage.measure import regionprops
from scipy import ndimage
from skimage import feature
microbes_img = io.imread("Microbes.png")
fig1 = plt.figure() # Original image.
plt.imshow(microbes_img,cmap="gray", interpolation="nearest")
width, height, _ = microbes_img.shape
fig2 = plt.figure() # Binary image.
binary = np.zeros((width, height))
for i, row in enumerate(microbes_img):
    for j, pixel in enumerate(row):
        if pixel[0] > 80:
            binary[i, j] = 1
plt.imshow(binary,cmap="gray")
print("There are {:,} white pixels".format(int(np.sum(binary))))
blobs = np.where(binary>0.5, 1, 0)
labels, no_objects = ndimage.label(blobs)
props = regionprops(blobs)
print("There are {:,} clusters of cells:".format(no_objects))
```

```
# fig3. Centroids of the clusters.
object_labels = measure.label(binary)
some_props=measure.regionprops(object_labels)
fig,ax = plt.subplots(1,1)
#plt.axis('off')
ax.imshow(microbes_img,cmap="gray")
centroids = np.zeros(shape=(len(np.unique(labels)),2))
for i , prop in enumerate(some_props):
    my_centroid = prop.centroid
    centroids[i,:]=my_centroid
    ax.plot(my_centroid[1],my_centroid[0],"r+")
#print(centroids)
fig4 = plt.figure() # Histogram of the data.
labeled_areas = np.bincount(labels.ravel())[1:]
print(labeled_areas)
plt.hist(labeled_areas,bins=no_objects)
plt.xlabel("Area",fontsize=15)
plt.ylabel("Number of clusters",fontsize=15)
plt.tick_params(labelsize=15)
fig5 = plt.figure() # Canny edge detector.
edges=feature.canny(binary,sigma=2,low_threshold=0.5)
plt.imshow(edges,cmap=plt.cm.gray)
plt.show()
```

Image Processing: Vascular Architecture of an Eye: Program_12d.py

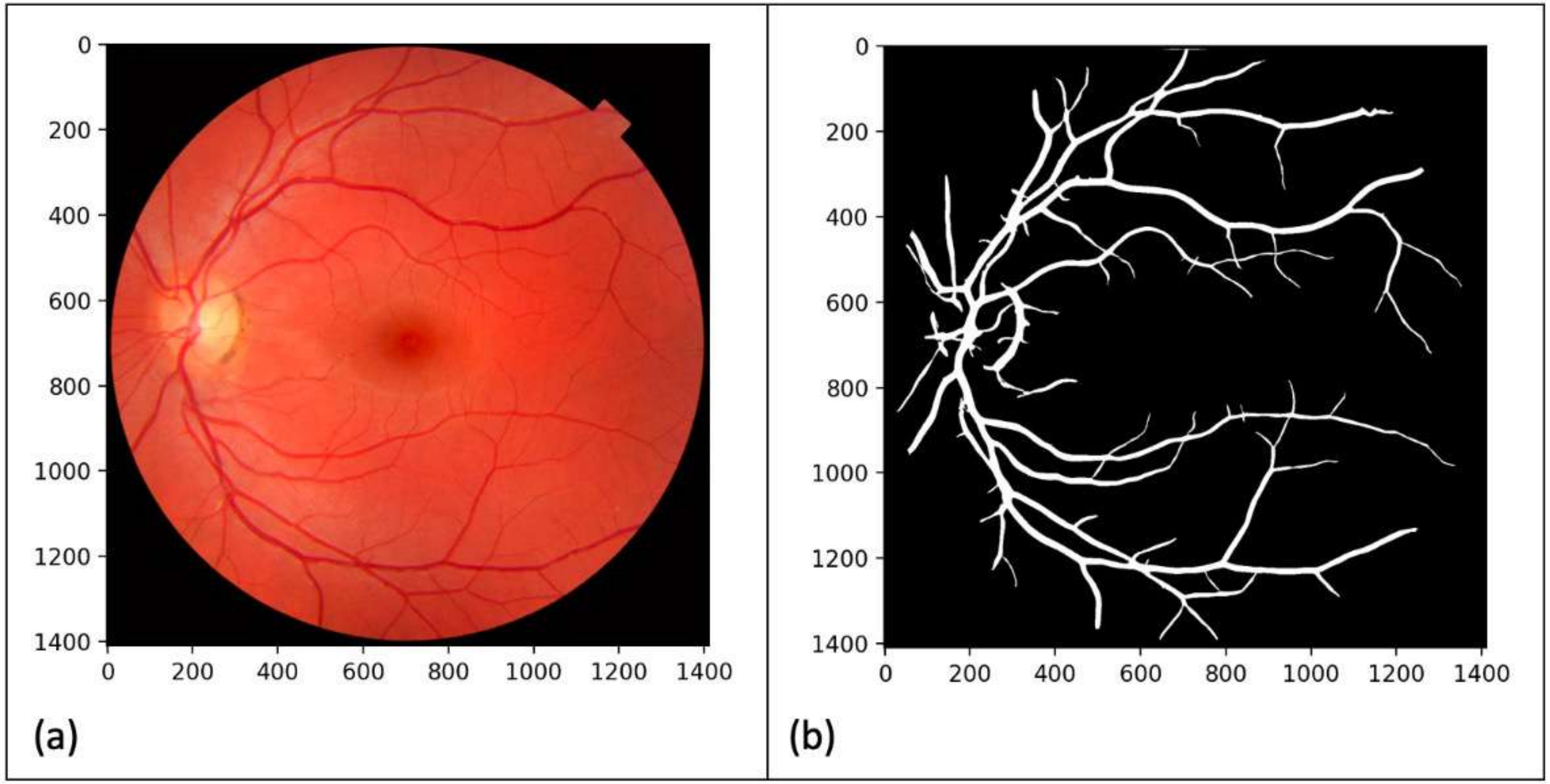


Figure 12.4 (a) The original image of a human retina. (b) Vascular architecture tracing using the sato ridge filter.

Image Processing: Brain Tumour: Program_12e.py: End Session 2

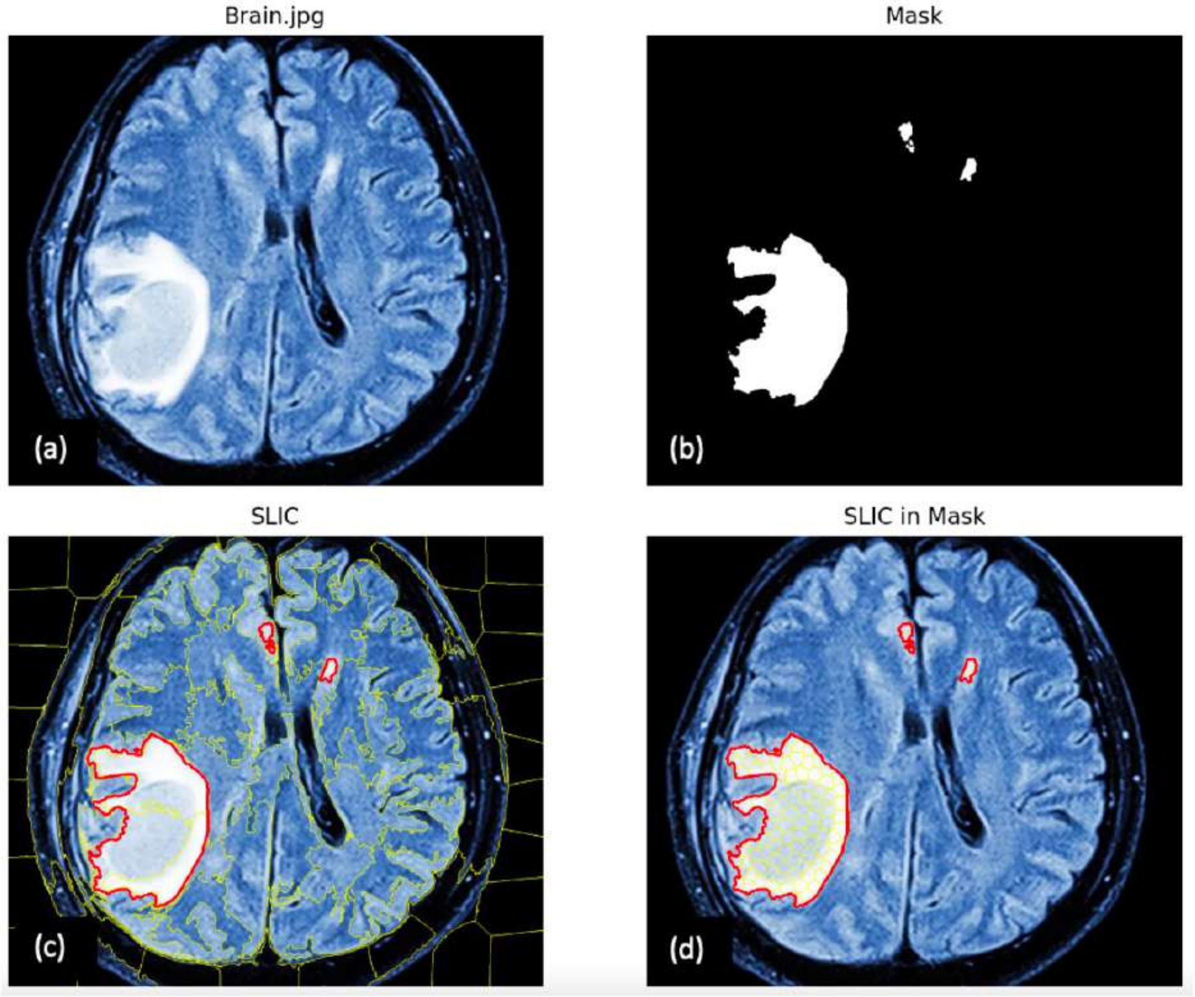


Figure 12.5 (a) The original image of a human brain with a tumour. (b) Compute a mask to identify the tumour. (c) SLIC image. (d) SLIC in the Mask.

Manchester Metropolitan University

Numerical Methods ODEs and PDEs: Start Session 3

Use a Taylor series expansion to approximate a solution to an ODE:

$$\frac{dy}{dx} = f(x, y(x)), \quad y\left(x_0\right) = y_0. \tag{13.1}$$

$$y\left(x_n + h\right) = y\left(x_n\right) + h\frac{dy}{dx}\left(x_n\right) + \frac{1}{2}h^2\frac{d^2y}{dx^2}\left(x_n\right) + \mathcal{O}\left(h^3\right), \tag{13.2}$$

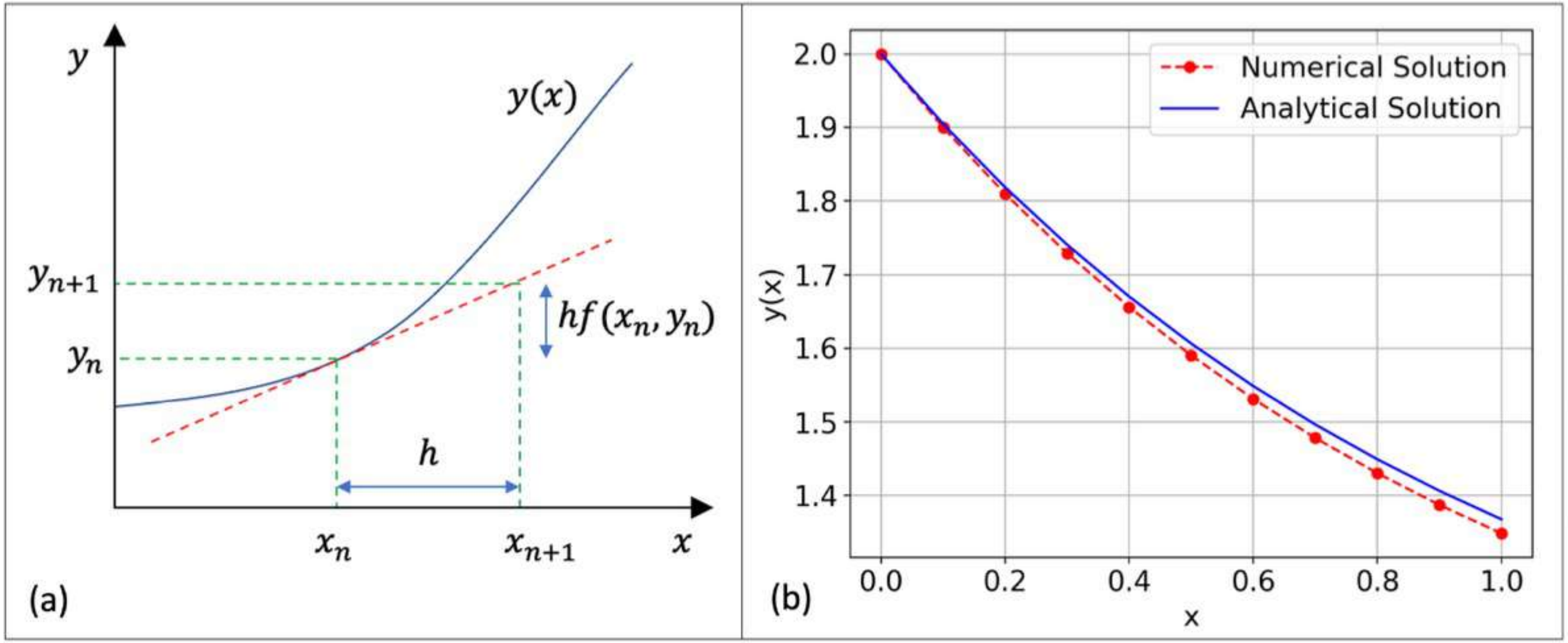
$$y\left(x_n + h\right) = y\left(x_{n+1}\right) = y\left(x_n\right) + hf\left(x_n, y_n\right).$$

Definition 13.1.1. Euler's explicit iterative formula to solve an IVP of the form (13.1), is defined by:

$$y_{n+1} = y_n + hf\left(x_n, y_n\right). \tag{13.3}$$

This iterative formula is easily implemented in Python.

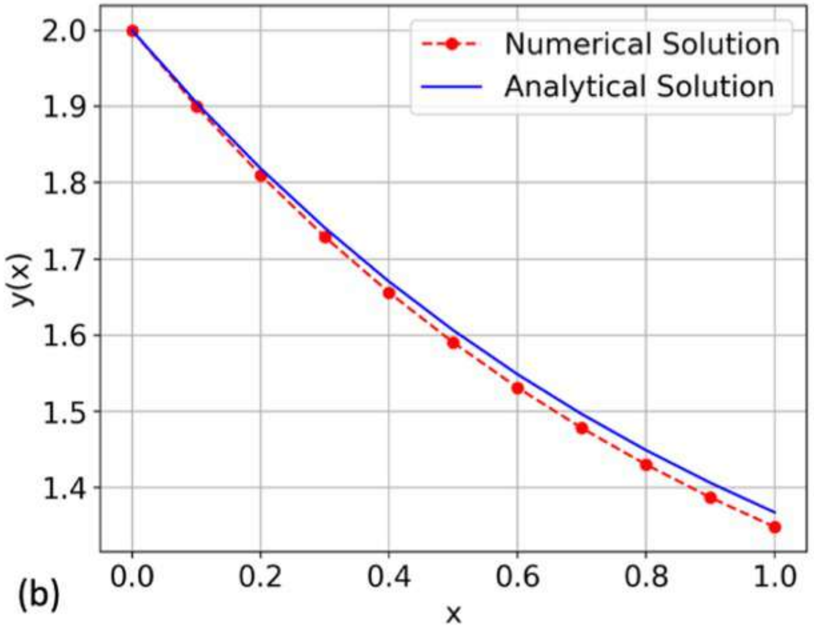
Numerical Methods: Euler's Method



Numerical Methods: Euler's Method

```
# Program_13a.py: Eulers Method for an IVP.
import numpy as np
import matplotlib.pyplot as plt
f = lambda x, y: 1 - y      # The ODE.
h , y0 = 0.1 , 2            # Step size and y(0).
x = np.arange(0, 1 + h, h) # Numerical grid.
y , y[0] = np.zeros(len(x)) , y0
for n in range(0, len(x) - 1):
    y[n + 1] = y[n] + h*f(x[n] , y[n])
plt.rcParams["font.size"] = "16"
plt.figure()
plt.plot(x, y, "ro--", label='Numerical Solution')
plt.plot(x, np.exp(-x) + 1, "b", label="Analytical Solution")
plt.xlabel("x")
plt.ylabel("y(x)")
plt.grid()
plt.legend(loc="upper right")
plt.show()
```

$$\frac{dy}{dx} = 1 - y(x), \quad y(0) = 2.$$



Manchester Metropolitan University

Numerical Methods: Runge-Kutta Method RK4: Program_13b.py

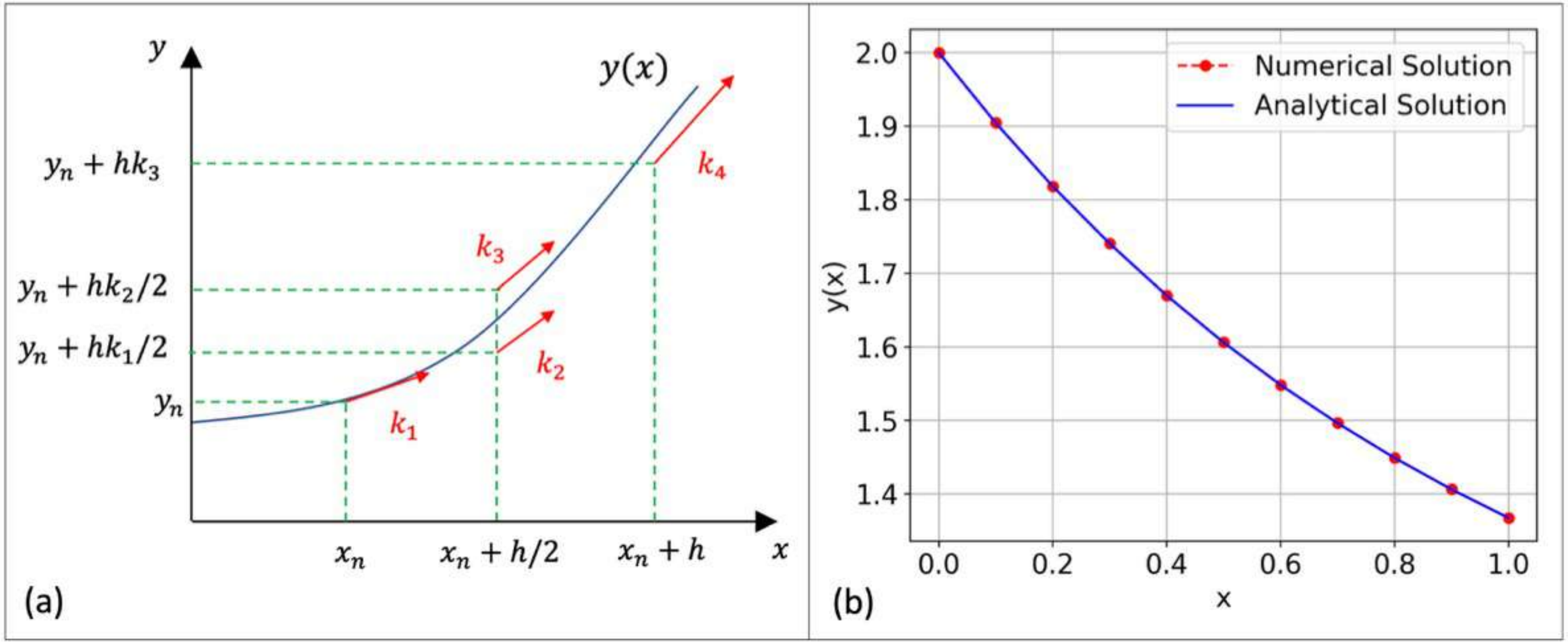


Figure 13.2 (a) Geometrical interpretation of the RK4 method. (b) Numerical and analytical solution of the IVP (13.4) using RK4.

Numerical Methods: PDEs Finite Difference Approximations

Partial Derivative	FDA	Order and Type
$\frac{\partial U}{\partial x} = U_x$	$\frac{U_{i+1}^n - U_i^n}{\Delta x}$	First Order Forward
$\frac{\partial U}{\partial x} = U_x$	$\frac{U_i^n - U_{i-1}^n}{\Delta x}$	First Order Backward
$\frac{\partial U}{\partial x} = U_x$	$\frac{U_{i+1}^n - U_{i-1}^n}{2\Delta x}$	Second Order Central
$\frac{\partial^2 U}{\partial x^2} = U_{xx}$	$\frac{U_{i+1}^n - 2U_i^n - U_{i-1}^n}{\Delta x^2}$	Second Order Symmetric
$\frac{\partial U}{\partial t} = U_t$	$\frac{U_i^{n+1} - U_i^n}{\Delta t}$	First Order Forward
$\frac{\partial U}{\partial t} = U_t$	$\frac{U_i^n - U_i^{n-1}}{\Delta t}$	First Order Backward
$\frac{\partial U}{\partial t} = U_t$	$\frac{U_i^{n+1} - U_i^{n-1}}{2\Delta t}$	Second Order Central
$\frac{\partial^2 U}{\partial t^2} = U_{tt}$	$\frac{U_i^{n+1} - 2U_i^n - U_i^{n-1}}{\Delta t^2}$	Second Order Symmetric

Table 13.1 Finite difference approximations (FDAs) for partial derivatives derived from Taylor series expansions.

Numerical Methods: Heat Diffusion in a Rod: Program_13c.py $U_t - \alpha U_{xx} = 0$

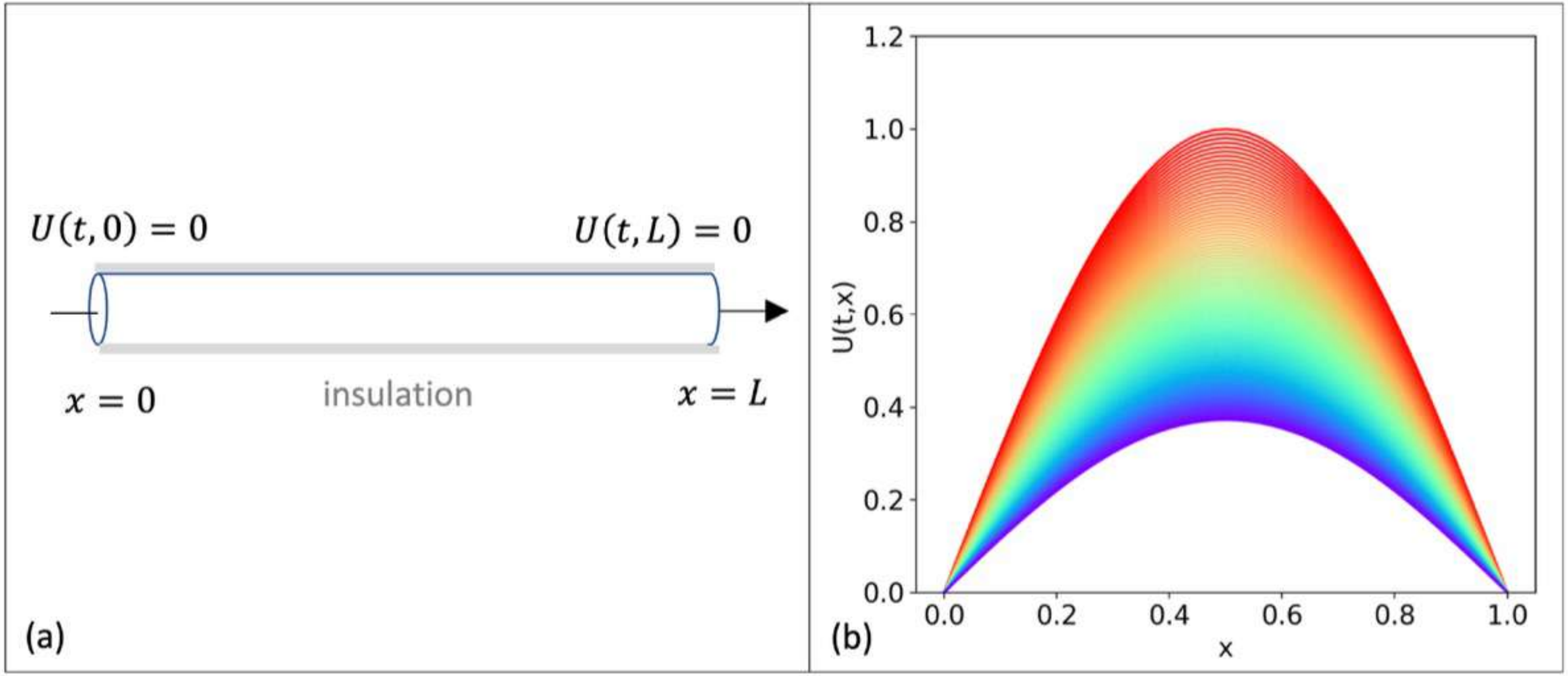


Figure 13.3 (a) Heat diffusion in an insulated rod, $U(t,0) = U(t,L) = 0$, $U(0,x) = \sin\left(\frac{\pi x}{L}\right)$, $\alpha = 0.1$, $L = 1$, for equation (13.9). (b) Numerical solution $U(t,x)$, for $0 \leq t \leq 1$, is stable as $\Delta t = 0.001$, and $\frac{\Delta x^2}{2\alpha} = 0.002$.

Manchester Metropolitan University

Numerical Methods: Vibrating String: Program_13d.py

$$U_{tt} - c^2 U_{xx} = 0$$

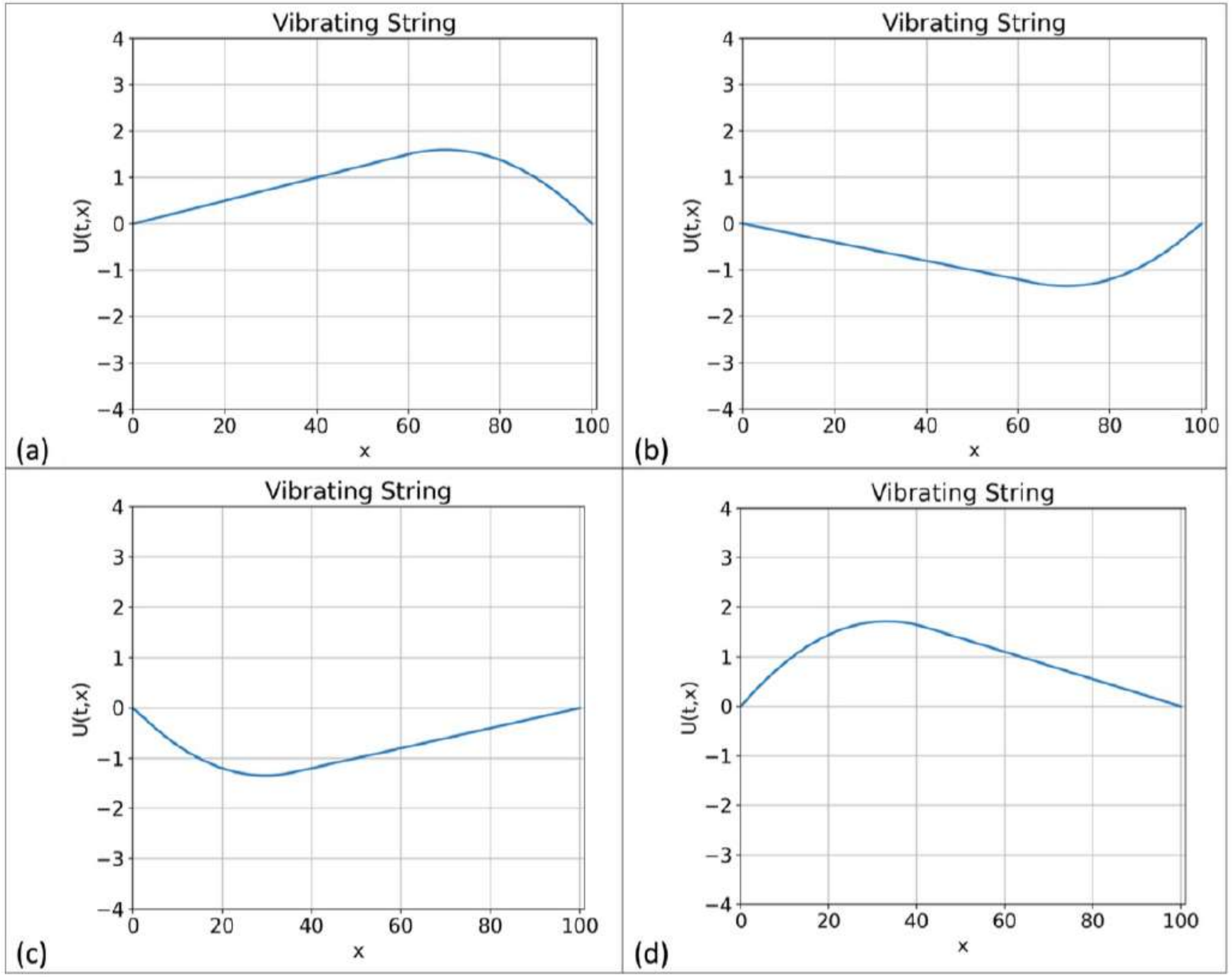


Figure 13.4 Vibration of a guitar string. Screenshots from the animation produced by running Program13d.py. The string vibrates in an oscillatory manner, (a)-(d), and back to (a).

Numerical Methods: Advection

$$U_t + vU_x = 0$$

Advection is the transfer of heat or matter by the flow of a fluid.

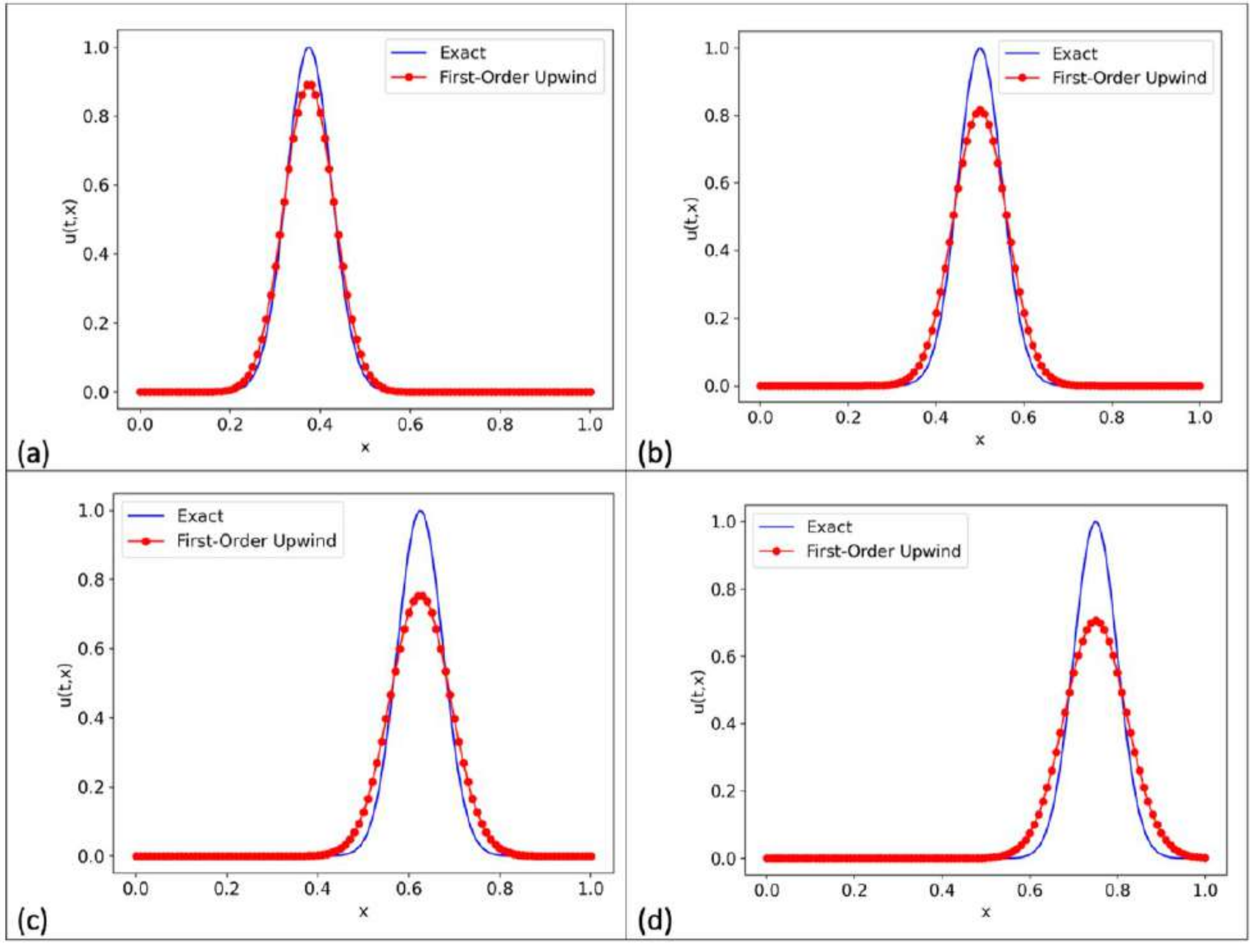


Figure 13.6 Solution of the advection equation (P13.2). (a) $t = 0.25s$; (b) $t = 0.5s$; (c) $t = 0.75s$; (d) $t = 1s$.

Manchester Metropolitan University

Numerical Methods: Heat in a Plate $U_t = \alpha\left(U_{xx} + U_{yy}\right)$ End Session 3

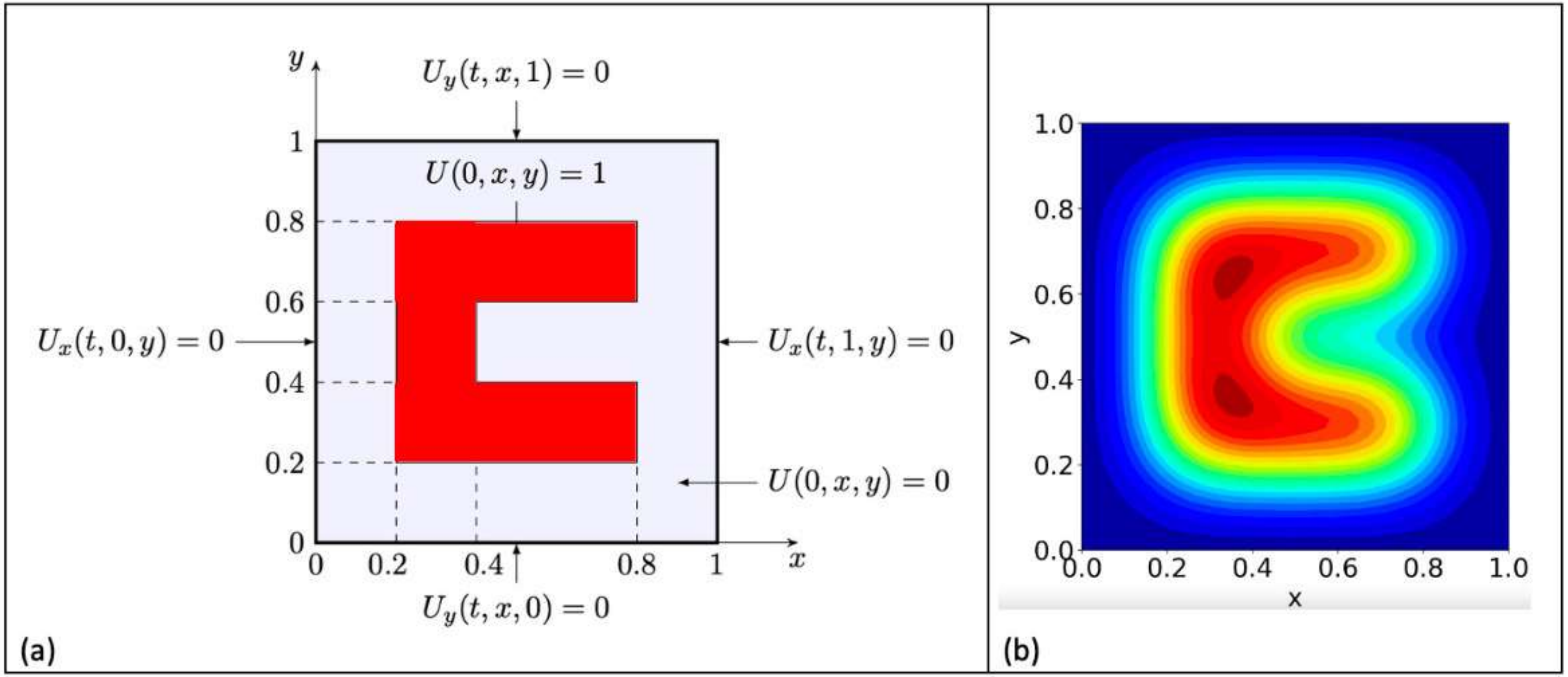
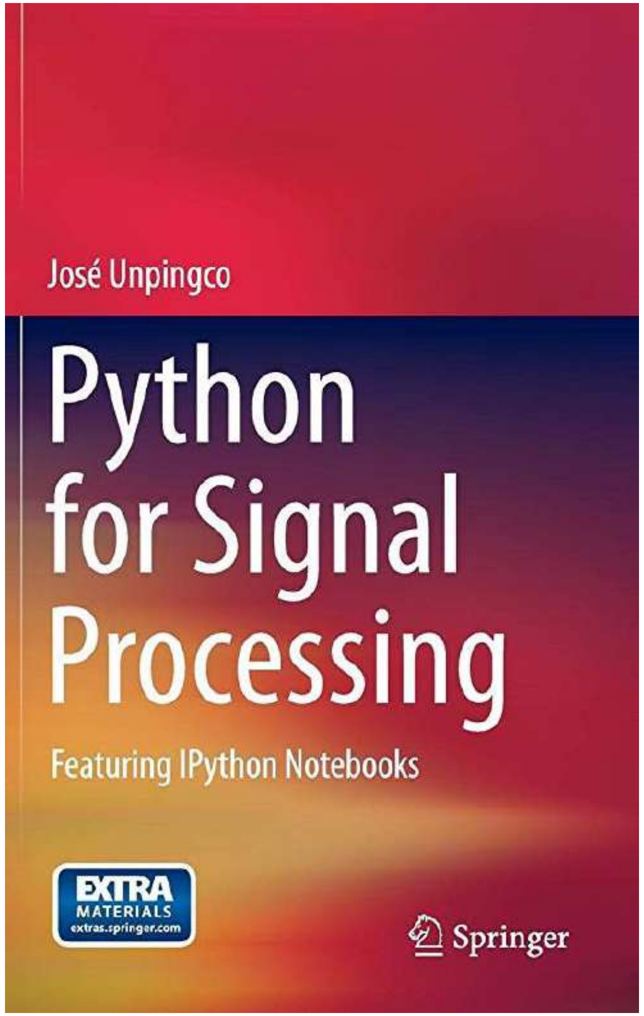


Figure 13.7 (a) Boundary conditions for heat diffusion across a metal sheet. (b) Numerical solution using Python.

Physics: Signal Processing: Start Session 4




```
# Program_14a.py: Fast Fourier transform of a noisy signal.
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft
Ns = 1000                          # Number of sampling points
Fs = 800                           # Sampling frequency
T = 1/Fs                           # Sample time
t = np.linspace(0, Ns*T, Ns)
amp1, amp2 , amp3 = 0.5 , 1 , 2
freq1, freq2 , freq3 = 60 , 120 , 30
# Sum a 30Hz, 60Hz and 120 Hz sinusoid
x = amp1 * np.sin(2*np.pi * freq1*t) + amp2*np.sin(2*np.pi * freq2*t) \
    +amp3 * np.sin(2*np.pi * freq3*t)
NS = x + 0.5*np.random.randn(Ns)          # Add noise.
fig1 = plt.figure()
plt.plot(t, NS)
plt.xlabel("Time (ms)", fontsize=15)
plt.ylabel("NS(t)", fontsize=15)
plt.tick_params(labelsize=15)
fig2 = plt.figure()
Sf = fft(NS)
xf = np.linspace(0, 1/(2*T), Ns//2)
plt.plot(xf, 2/Ns * np.abs(Sf[0:Ns//2]))
plt.xlabel("Frequency (Hz)", fontsize=15)
plt.ylabel("$|NS(f)|$", fontsize=15)
plt.tick_params(labelsize=15)
plt.show()
```

Physics: Fast Fourier Transform FFT

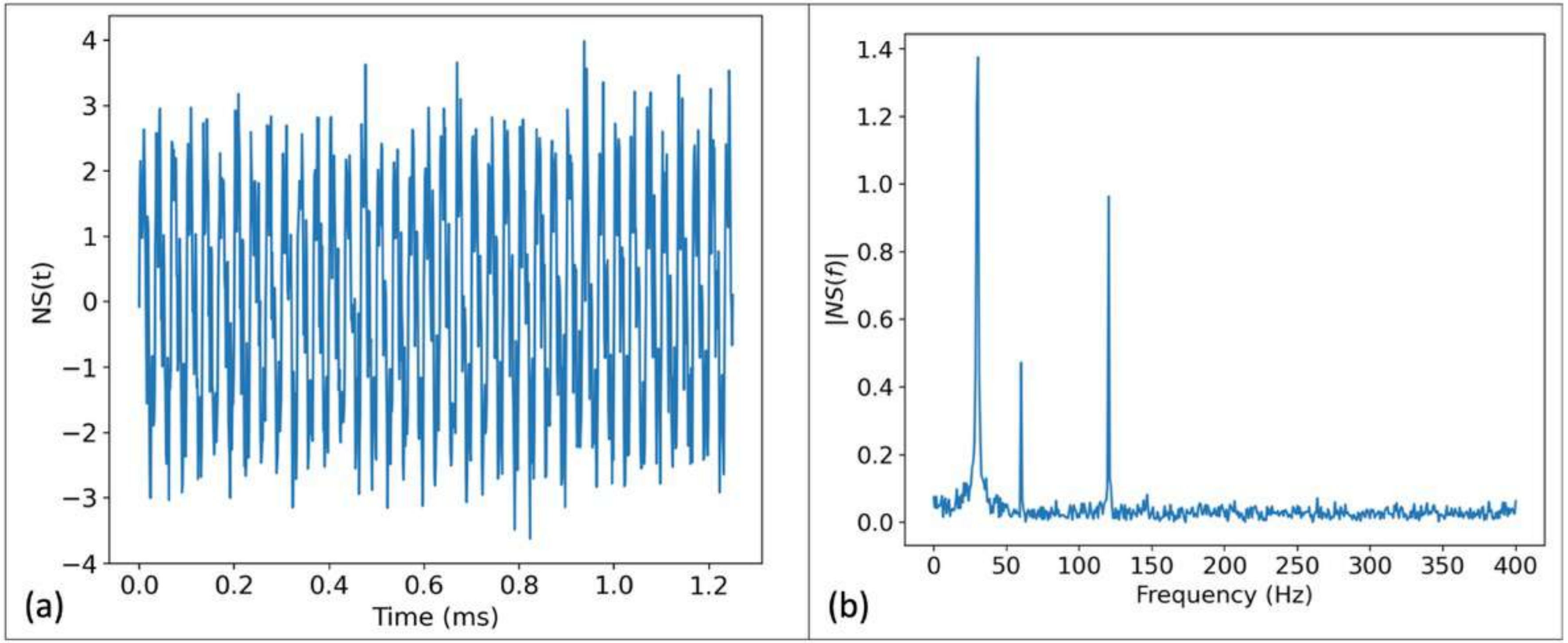


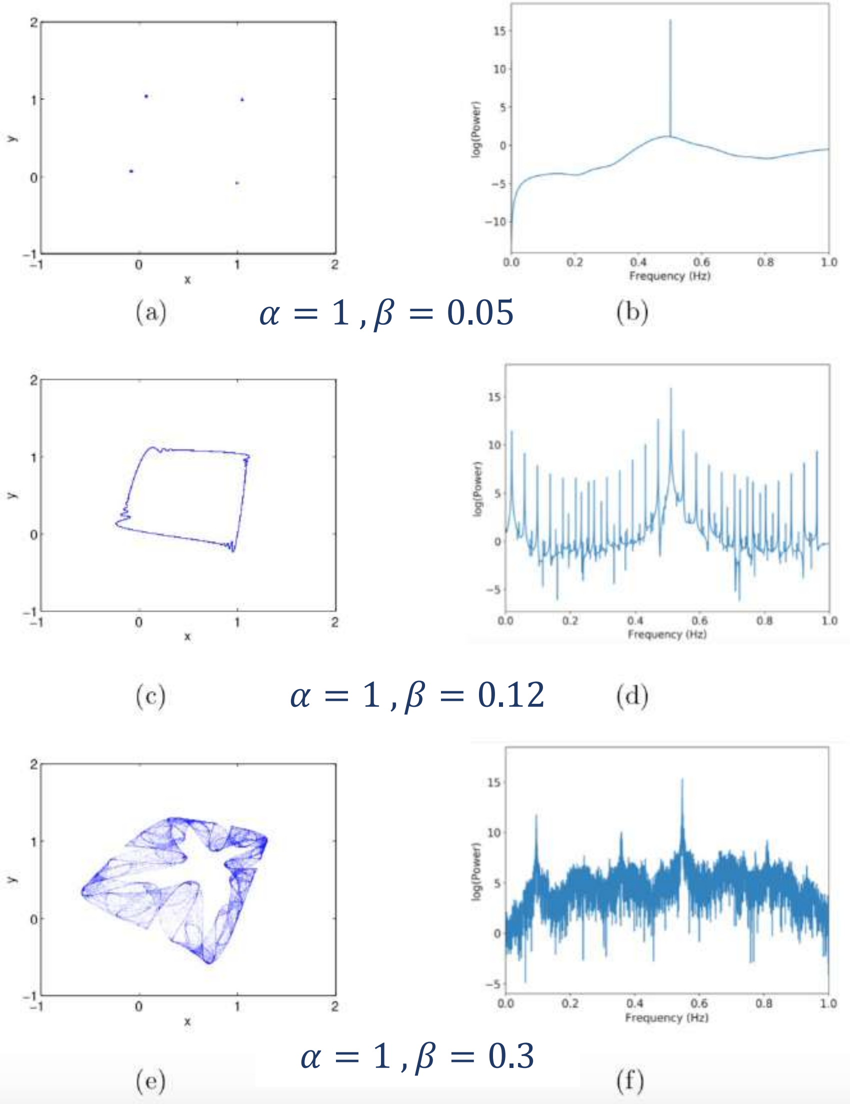
Figure 14.1 (a) Noisy signal, $NS(t) = noise + 0.5\sin(2\pi(60t)) + 1\sin(2\pi(120t)) + 2\sin(2\pi(30t))$. (b) The amplitude spectrum $|NS(f)|$. You can read off the amplitudes and frequencies. Increase the number of sampling points, Ns, in the program to increase accuracy.

Image Processing: FFT as a Chaos Detector

14.1 The power spectrum is given as $|\text{fft}(X)|^2$, where X is a vector of length n. Consider the 2-dimensional discrete map defined by

$$x_{n+1} = 1 + \beta x_n - \alpha y_n^2$$
$$y_{n+1} = x_n, \qquad \text{(P14.1)}$$

where α and β are constants. Suppose that $\alpha = 1$, plot iterative plots, and plots of log(power) against frequency for system (P14.1) when (i) $\beta = 0.05$ (periodic); (ii) $\beta = 0.12$ (quasi-periodic), and (iii) $\beta = 0.3$ (chaotic). In this case, the power spectra gives an indication as to whether or not the system is behaving chaotically.



(a) $\alpha = 1, \beta = 0.05$ (b)

(c) $\alpha = 1, \beta = 0.12$ (d)

(e) $\alpha = 1, \beta = 0.3$ (f)

Physics: Simple Fibre Ring Resonator: $E_{n+1} = A + BE_n e^{i|E_n|^2}$

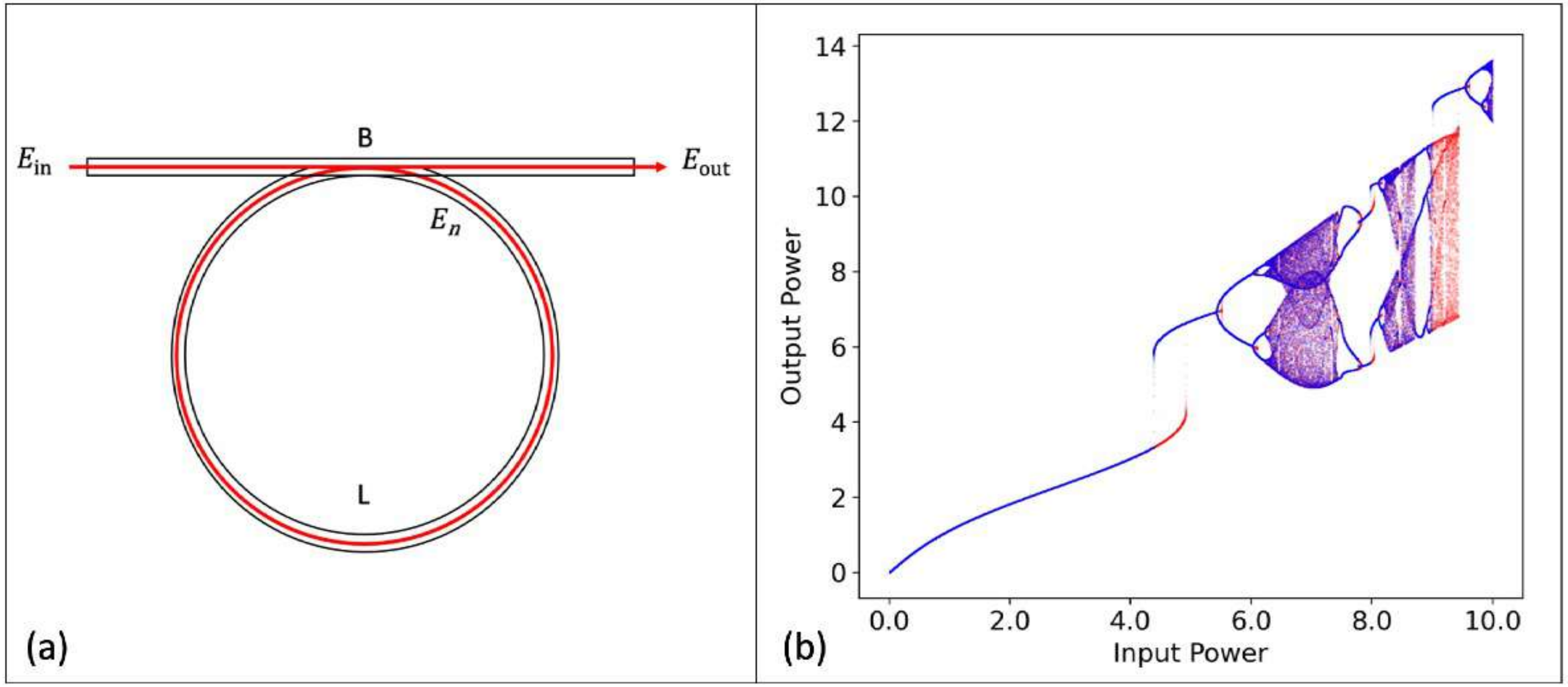


Figure 14.2 (a) The nonlinear simple fibre ring (SFR) resonator constructed with optical fibre. The red curves depict the laser in the optical fibre. (b) Bifurcation diagram of the Ikeda map (14.1) with feedback. There is a counterclockwise hysteresis loop and period-doubling and period-undoubling in and out of chaos. The red points indicate ramp up power, and the blue points represent ramp down power.

Physics: Complex Iteration

```
# Program_14b.py: Bifurcation diagram of the Ikeda map.
from matplotlib import pyplot as plt
import numpy as np
B , phi , Pmax , En = 0.15 , 0 , 10 , 0   # phi is a linear phase shift.
half_N = 99999
N = 2*half_N + 1
N1 = 1 + half_N
esqr_up, esqr_down = [], []
ns_up = np.arange(half_N)
ns_down = np.arange(N1, N)
# Ramp the power up
for n in ns_up:
    En = np.sqrt(n * Pmax / N1) + B * En * np.exp(1j*((abs(En))**2 - phi))
    esqr1 = abs(En)**2
    esqr_up.append([n, esqr1])
esqr_up = np.array(esqr_up)
# Ramp the power down
for n in ns_down:
    En = np.sqrt(2 * Pmax - n * Pmax / N1) + \
     B*En* np.exp(1j*((abs(En))**2 - phi))
    esqr1 = abs(En)**2
    esqr_down.append([N-n, esqr1])
esqr_down=np.array(esqr_down)
fig, ax = plt.subplots()
xtick_labels = np.linspace(0, Pmax, 6)
ax.set_xticks([x / Pmax * N1 for x in xtick_labels])
ax.set_xticklabels(["{:.1f}".format(xtick) for xtick in xtick_labels])
plt.plot(esqr_up[:, 0], esqr_up[:, 1], "r.", markersize=0.1)
plt.plot(esqr_down[:, 0], esqr_down[:, 1], "b.", markersize=0.1)
plt.xlabel("Input Power", fontsize=15)
plt.ylabel("Output Power", fontsize=15)
plt.tick_params(labelsize=15)
plt.show()
```

$$E_{n+1} = A + BE_n e^{i|E_n|^2}$$

$|A|^2$ is input power

$|E_n|^2$ is output power

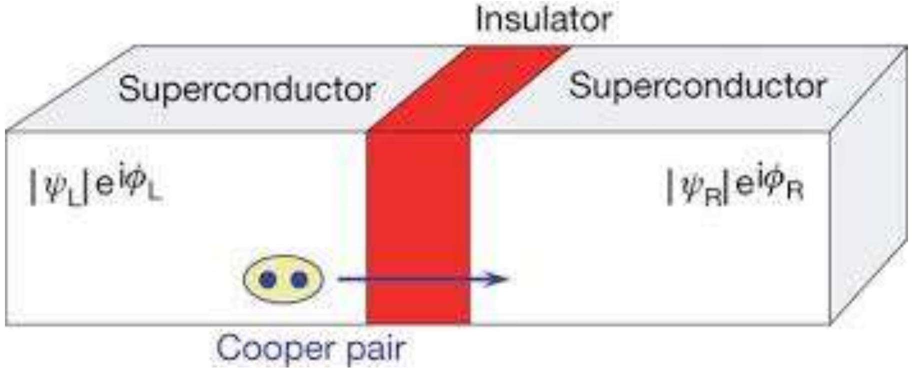
B is the fibre coupling ratio

Josephson Junctions

$$\frac{d\phi}{d\tau} = \Omega, \quad \frac{d\Omega}{d\tau} = \kappa - \beta_J \Omega - \sin \phi$$

In 1962, Brian David Josephson predicted the Josephson effect. He was the first to predict the tunneling of superconducting Cooper pairs.

Brian David Josephson



Josephson junctions are natural threshold super-cooled (4 Kelvin), superconducting oscillators that oscillate up to **100 million** times faster than neurons!

Manchester Metropolitan University

Physics: The Josephson Junction: Threshold Oscillator

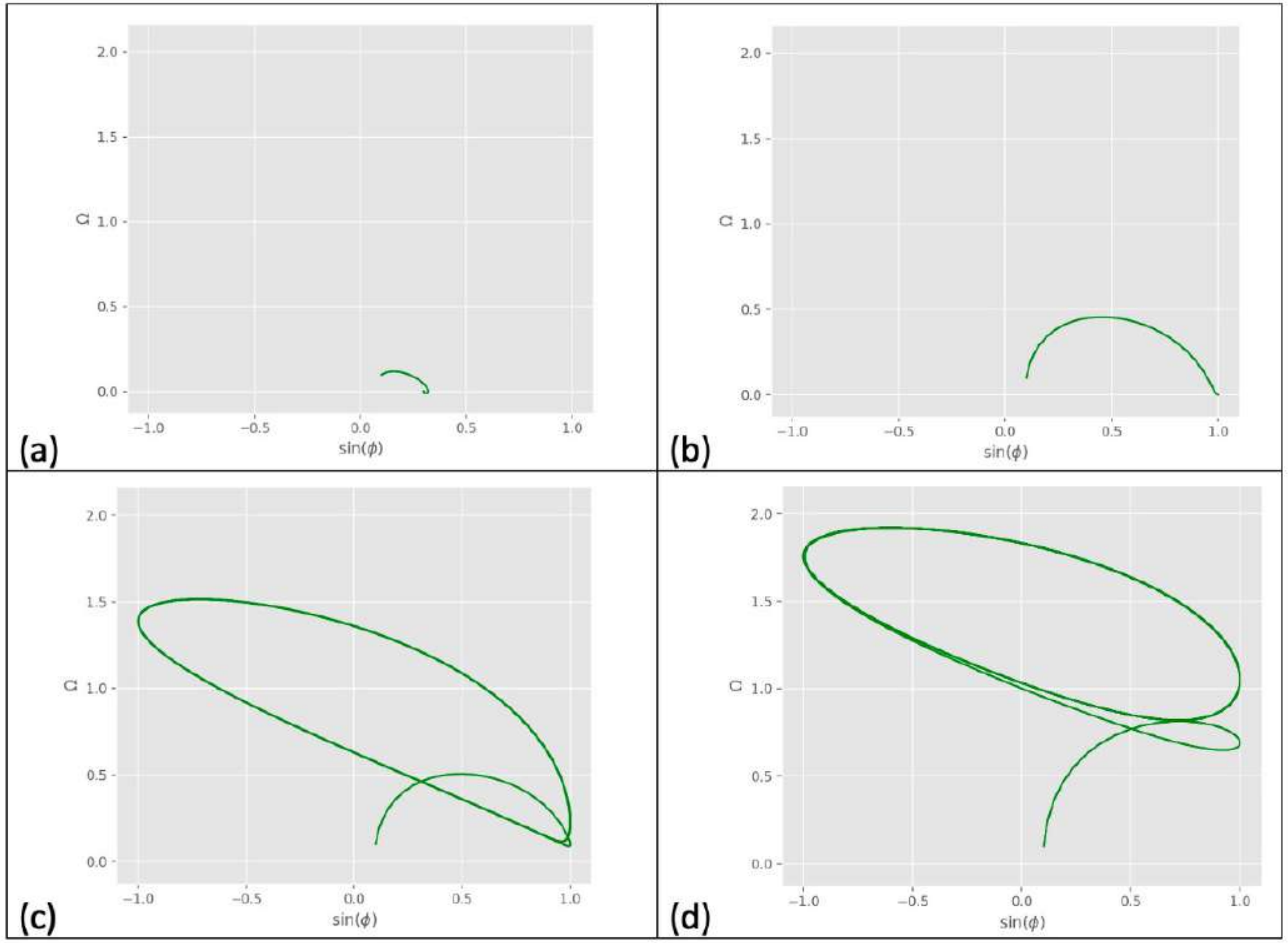


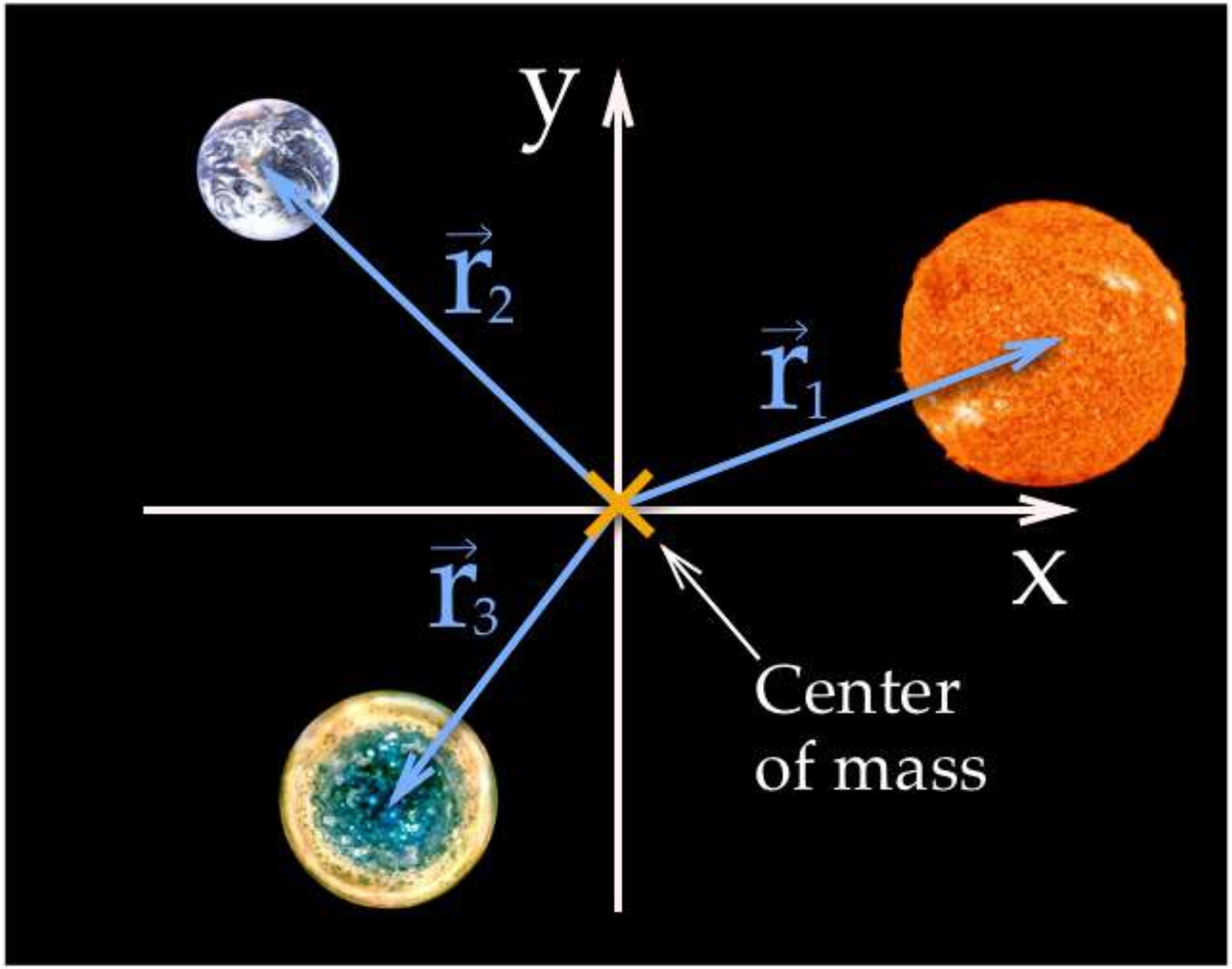
Figure 14.3 Animation of a resistively shunted JJ acting as a threshold oscillator. The current across the junction increases from $\kappa = 0.1$ to $\kappa = 2$. (a) No oscillation, for small κ. (b) Close to threshold. (c) Bifurcation of a limit cycle at $\kappa \approx 1.0025$. (d) The limit cycle moves vertically upwards for $\kappa > 1.0025$.

Physics: The Josephson Junction: Threshold Oscillator

```
# Program_14c.py: Animation of a JJ limit cycle bifurcation.
from matplotlib import pyplot as plt
from matplotlib import animation
import numpy as np
from scipy.integrate import odeint
from matplotlib import style
fig=plt.figure()
myimages=[]
BJ=1.2;Tmax=100;
def JJ_ODE(x, t):
    return [x[1],kappa-BJ*x[1]-np.sin(x[0])]
style.use("ggplot")           # To give oscilloscope-like graph.
time = np.arange(0, Tmax, 0.1)
x0=[0.1,0.1]
for kappa in np.arange(0.1, 2, 0.1):
    xs = odeint(JJ_ODE, x0, time)
    imgplot = plt.plot(np.sin(xs[:,0]), xs[:,1], "g-")
    myimages.append(imgplot)
my_anim=animation.ArtistAnimation(fig,myimages,interval=500,\
                                  blit=False,repeat_delay=100)
plt.rcParams["font.size"] = "18"
plt.xlabel("$\sin(\phi)$")
plt.ylabel("$\Omega$")
plt.show()
```

Physics: Motion of Planetary Bodies: Program_14d.py

$$\frac{d\mathbf{r_1}}{dt} = \mathbf{v_1}, \quad \frac{d\mathbf{r_2}}{dt} = \mathbf{v_2}, \quad \frac{d\mathbf{r_3}}{dt} = \mathbf{v_3},$$
$$\frac{d\mathbf{v_1}}{dt} = -Gm_2\frac{\mathbf{r_1}-\mathbf{r_2}}{|\mathbf{r_1}-\mathbf{r_2}|^3} - Gm_3\frac{\mathbf{r_1}-\mathbf{r_3}}{|\mathbf{r_1}-\mathbf{r_3}|^3},$$
$$\frac{d\mathbf{v_2}}{dt} = -Gm_3\frac{\mathbf{r_2}-\mathbf{r_3}}{|\mathbf{r_2}-\mathbf{r_3}|^3} - Gm_1\frac{\mathbf{r_2}-\mathbf{r_1}}{|\mathbf{r_2}-\mathbf{r_1}|^3},$$
$$\frac{d\mathbf{v_3}}{dt} = -Gm_1\frac{\mathbf{r_3}-\mathbf{r_1}}{|\mathbf{r_3}-\mathbf{r_1}|^3} - Gm_2\frac{\mathbf{r_3}-\mathbf{r_2}}{|\mathbf{r_3}-\mathbf{r_2}|^3},$$



Physics: Motion of Planetary Bodies: Program_14d.py

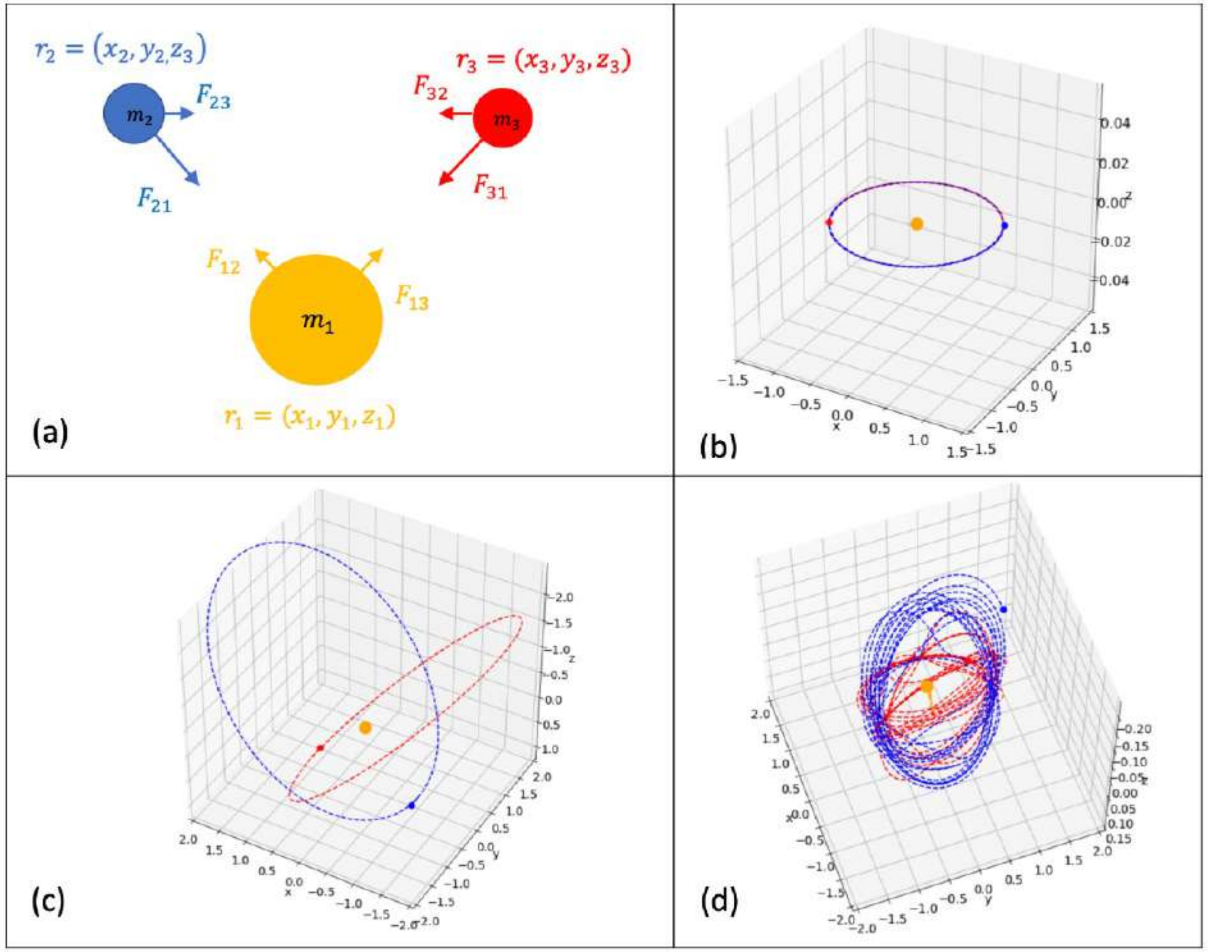


Figure 14.4 The three-body problem. (a) The positions and forces on the three bodies. (b) Circular motion for equations (14.4) under conditions (i). The red and blue bodies move on circular orbits about the heavier orange body in a plane. (c) Elliptic motions for equations (14.4) under conditions (ii). (d) What appears to be more random behaviour under conditions (iii).

Statistics: Linear Regression: Program_15a.py

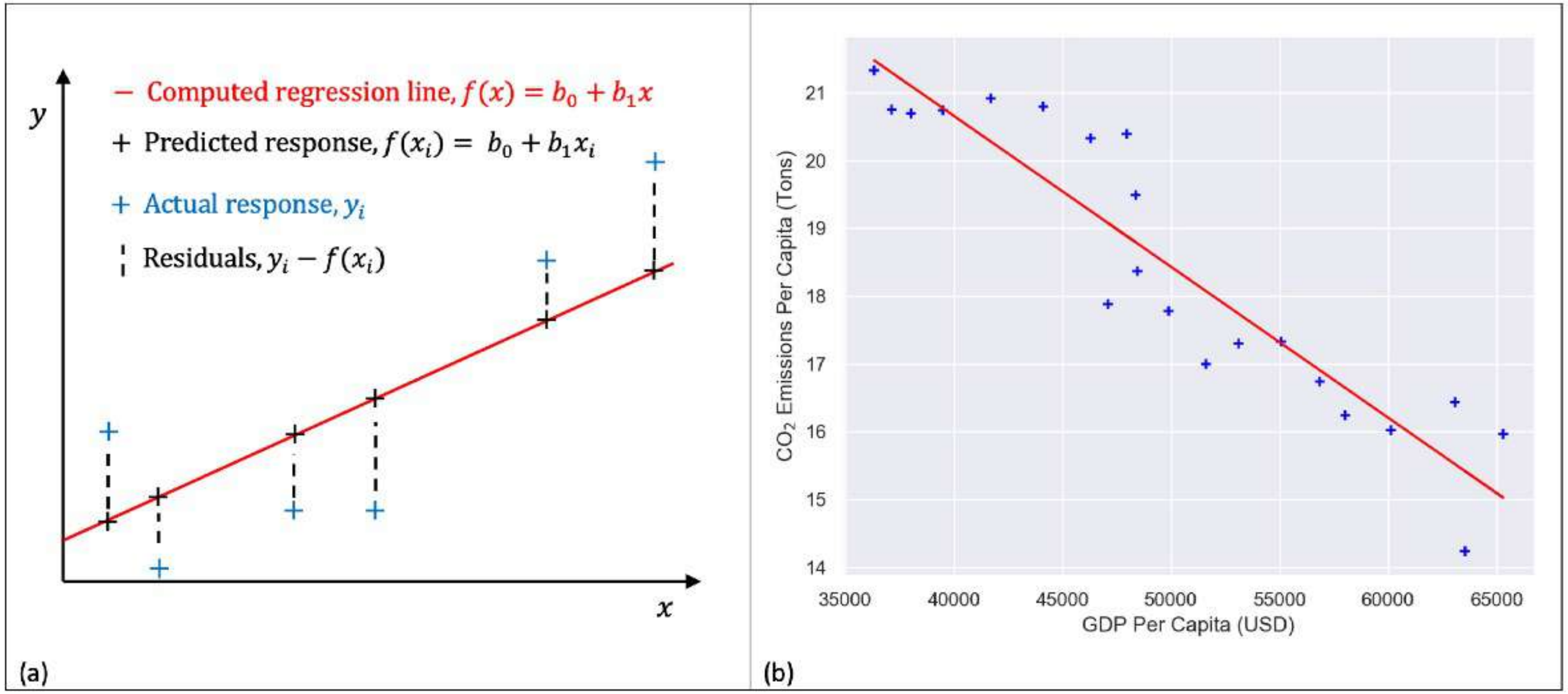


Figure 15.1 (a) Simple linear regression. (b) Simple linear regression for carbon dioxide emissions per capita, and gross domestic product per capita, for USA, in the years 2000 to 2020. In this case, the gradient $b_1 = -0.00022269$, the y-intercept is 29.567819, the mean squared error is 0.6 and the R^2 score is 0.86.

Manchester Metropolitan University

Statistics:

```
#Program_15a.py: Simple Linear Regression.
import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd
import statsmodels.api as sm
import seaborn as sns
sns.set()
data=pd.read_csv("CO2_GDP_USA_Data.csv")
data.head()
plt.rcParams["font.size"] = "20"
y = np.array(data["co2 per capita (metric tons)"])
x = np.array(data["gdp per capita (USD)"]).reshape((-1, 1))
plt.scatter(x , y , marker = "+" , color = "blue")
plt.ylabel("CO$_2$ Emissions Per Capita (Tons)")
plt.xlabel("GDP Per Capita (USD)")
regr = linear_model.LinearRegression()
regr.fit(x , y)
y_pred = regr.predict(x)
print("Gradient: \n", regr.coef_)
print("y-Intercept: \n", regr.intercept_)
print("MSE: %.2f"% mean_squared_error(y , y_pred))
print("R2 Score: %.2f" % r2_score(y , y_pred))
plt.plot(x , y_pred , color = "red")
plt.show()
sm.add_constant(x)
results = sm.OLS(y , x).fit()
print(results.summary())
```

```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:                      y   R-squared (uncentered):                   0.924
Model:                            OLS   Adj. R-squared (uncentered):              0.920
Method:                 Least Squares   F-statistic:                              243.7
Date:                Sun, 17 Apr 2022   Prob (F-statistic):                    1.15e-12
Time:                        08:05:32   Log-Likelihood:                         -64.030
No. Observations:                  21   AIC:                                      130.1
Df Residuals:                      20   BIC:                                      131.1
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.0004   2.25e-05     15.611      0.000       0.000       0.000
==============================================================================
Omnibus:                        3.054   Durbin-Watson:                   0.030
Prob(Omnibus):                  0.217   Jarque-Bera (JB):                1.289
Skew:                          -0.113   Prob(JB):                        0.525
Kurtosis:                       1.808   Cond. No.                         1.00
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Statistics: Markov Chains

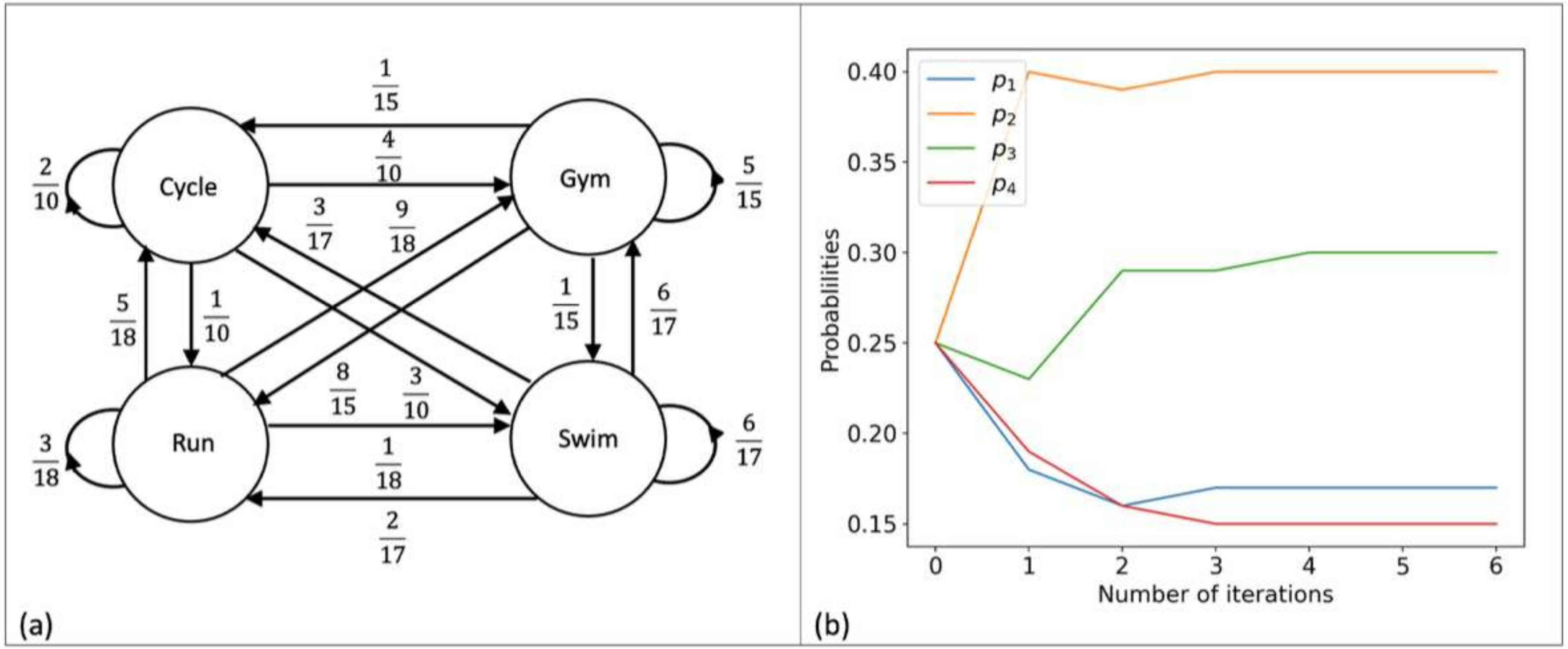


Figure 15.3 (a) Markov chain directed graph, where circles represent workout events and the directed edges are probability transitions. (b) Convergence of the probability vector to the steady state vector, $\pi = [0.17, 0.4, 0.3, 0.15]$, after five iterations, where p_i are probabilities.

Statistics: Markov Chains

```
# Program_15b.py: Markov Chain.
import numpy as np
import matplotlib.pyplot as plt
T = np.array([[2/10,4/10,1/10,3/10], \
              [1/15,5/15,8/15,1/15], \
              [5/18,9/18,3/18,1/18], \
              [3/17,6/17,2/17,6/17]
              ])
n = 20
v=np.array([[0.25, 0.25, 0.25 , 0.25]])
print(v)
vHist = v
for x in range(n):
  v = np.dot(v , T).round(2)
  vHist = np.append(vHist , v , axis = 0)
  if np.array_equal(vHist[x] , vHist[x-1]):
      print("Steady state after" , x , "iterations.")
      break
  print(v)
plt.rcParams["font.size"] = "14"
plt.xlabel("Number of iterations")
plt.ylabel("Probablilities")
plt.plot(vHist)
#plt.rcParams["linecolor"]
plt.legend(["$p_1$","$p_2$","$p_3$","$p_4$"],loc="best")
plt.show()
```

Statistics: The Student T-Test: Program_15d.py

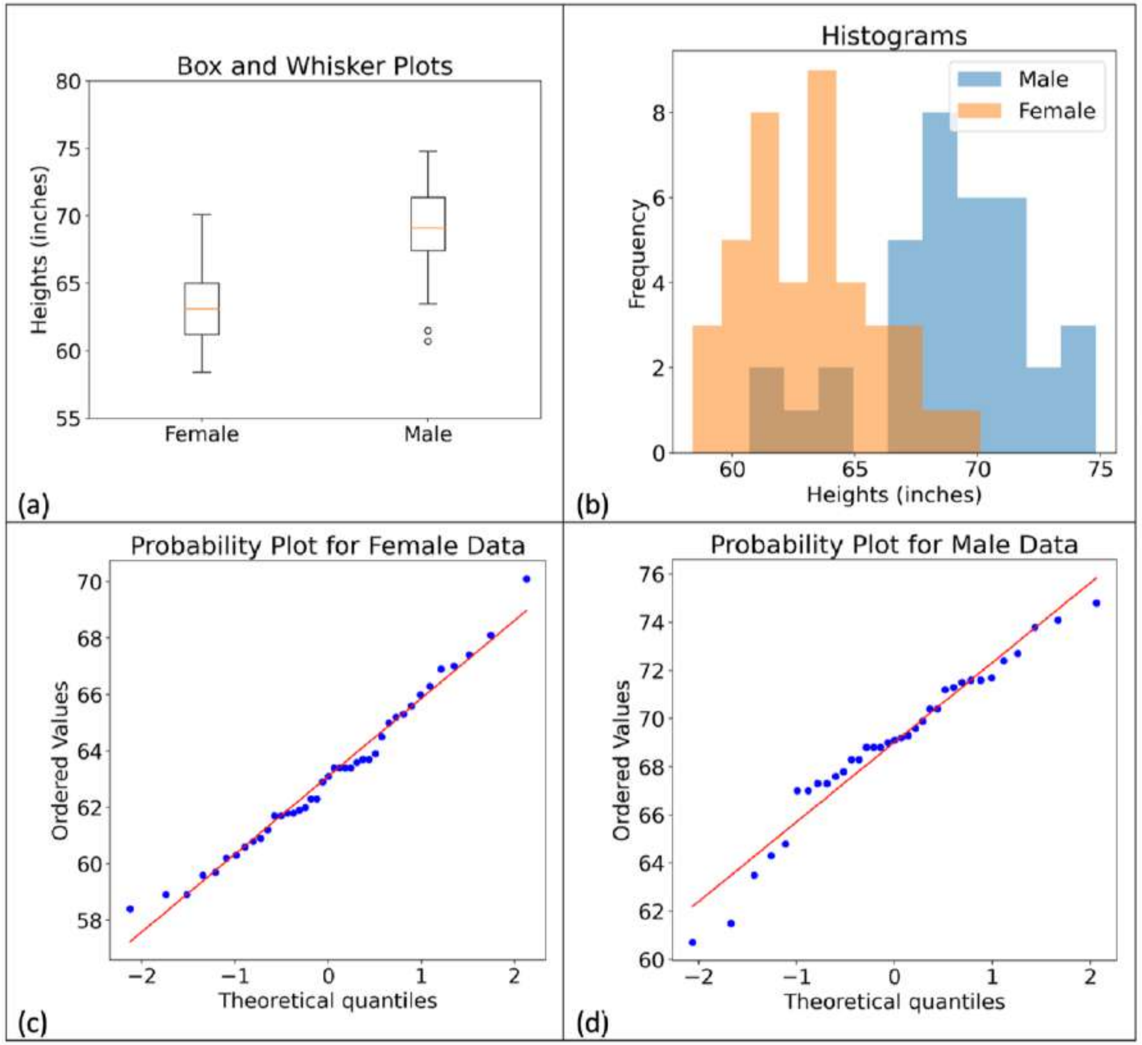


Figure 15.5 (a) Box and whisker plots of the data, notice the two outliers (circles) in the male data. (b) Histograms of the data. (c) and (d) Quantile-quantile (or Q-Q) plots, both distributions are nearly normal.

Manchester Metropolitan University

Statistics: Monte-Carlo Simulation: Roulette: Program_15e.py: End Session 4

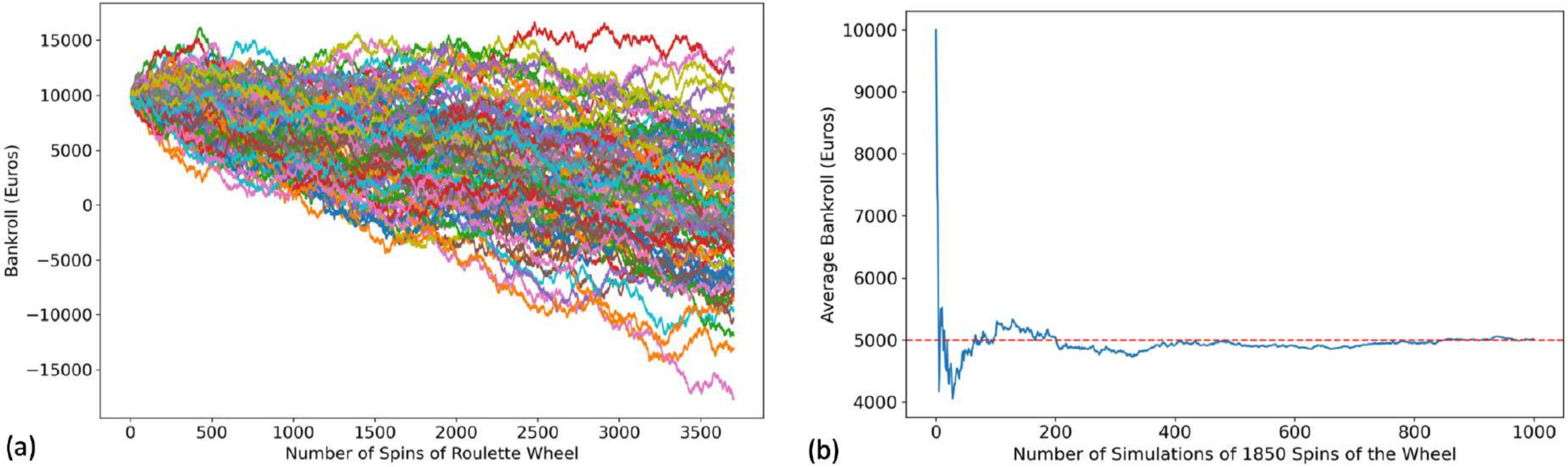
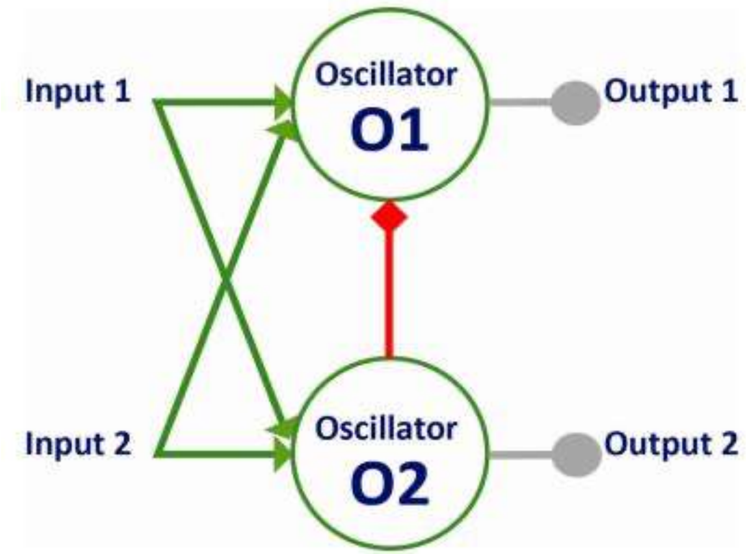


Fig. Monte Carlo simulations: (a) The bankrolls of 100 gamblers each betting on 3700 spins, where the gambler wages 100 euros on each spin. In this case, the program shows that on average, the gambler would leave the casino with a small bankroll, or even go into debt. Each time you run the simulation you will get different answers. (b) As the number of simulations of 1850 spins of the roulette wheel goes large, one can clearly see that the average bankroll tends to 5000 euros. So, on average, the gambler would expect to lose half of the bankroll after 1850 spins.

Binary Oscillator Computing

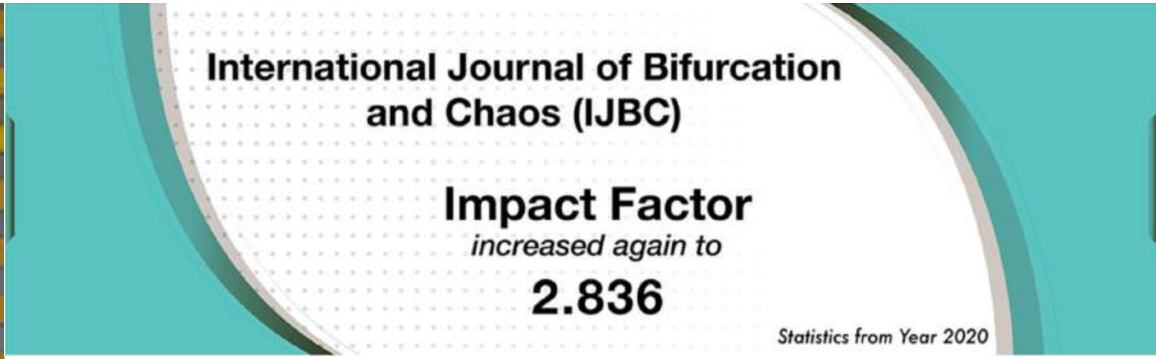


International patents.

The Inventors: Myself and Jon Borresen

In 2002, we had a journal paper published.

My co-inventor, Jon Borresen was my final year project student in 2001.

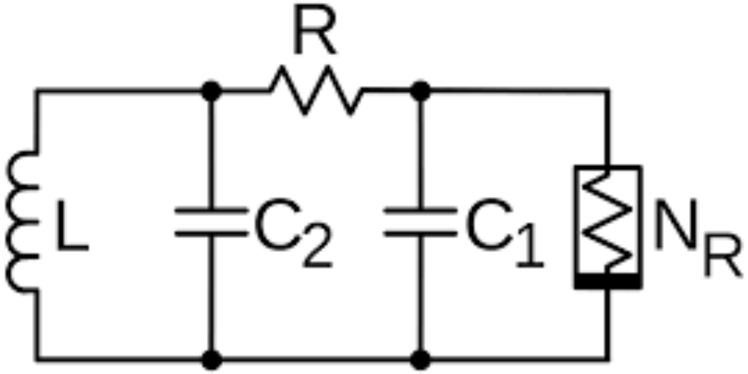


International Journal of Bifurcation and Chaos | Vol. 12, No. 01, pp. 129-134 (2002) | Letters

FURTHER INVESTIGATION OF HYSTERESIS IN CHUA'S CIRCUIT

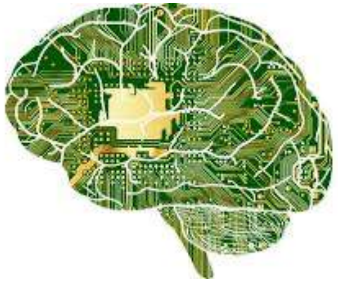
J. BORRESEN and S. LYNCH

https://doi.org/10.1142/S021812740200422X | **Cited by:** 18



AI and Binary Oscillator Computing

Artificial Intelligence



Artificial Intelligence, machine learning and deep learning.

Using computers to act like the human brain.

Brain Inspired Computing

Binary oscillator computing.

Using biological brain dynamics to create a powerful conventional supercomputer.

Outline

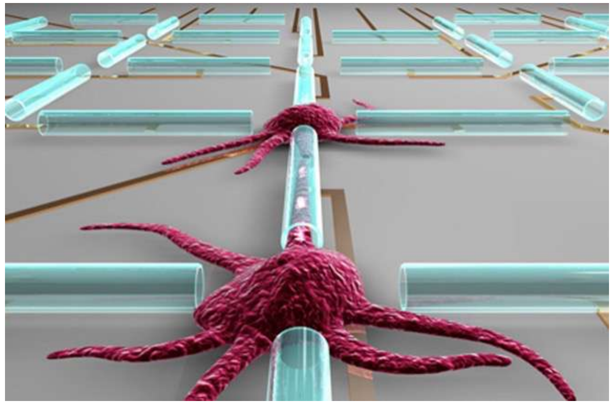
Neurons and the brain

Computing with threshold oscillators:

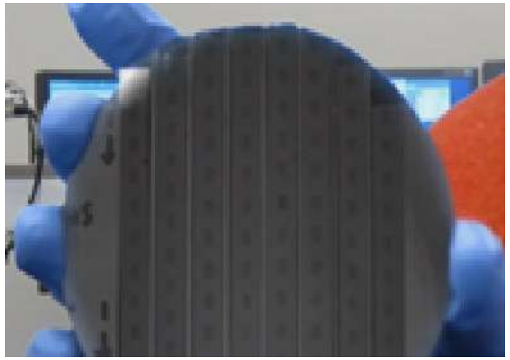
- Memristor oscillators
- Superconducting devices
- Biological neurons
- Transistor-based oscillators
- All-optical oscillators

Possible applications:

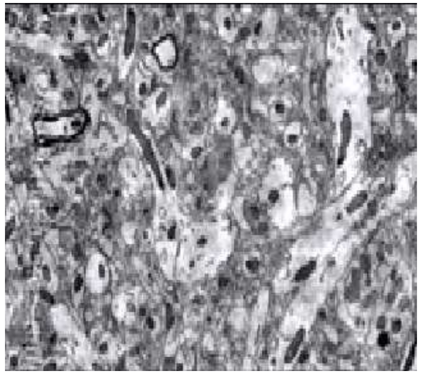
- Exascale (or beyond) supercomputing
- An assay for neuronal degradation



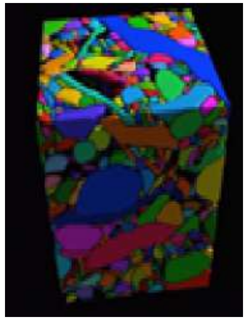
Neurons in the Mouse Brain



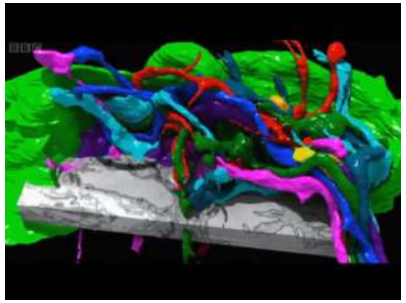
Slices of a mouse brain.



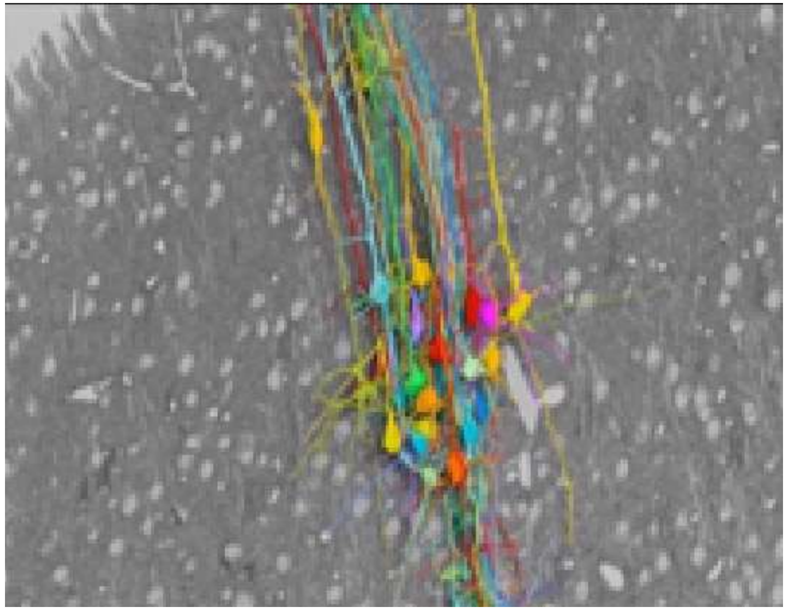
One slice of a mouse brain.



A cube of synaptic connections.



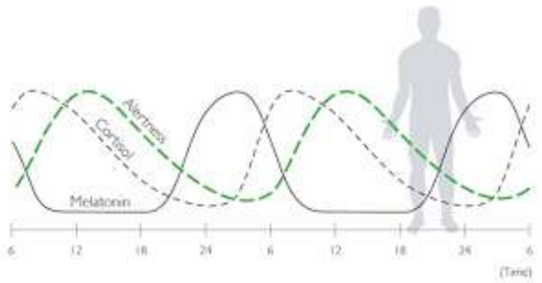
A few connections.

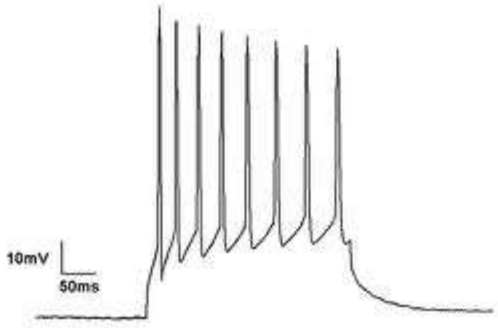


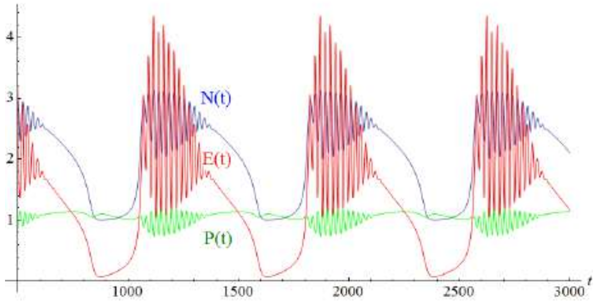
Mouse neurons.

Using current technology it would take 4 million years to produce a map like this for one human brain!
Research conducted by Jeff Lichtman and his group (Harvard University, USA).

Oscillators in the Human Body





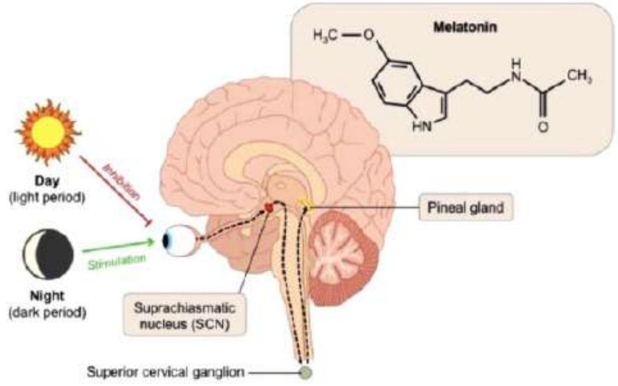


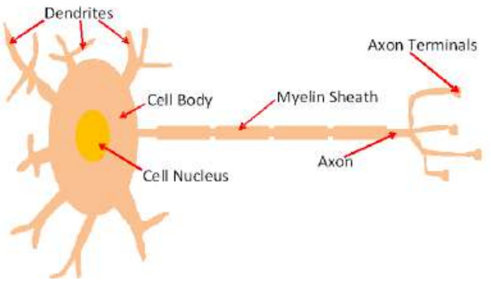
Circadian oscillations – wake/sleep.

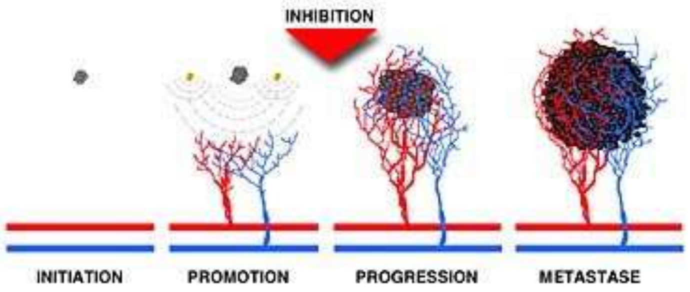
The human heart: an average 60-80 beats per minute.

A neuron spike train: beats up to a thousand times faster than the heart.

Angiogenesis: new blood vessels form from pre-existing vessels. N is tumour size, P is quantity of growth factors, E is vessel density.



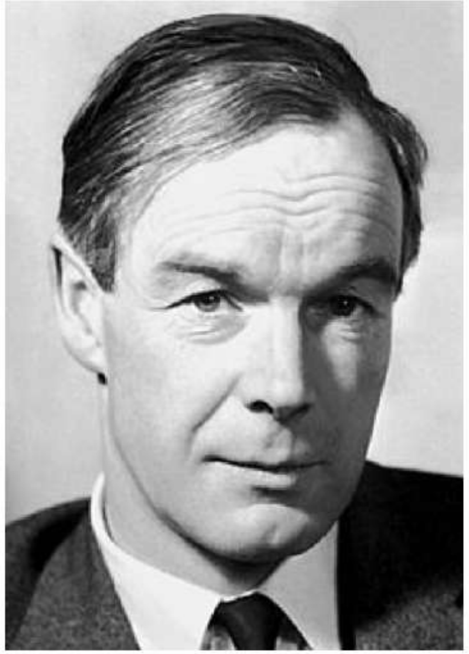




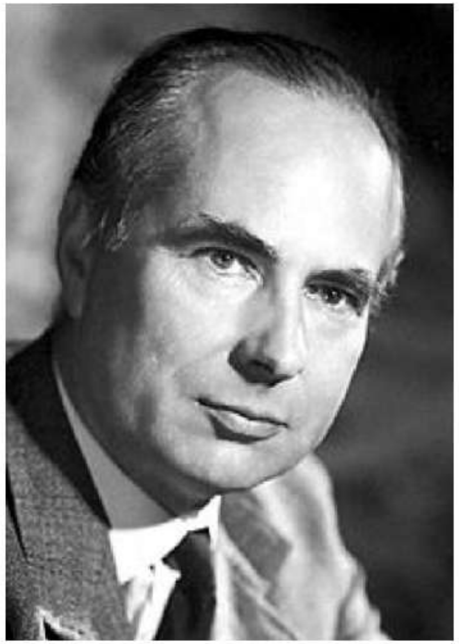
Manchester Metropolitan University

Mathematical Modelling of Neurons

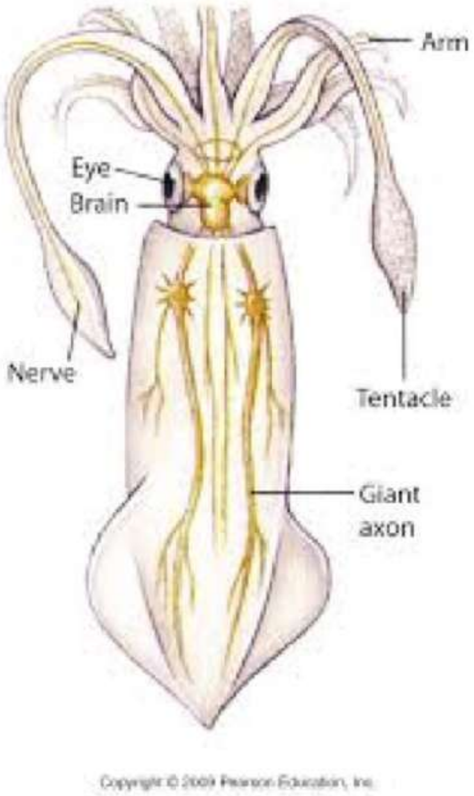
In 1952, Alan Lloyd Hodgkin and Andrew Huxley developed a mathematical model to describe how action potentials in neurons are initiated and propagated.



Sir Alan Lloyd Hodgkin

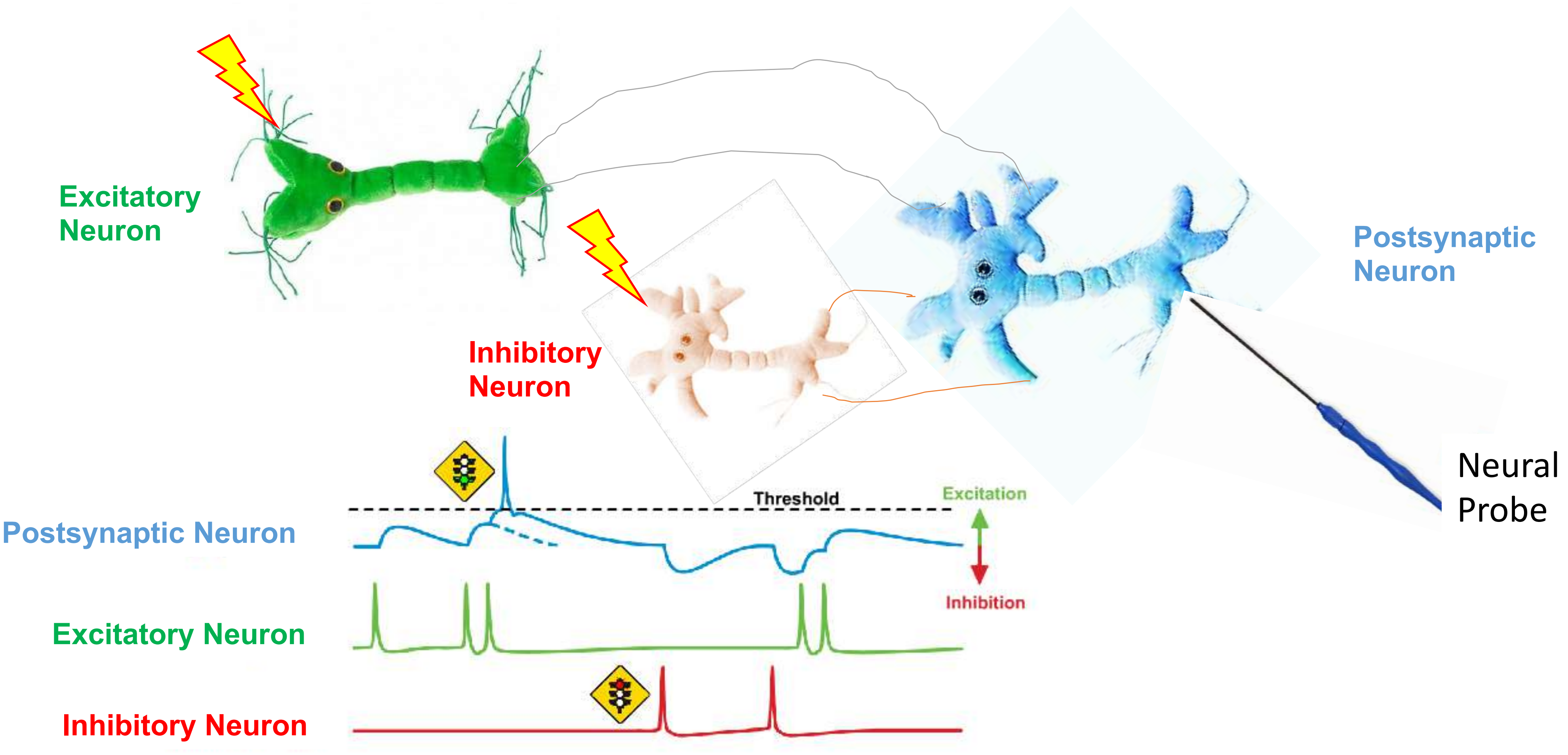


Sir Andrew Huxley



Hodgkin-Huxley studied the axon of a giant squid.

Neuron Model (Excitation and Inhibition)



Mathematical Modelling of Neurons

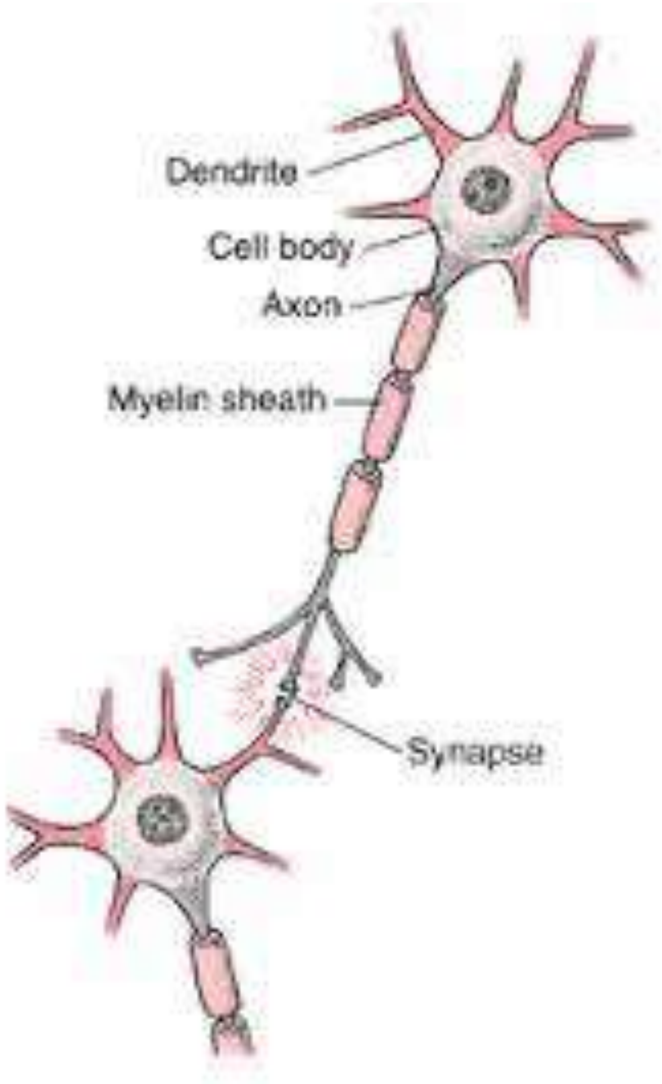
1952: The Hodgkin-Huxley Model
(Biophysically meaningful)

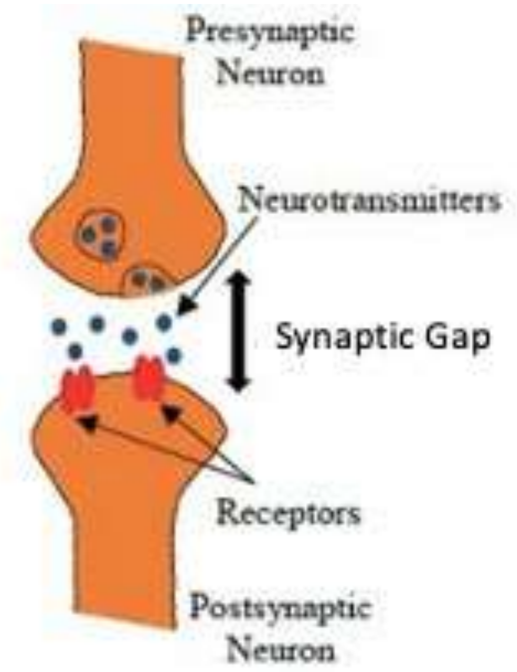
1961: Fitzhugh-Nagumo Models

1981: Morris-Lecar Model

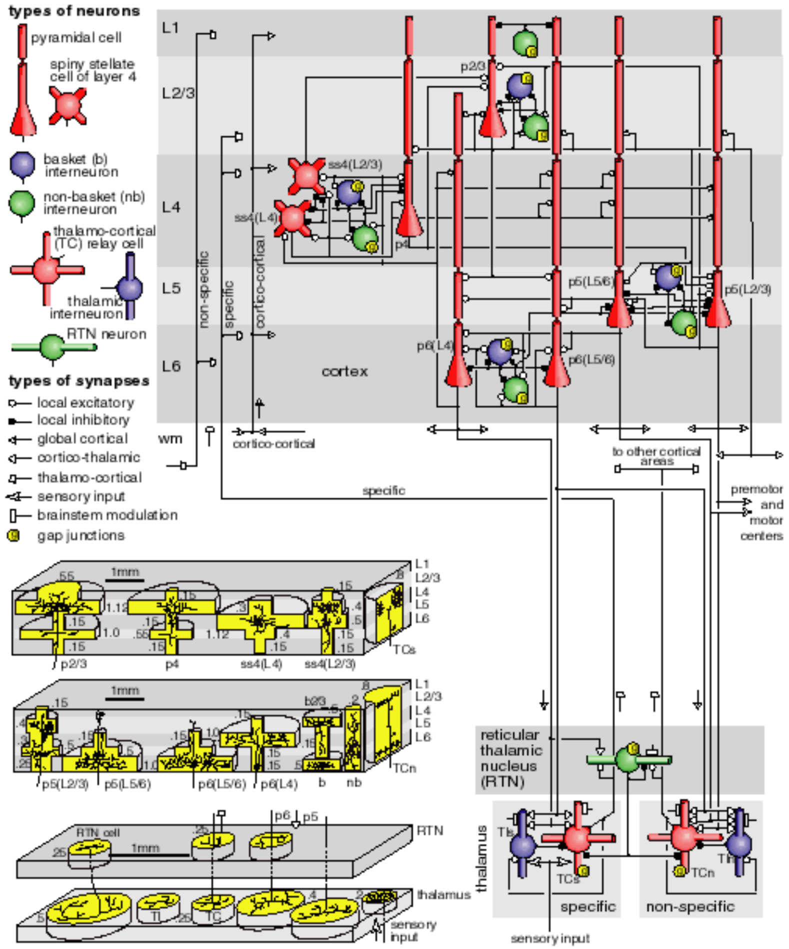
1984: Hindmarsh-Rose Model

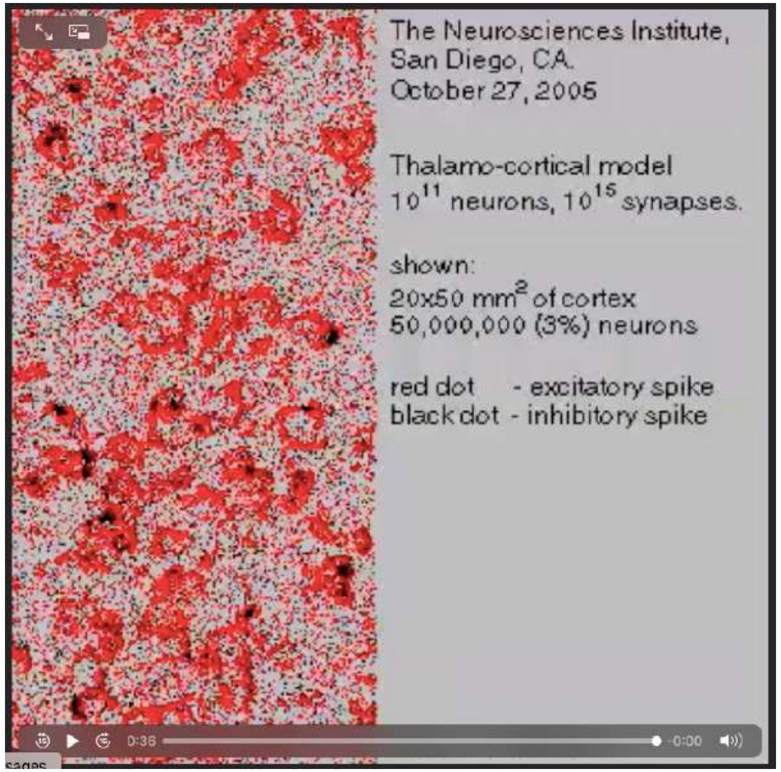
2005: Izhikevich Model
(Biophysically meaningless)





Izhikevich Mathematical Model of the Human Brain

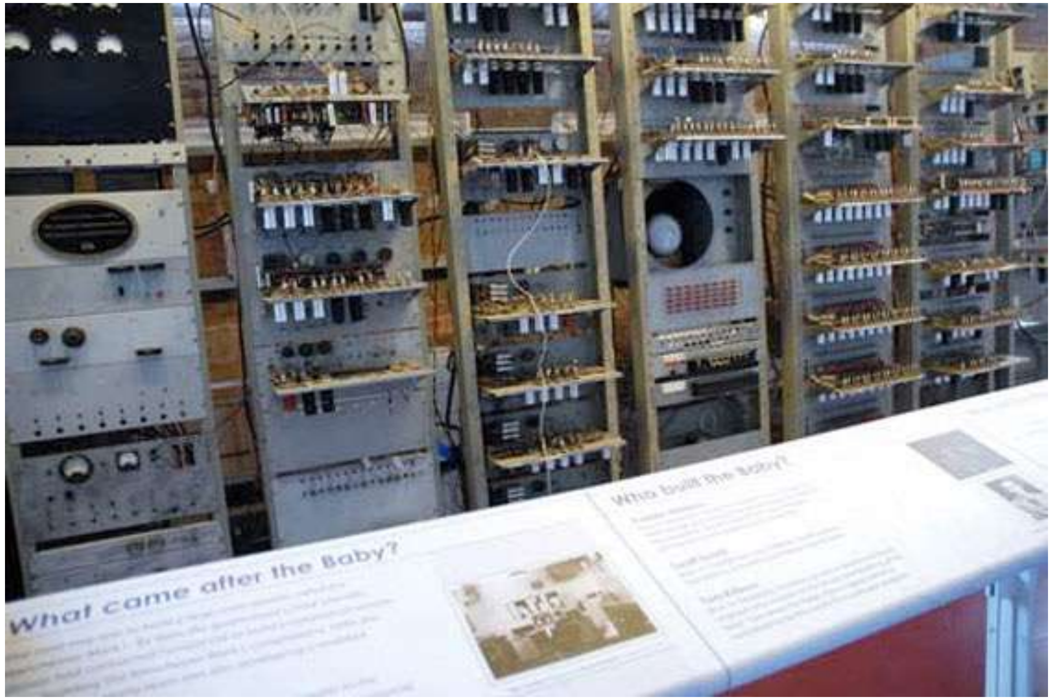




http://www.izhikevich.org/human_brain_simulation/Blue_Brain.htm

The Baby Computer

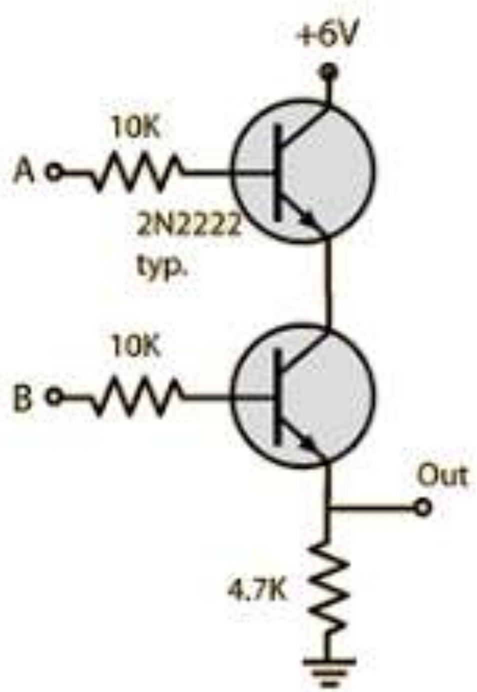
In 1948, the world's first program was run on Manchester University's small-scale experimental machine the "Baby". One of the principal components used was the vacuum tube oscillator. The Manchester Museum of Science and Industry (MOSI) built a working replica in 1998.



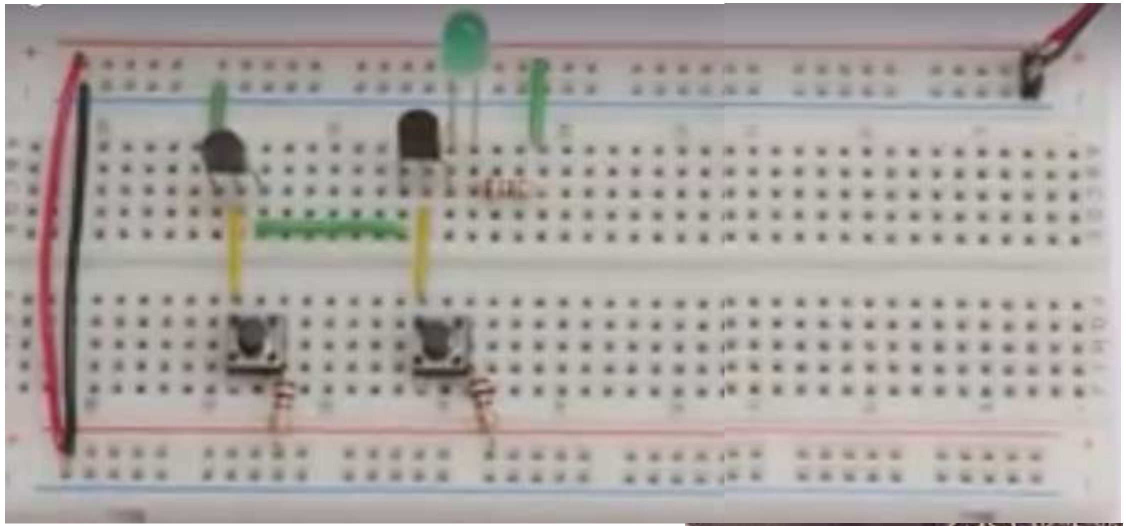
The Baby computer.

A vacuum tube.

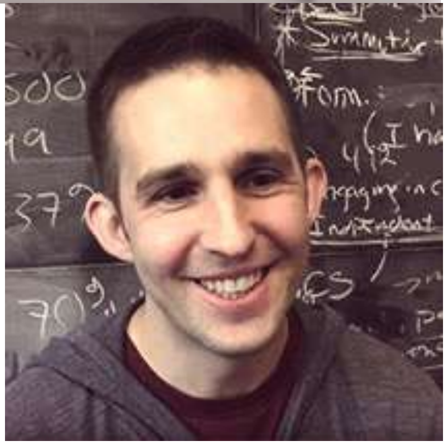
Ben Eater on YouTube: AND GATE



Two-transistor AND gate.

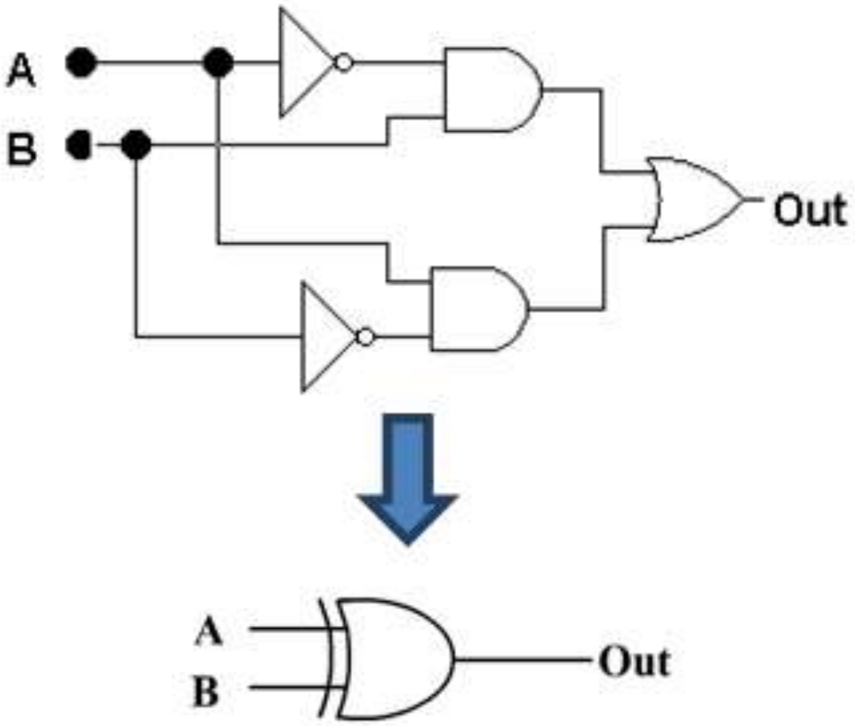


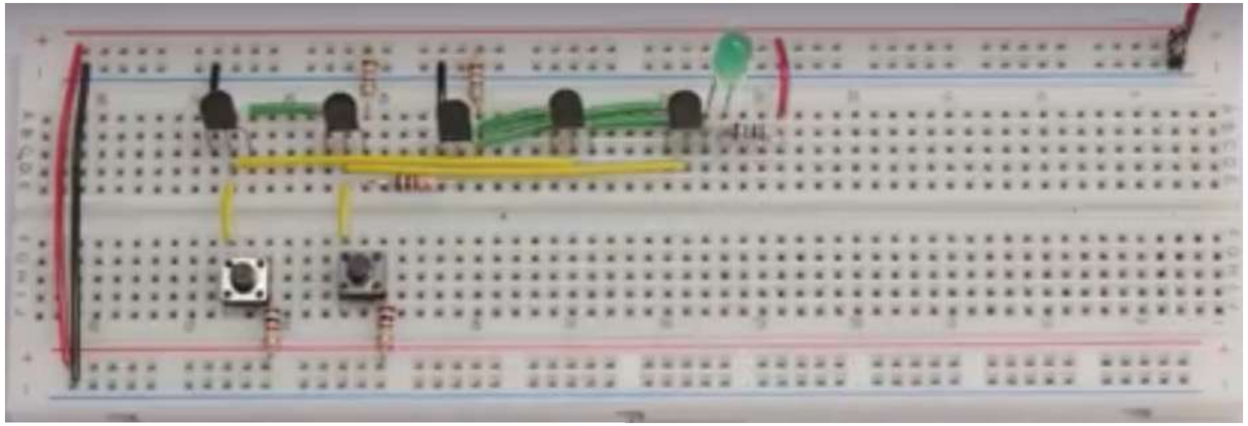
A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

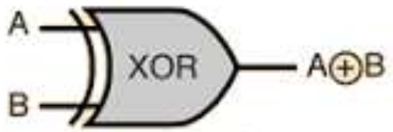


Electronics: XOR Gate with Transistors

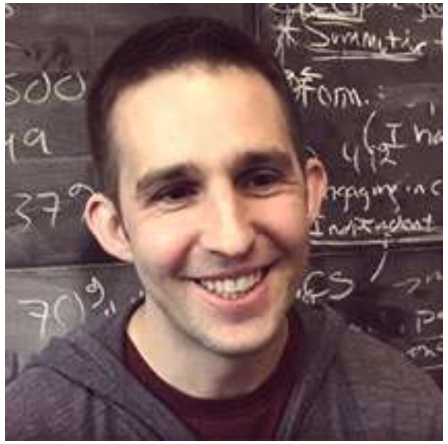
Ben Eater on YouTube: XOR GATE



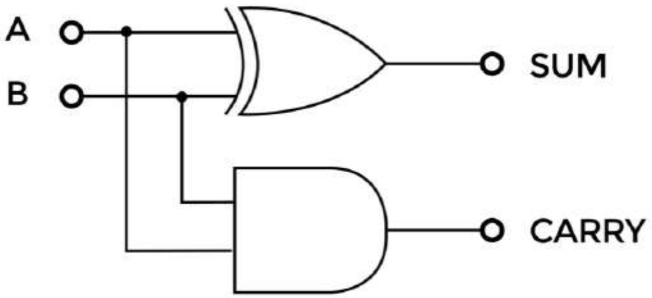




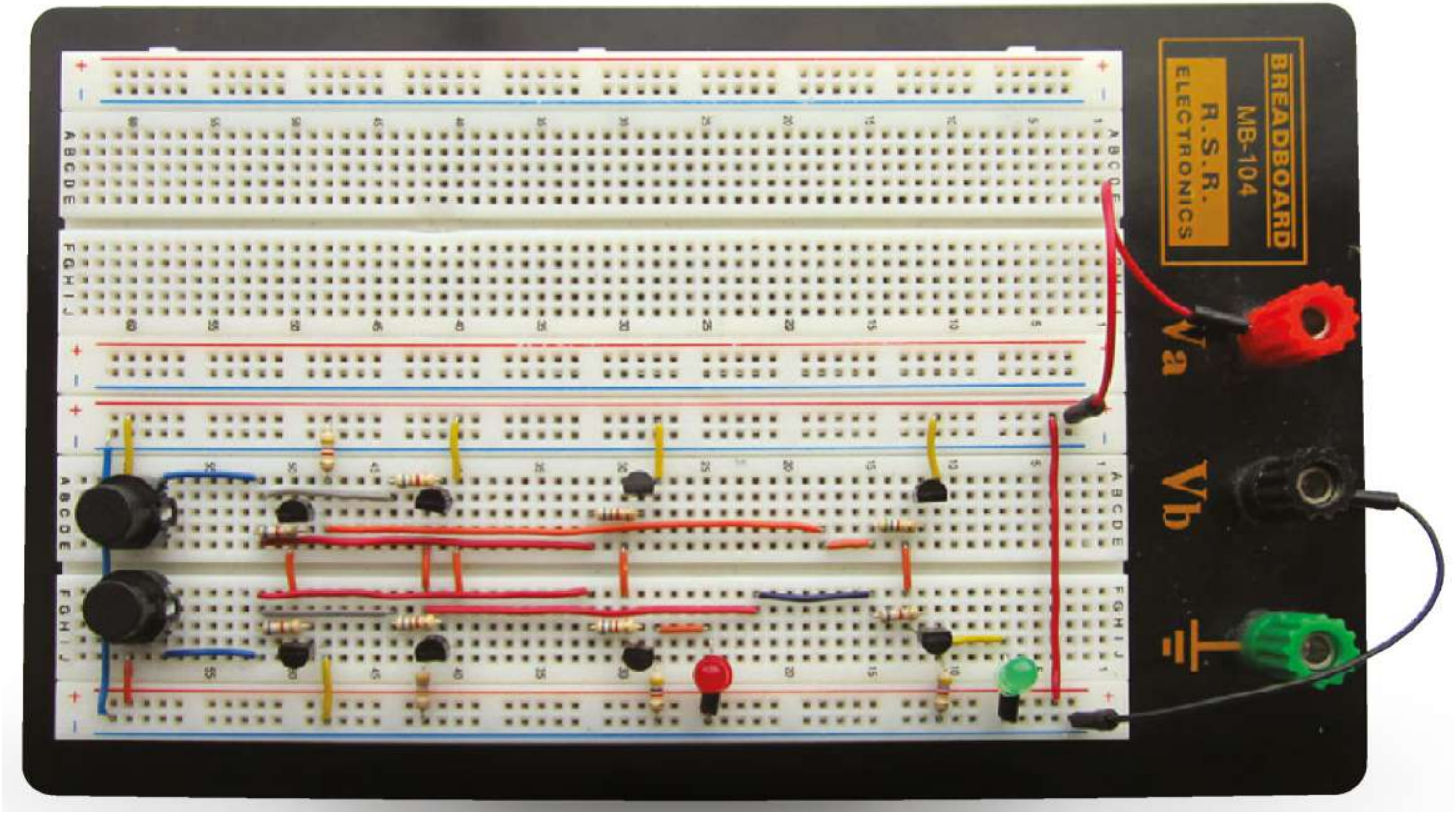
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0



Transistor-Based Half-Adder



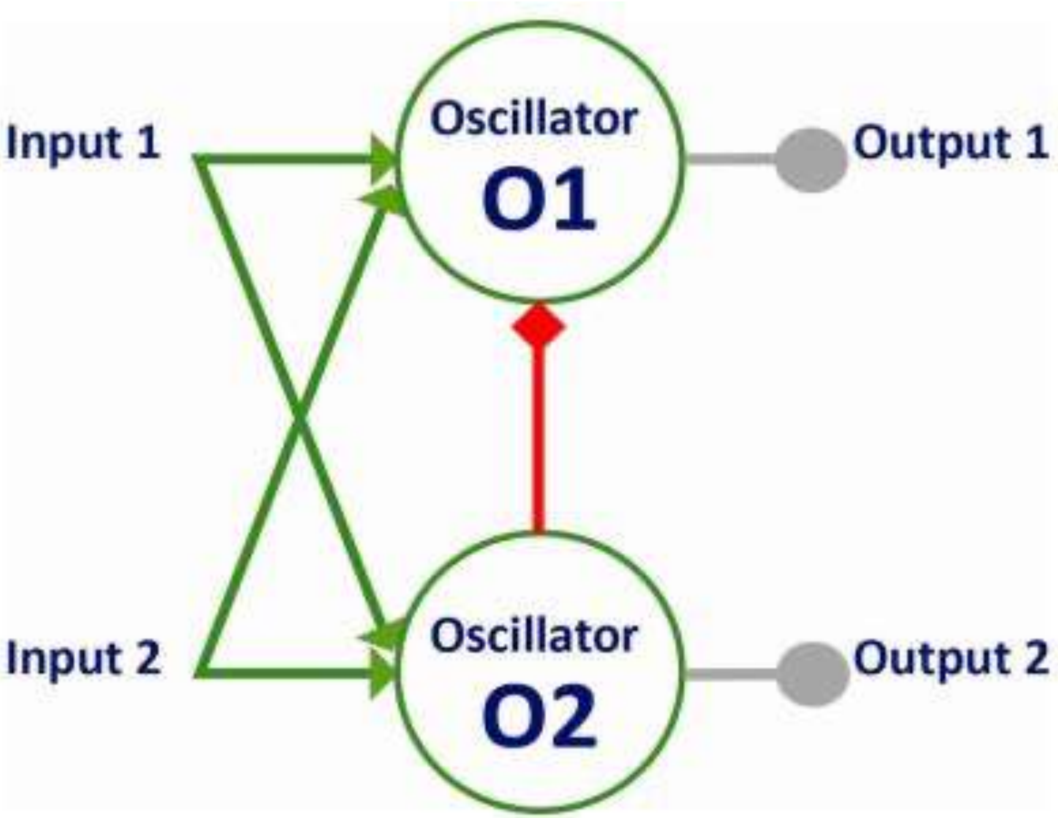
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



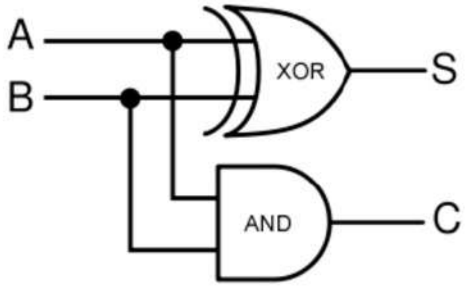
MMU Invention: Binary Oscillator Half Adder

Simple design:

Two oscillators
Four input connections
One inhibitory connection
One cross connection
Two outputs
Oscillator O1 has a low threshold
Oscillator O2 has a high threshold



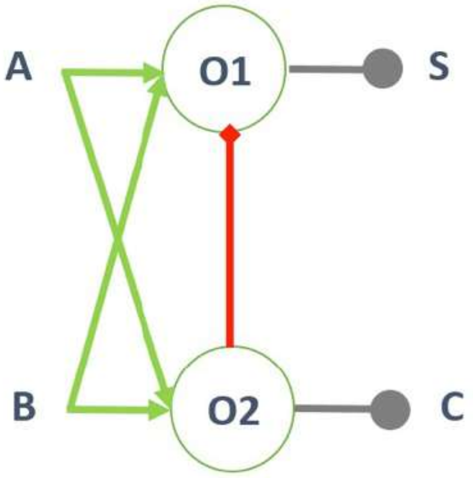
Binary Oscillator Half Adder: Arithmetic Logic



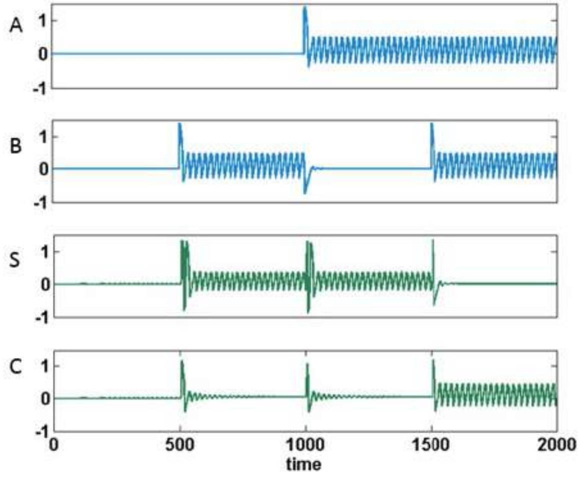
(a)

A	0	0	1	1
B	0	1	0	1
S	0	1	1	0
C	0	0	0	1

(b)



(c)



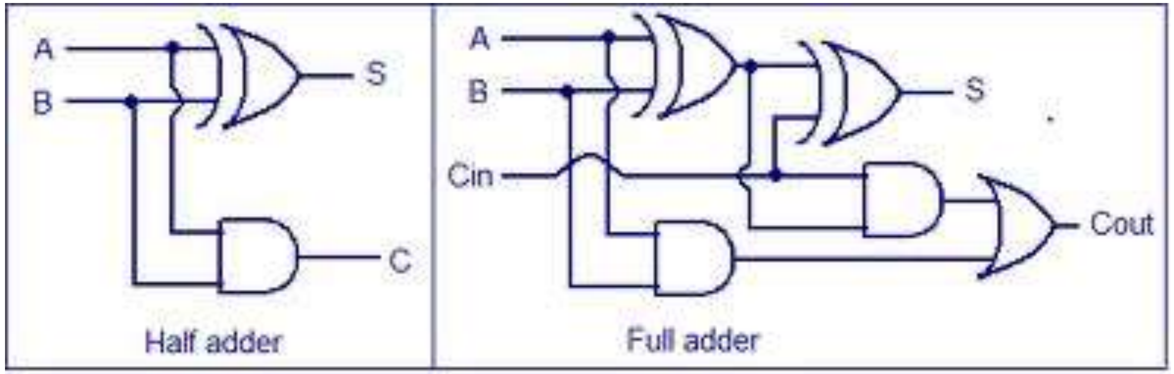
(d)

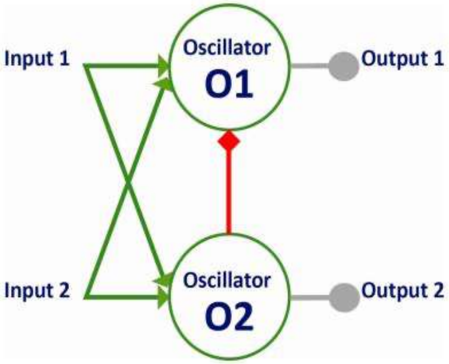
Using transistors: Two transistors in the AND gate and five in the XOR gate.

Using threshold oscillators: Four excitatory connections and one inhibitory connection.

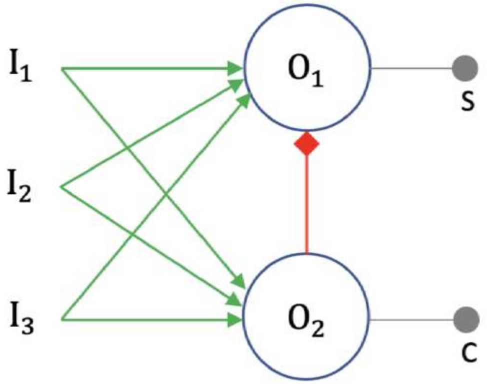
Manchester Metropolitan University

Binary Logic Circuitry





Half-adder schematic.

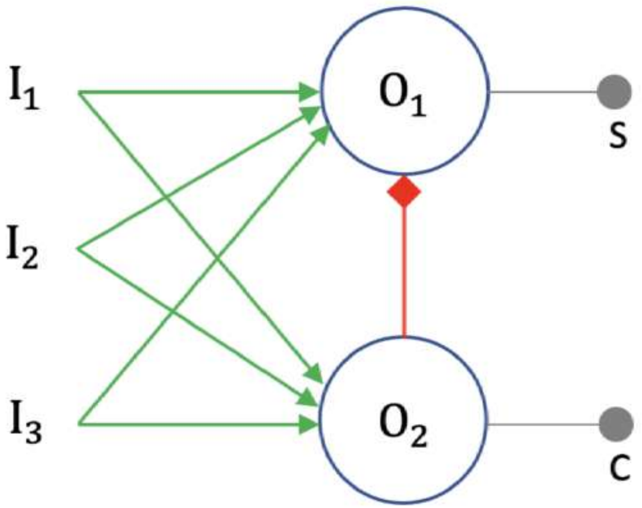


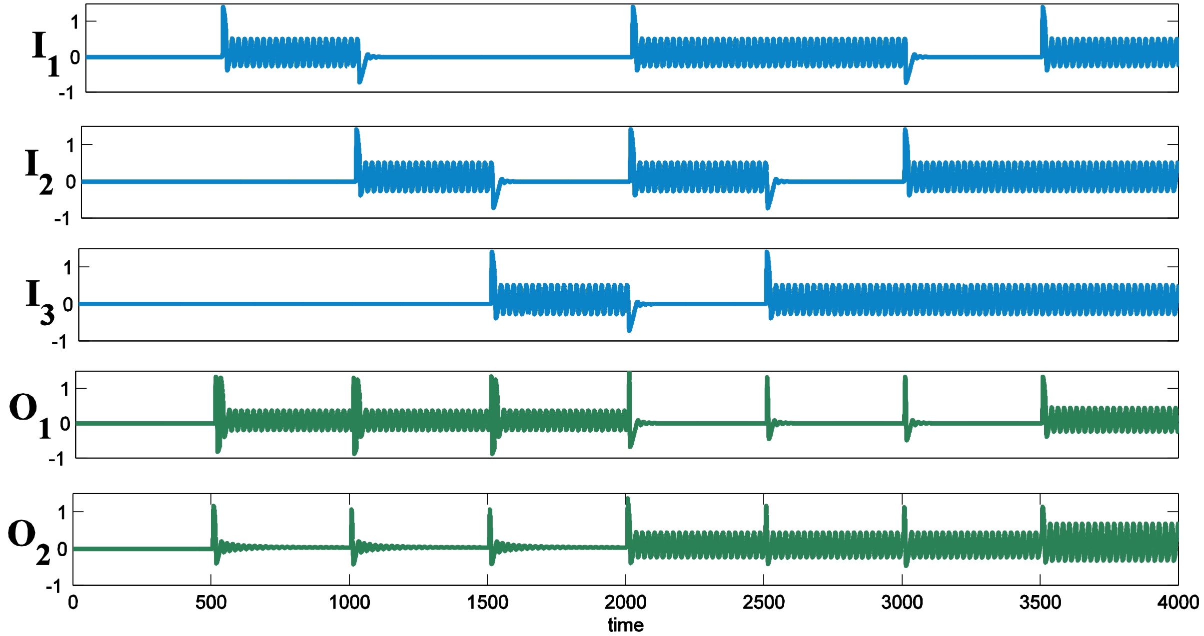
Full-adder schematic.

Using transistors: For standard designs, to double processing power it is necessary to at least double the number of components.

Using oscillators: For this design, it is possible to double processing power with a linear increase in components!

Binary Oscillator Full Adder

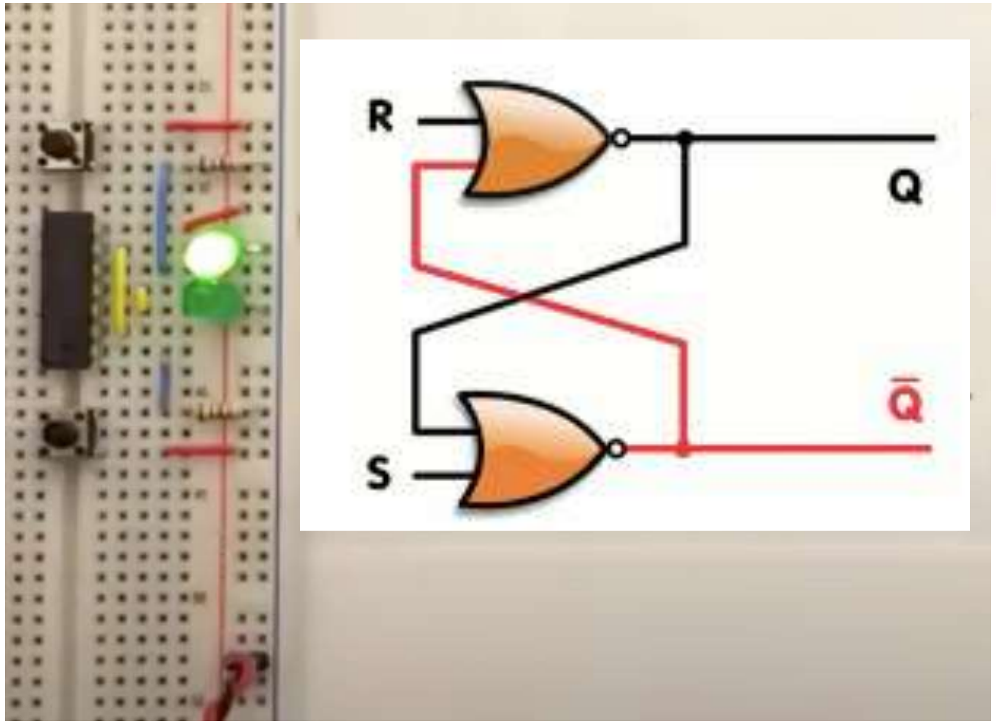


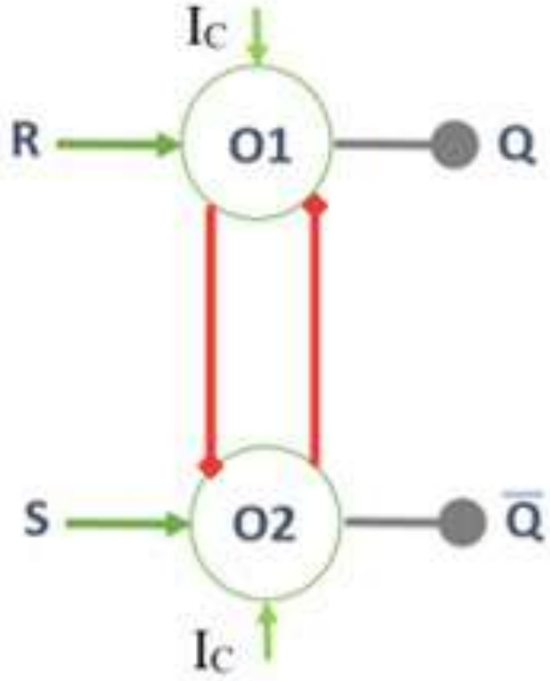


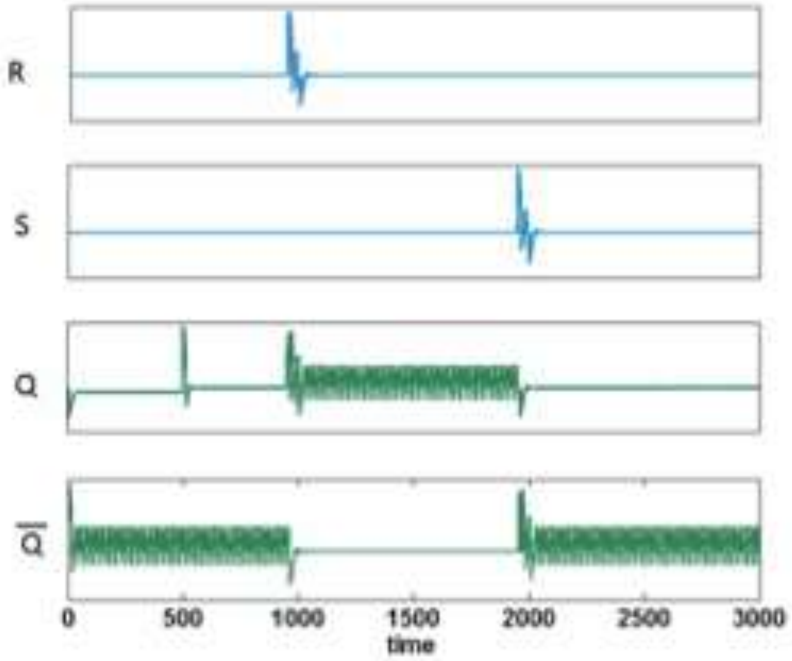
Full-adder schematic.

Input and output of a Fitzhugh-Nagumo full-adder.

Set Reset (SR) Flip-Flop: How Computers Store Memory





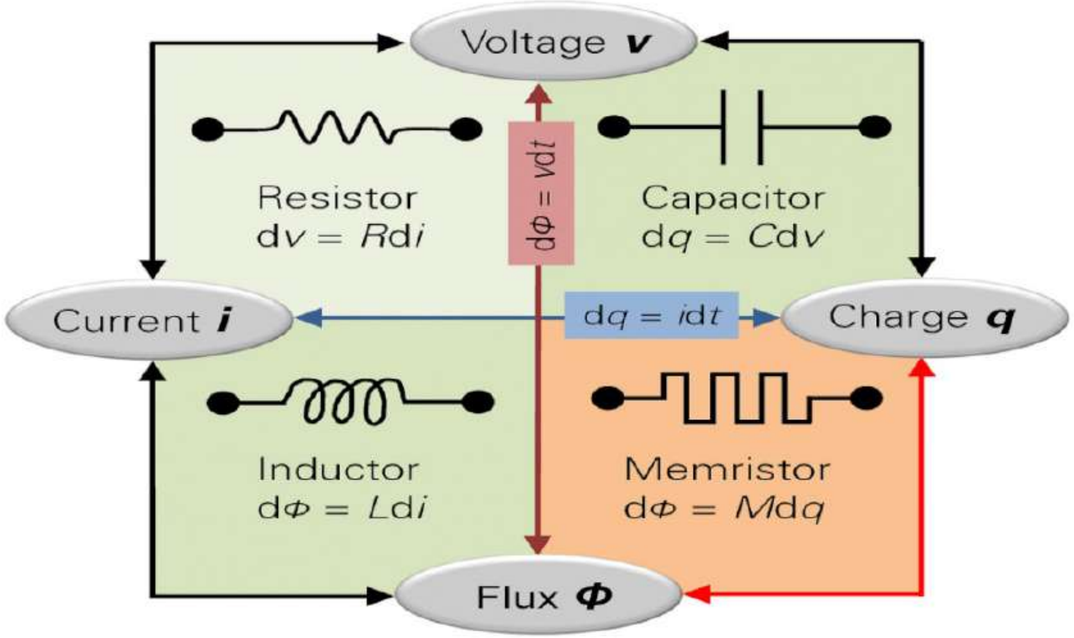


Manchester Metropolitan University

The Memristor – the Missing Circuit Element

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

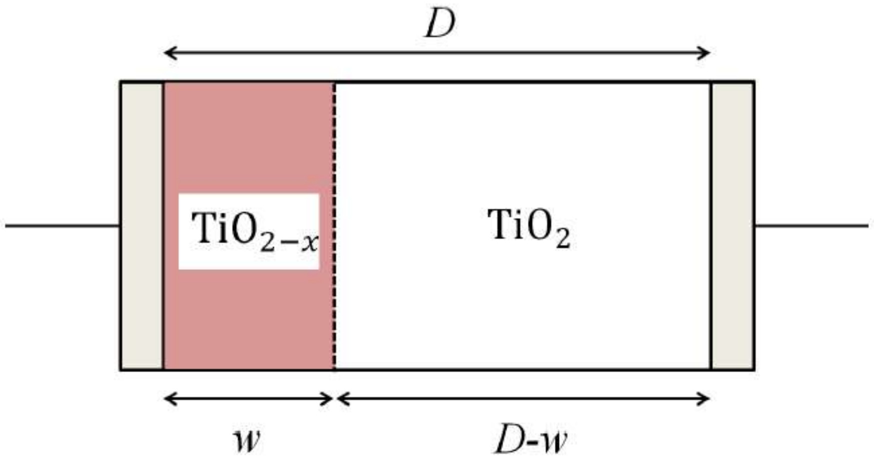
Steve Furber, Jon Borresen & Leon Chua.



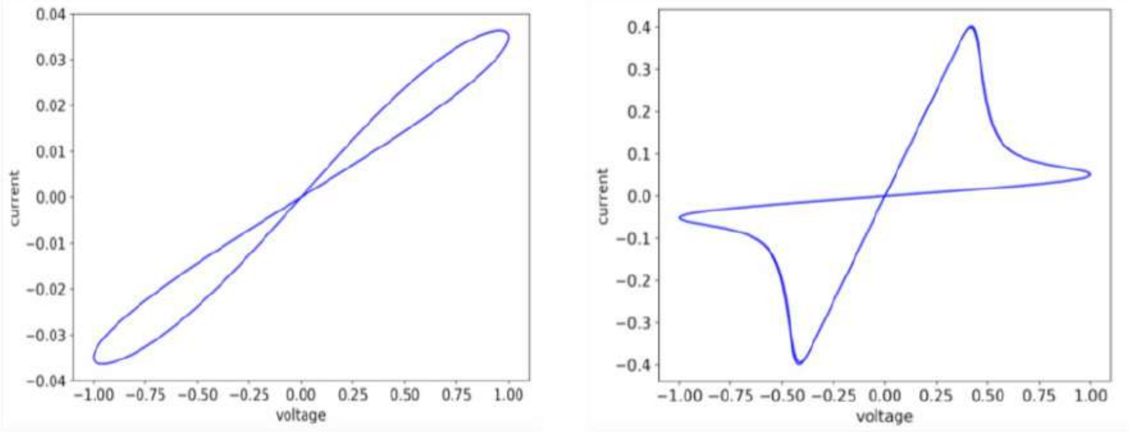
Fundamental circuit variable relationships.

Memristors

The Memristor (First built in 2008)



HP Labs titanium dioxide memristor.



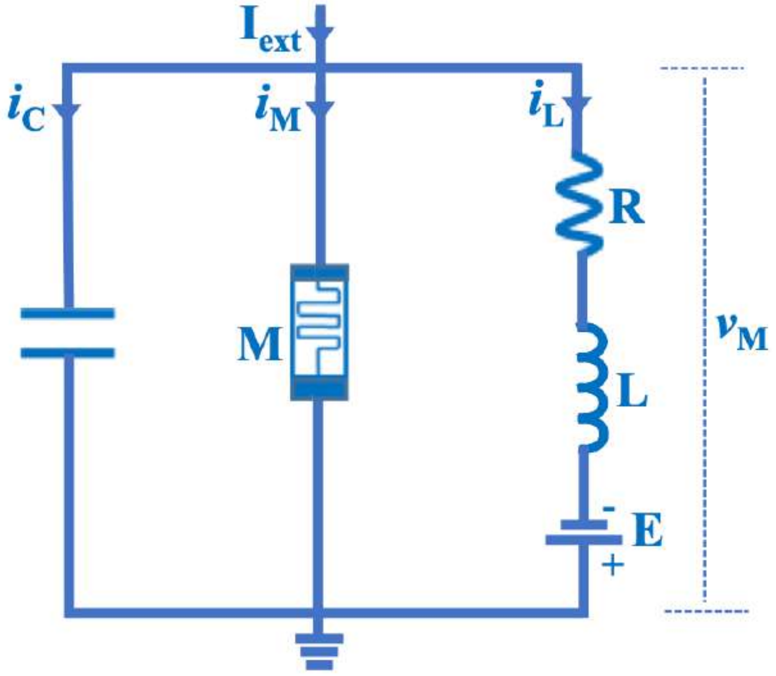
(a) (b)

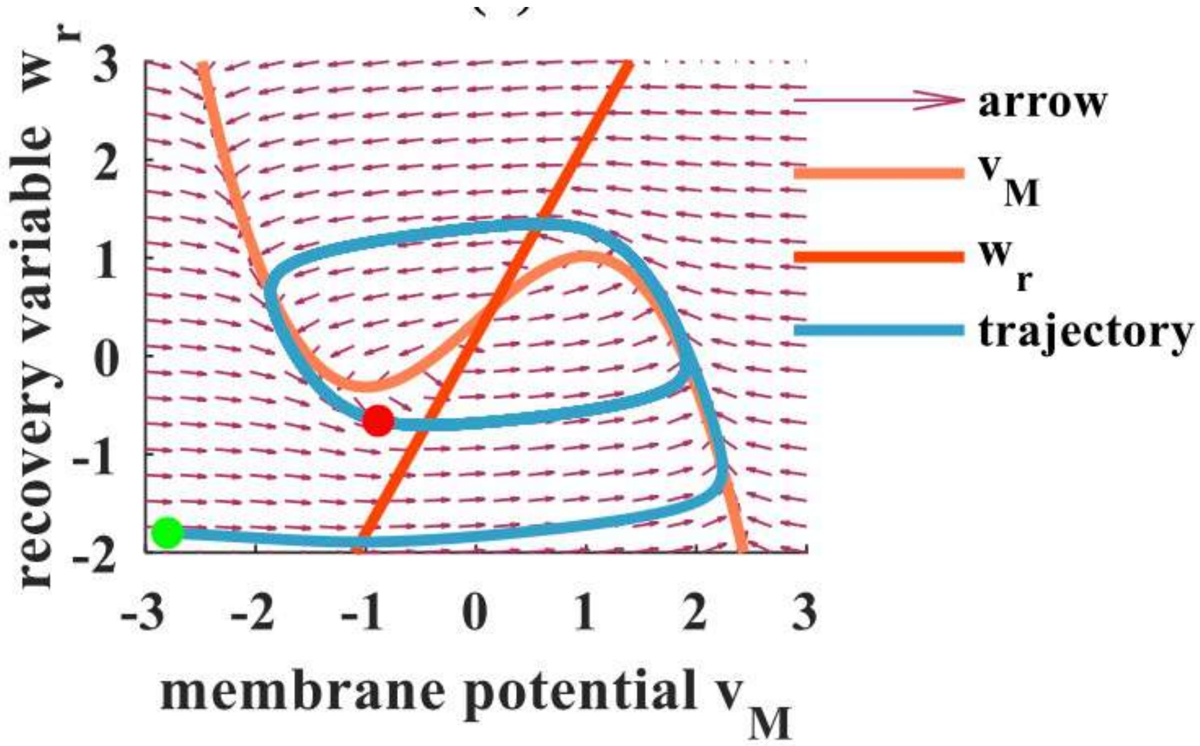
Pinched hysteresis loops of a memristor. They act like resistors with memory and form natural synaptic connections.

Neuristors (neuron-based memristors) could also be used to act like neurons and memristors act like synapses!

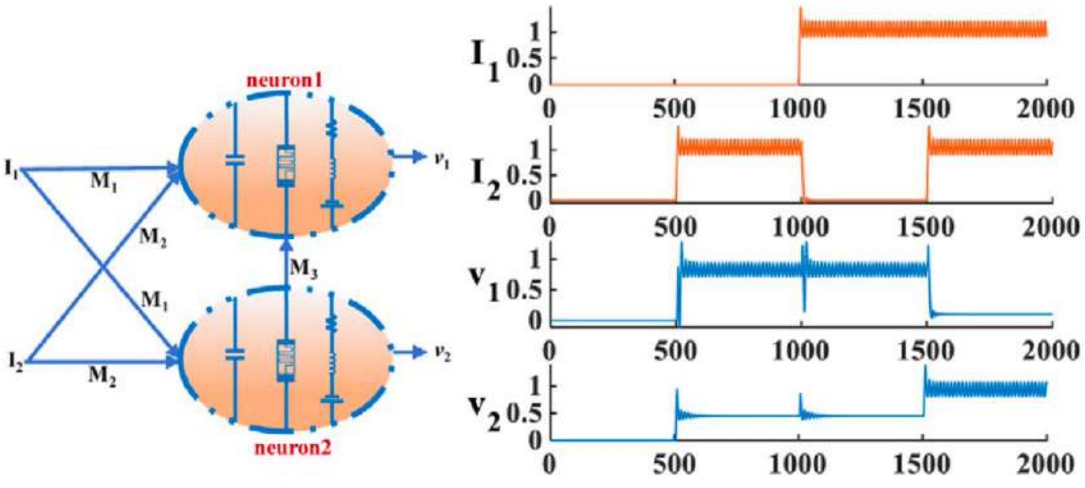
Memristors

Action potentials in Neurons: The Memristor Fitzhugh-Nagumo Model.

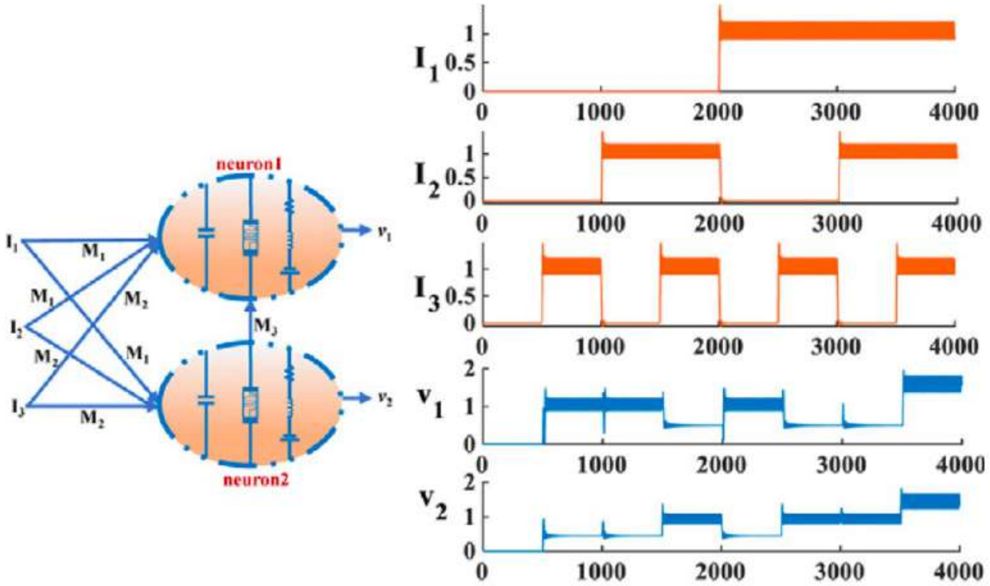




Mermristive Threshold Oscillator Logic



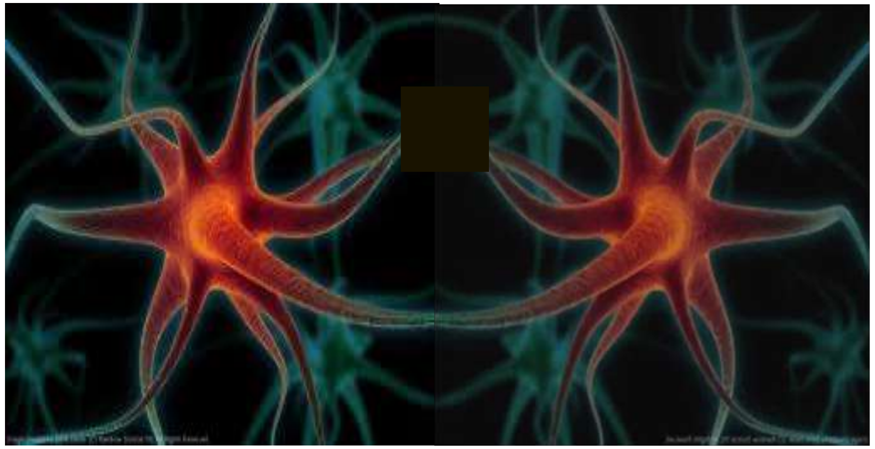
Memristor-based threshold oscillator half-adder.



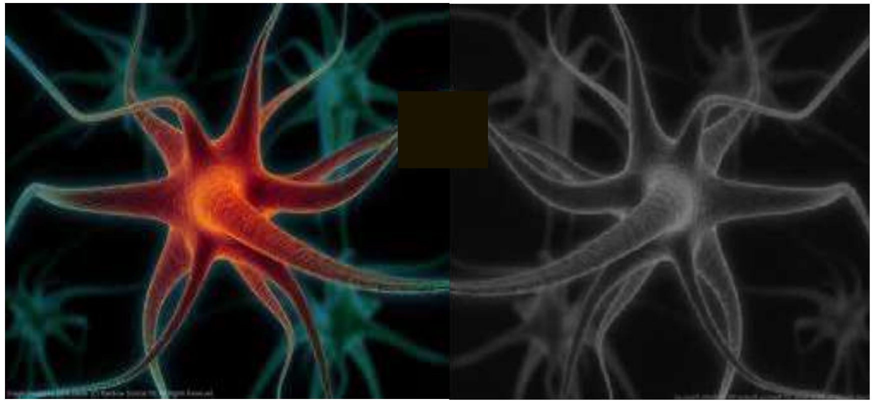
Memristor-based threshold oscillator full-adder.

Fang X, Duan S & Wang L (2022) Memristive FHN spiking neuron model and brain-inspired threshold logic computing, *Neurocomputing*, **517**, *93-105.*

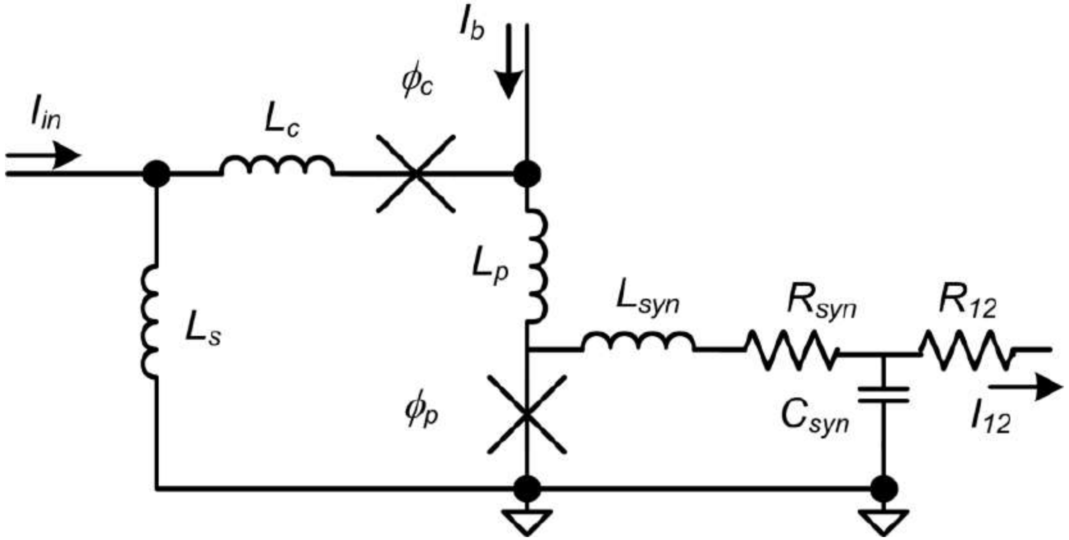
Using JJs to Model Neurons



Excitatory connection.



Inhibitory connection.

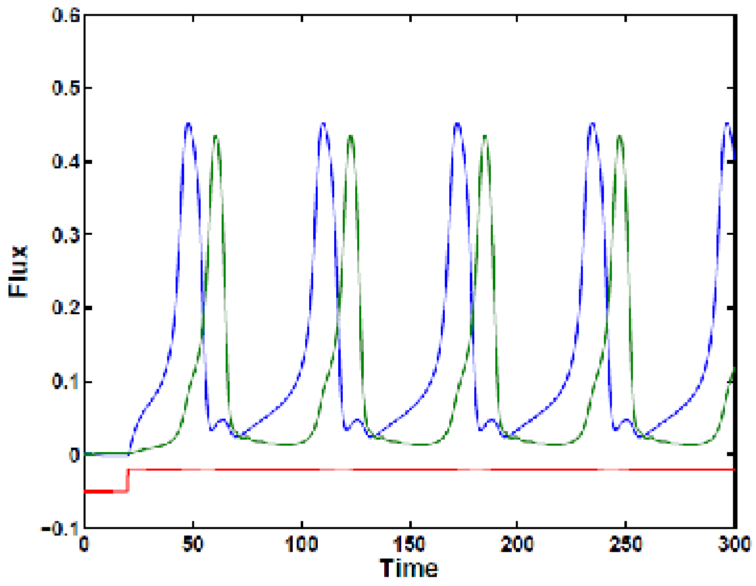


The loop on the left represents a neuron, the loop on the right is a synapse. If the bias current applied to the JJ neuron is positive (negative) with respect to ground, then the synapse is excitatory (inhibitory).

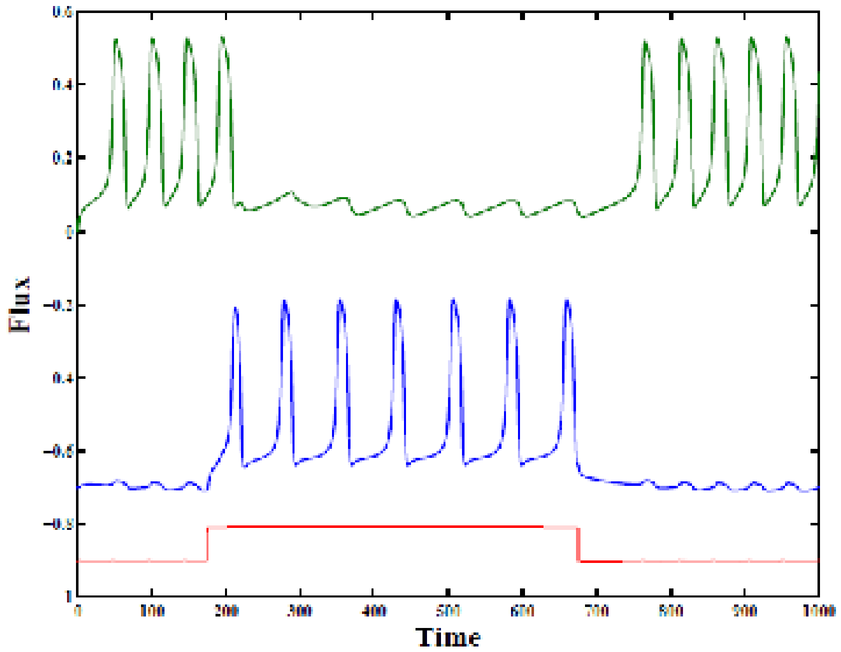
P. Crotty, D. Schult and K. Segall, Josephson junction simulation of neurons, *Phys. Rev. E* **82** (1), 011914, (2010).

Manchester Metropolitan University

Results from Phys. Rev. E Paper



JJ excitatory synaptic coupling.



JJ inhibitory synaptic coupling.

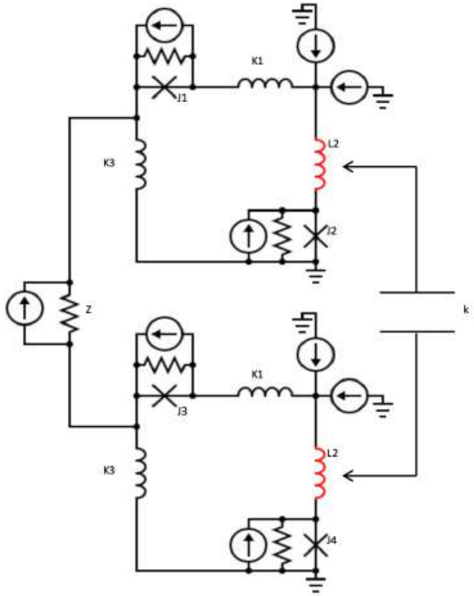
Ken Segall.

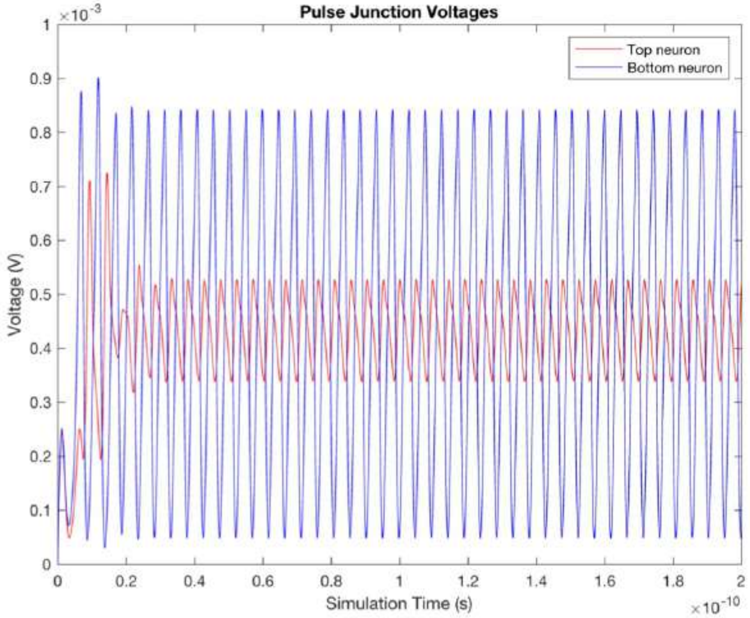
Ken Segall et al, Synchronization dynamics on the picosecond timescale in coupled Josephson junction neurons, Phys. Rev. E 95.032220 (2017). COLGATE UNIVERSITY, NEW YORK

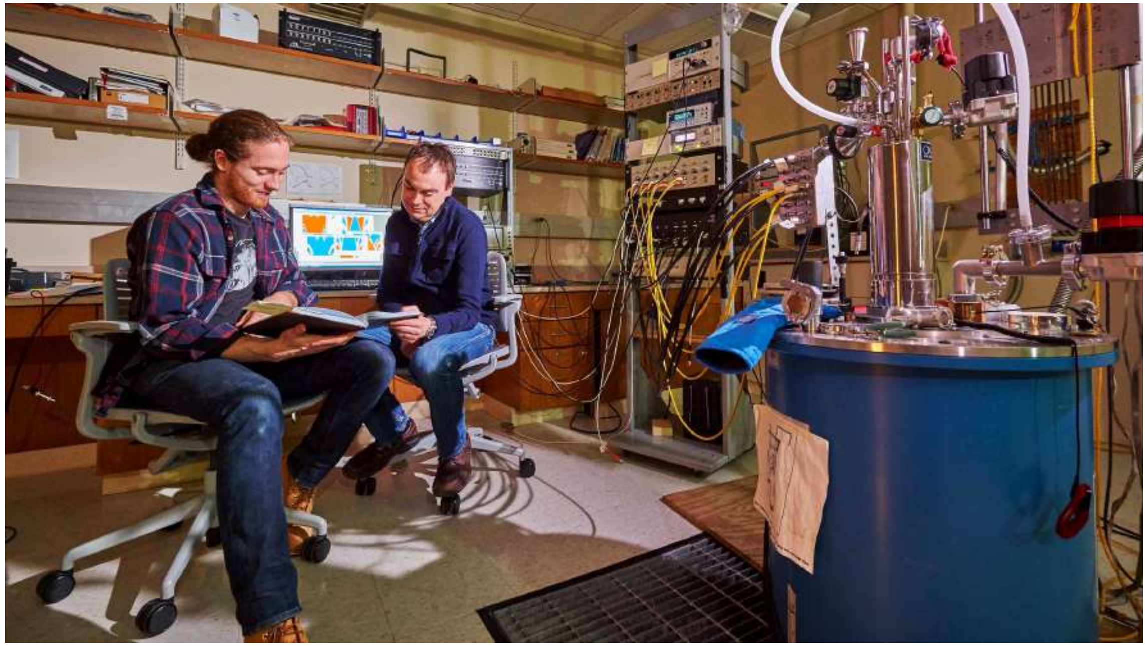
Toomey E, Segall K and Berggren KK, Design of a Power Efficient Artificial Neuron using Superconducting Nanowires. Front. Neurosci. 13:933 (2019). MIT, BOSTON

Exciting New Results

Using JJ Neurons for Memory





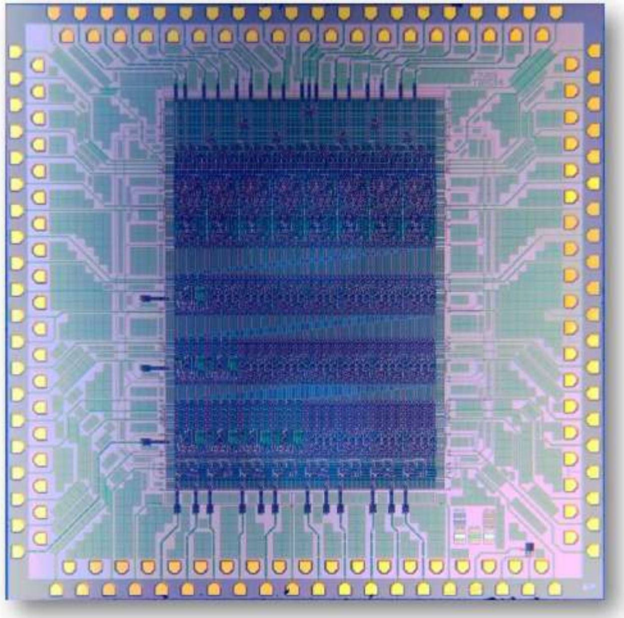


Ken Segall in his lab with a student.

We have a fully working SR JJ flip-flop!

If Our Invention Works …

Double Processing Power with a Linear Increase in Components



The world's fastest ALU chip has about 8000 JJs (Ref: HYPRES).

2022: Oak Ridge National Laboratory (ORNL) Frontier system in the US is the first to break the exaflop ceiling. It is built using thousands of trillions of transistors.

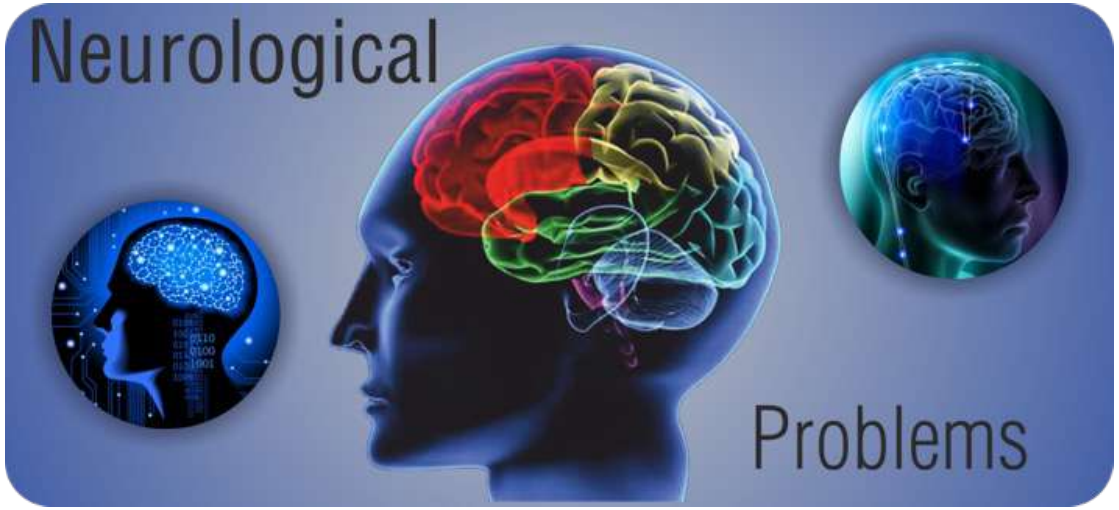
Which means that the JJ chip on the left would be more powerful than the computer on the right!

Manchester Metropolitan University

2nd Avenue of Research: An Assay for Neuronal Degradation

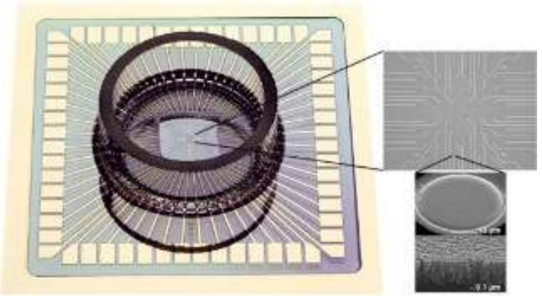
Examples of neurological conditions and disorders include:

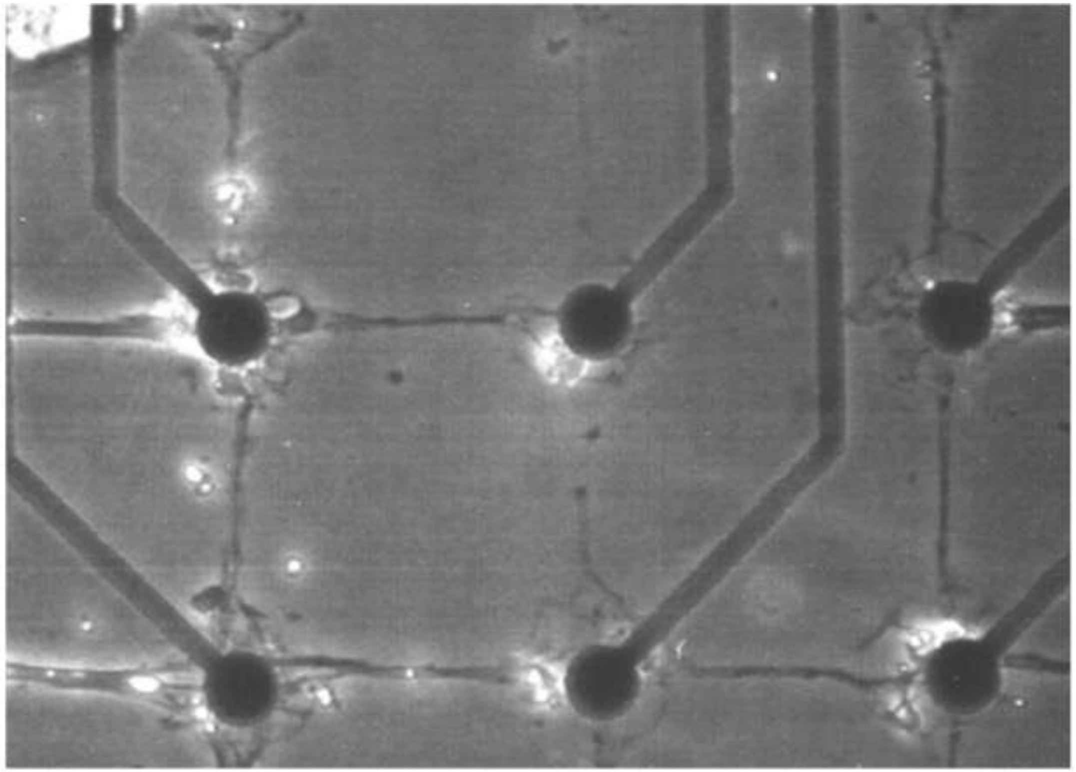
- Alzheimer's disease
- Autism
- Parkinson's disease
- Epilepsy
- Stroke and tetanus
- Brain damage
- Cerebral palsy



Neurons on a Chip

Multi-Electrode Array (MEA)





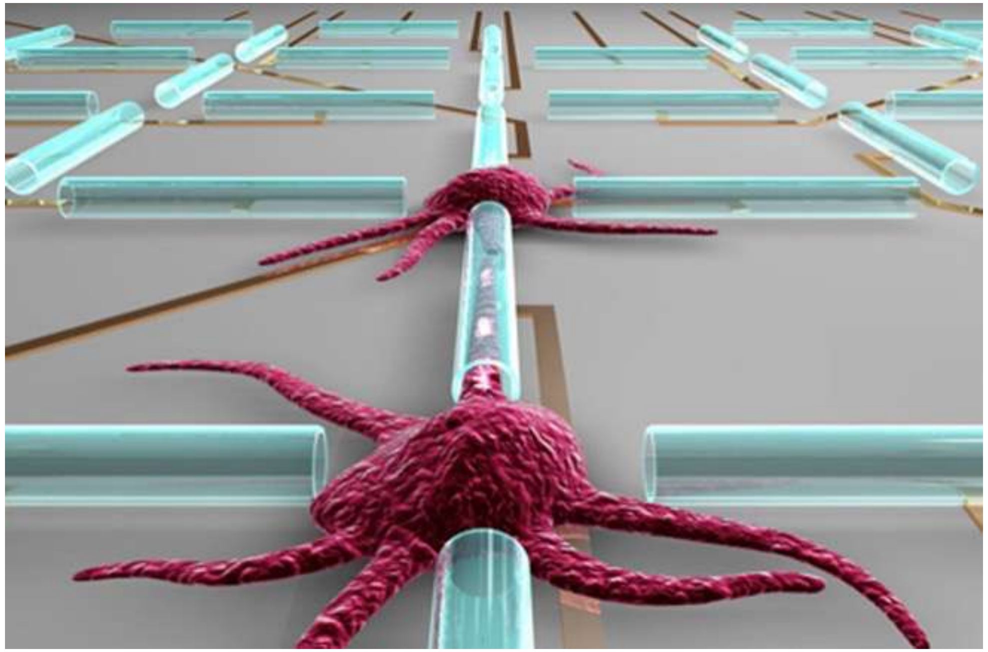
Magnification: Neurons sitting on electrodes of an MEA connected with axons.

An Assay for Neuronal Degradation

Collaborative work with Loughborough Interdisciplinary Science Centre

Eric Hill

Paul Roach



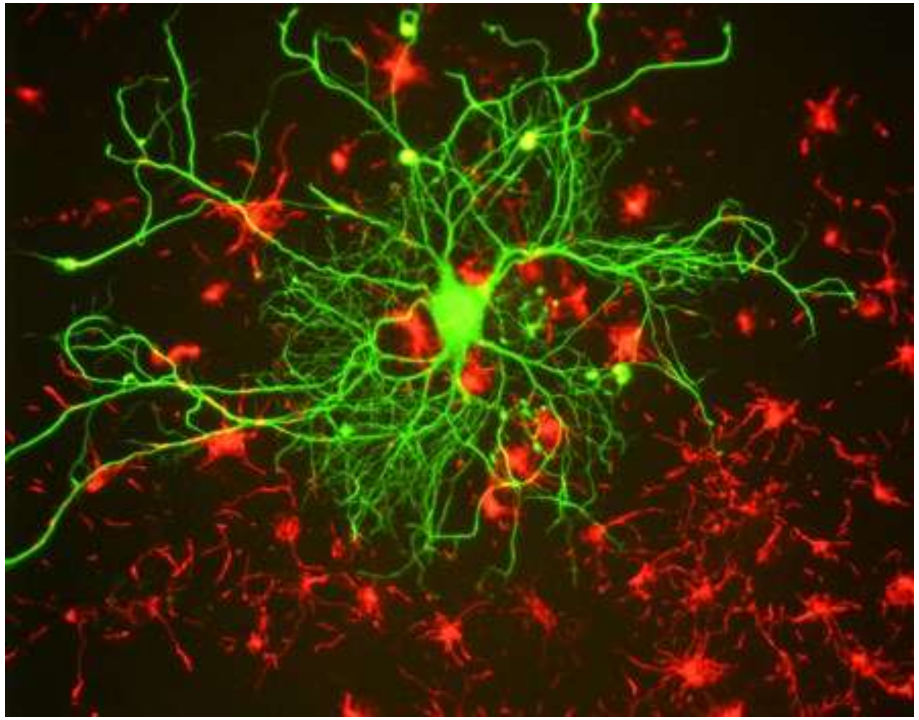
Lynch S, Borresen J, Roach P, Kotter M and Slevin MA, Mathematical modelling of neuronal logic, memory and clocking circuits. International Journal of Bifurcation and Chaos, 30, 2050003, 1-16 (2020).

In the Future … No Need for Electronics: End Session 5

Biological Neuron Circuits

In the future we can control our circuits with light.

Caenorhabditis elegans has just 302 neurons. Genetically modified worm. Neurons can be switched on and off using light.



Optogenetics: biological neurons can be controlled (switched) with light and fluorescent dyes can be used to indicate whether or not a neuron is firing.

Summary Day 3

Day 3			
Fractals and Multifractals	10am-11am	Physics and Statistics	2pm-3pm
Image Processing	11am-12pm	Brain-Inspired Computing	3pm-4pm
Numerical Methods ODEs/PDEs	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