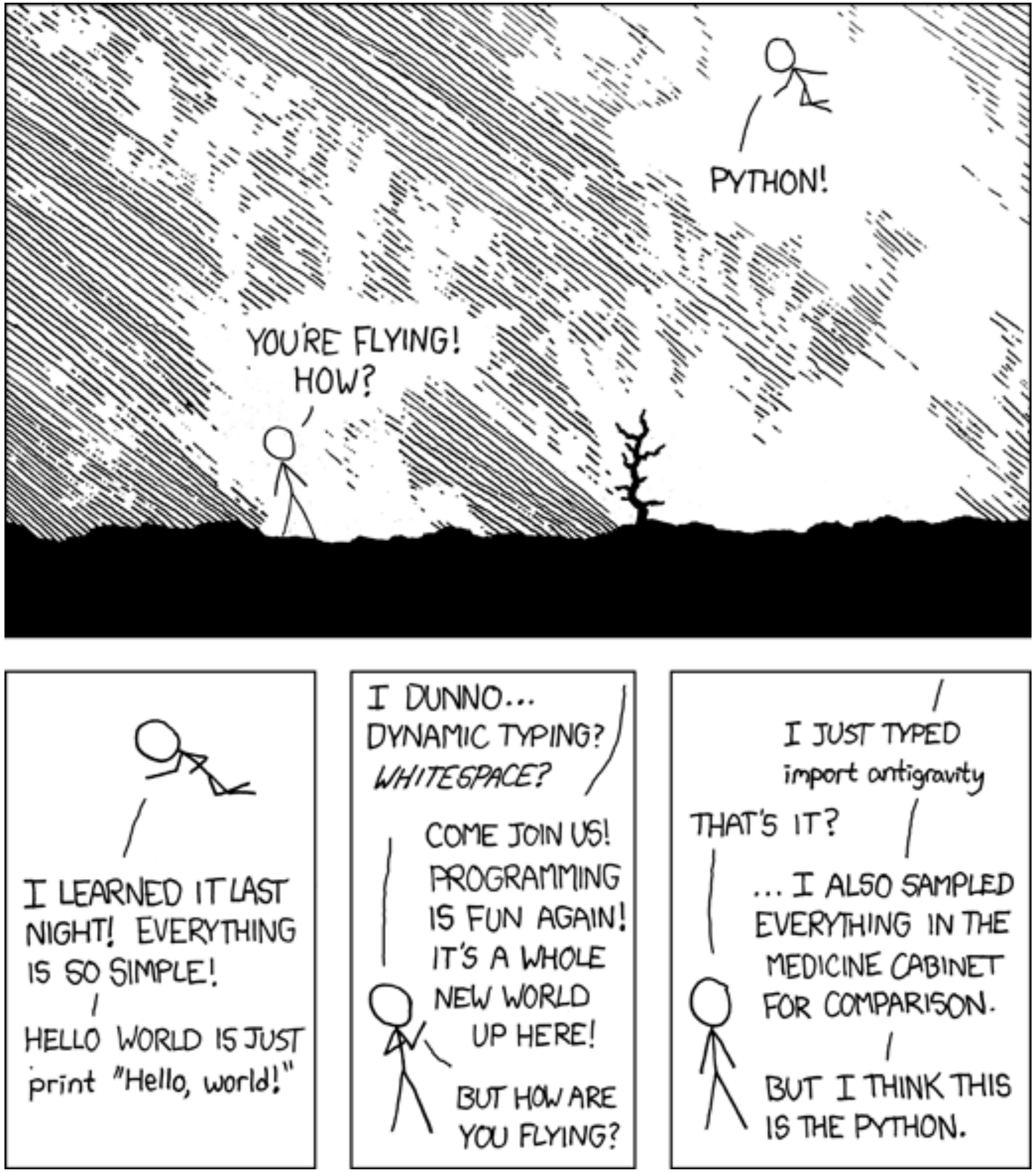


Source: https://xkcd.com/353/

Time series analysis for coupled neurons

Indranil Ghosh

https://indrag49.github.io

13 November, 2025

Notebook long URL : https://github.com/indrag49/Pycon-Ireland-Tutorial-2025

Notebook tiny URL: https://tinyurl.com/2wbj5x8c

University College Dublin

Email: indra.ghosh@ucd.ie

’Bout me!

1. B.Sc. and M.Sc. in Physics (2015–2020)
2. Ph.D. in Applied Math (2021–2024)
3. First postdoc in Applied Math (2024—2025)
4. Current postdoc in Mathematical Neuroscience (2025—Present)

যাদবপুর বিশ্ববিদ্যালয়
JADAVPUR UNIVERSITY

TE KUNENGA KI PŪREHUROA | MASSEY UNIVERSITY
UNIVERSITY OF NEW ZEALAND

University College Dublin
An Coláiste Ollscoile, Baile Átha Cliath

Dynamical systems (ODEs)

1. ODEs : Ordinary Differential Equations.

2. Rate of change of a physical quantity over time.

3. Generates a data of time series, given an initial time stamp.

$$\dot{\mathbf{x}} = \frac{d\mathbf{x}}{dt} = f(\mathbf{x}, t)$$

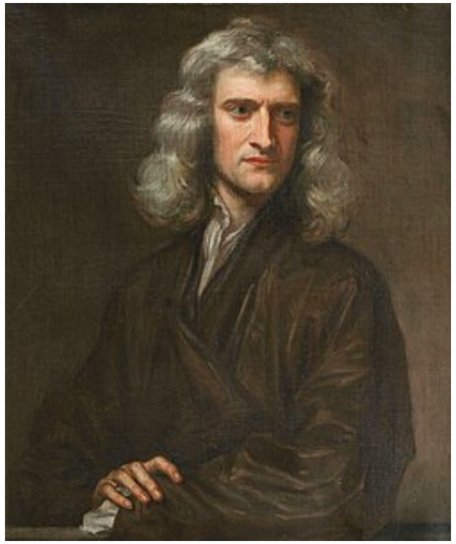


Fig. Newton and Leibniz (Wikipedia)

“From a drop of water, a logician could infer the possibility of an Atlantic or a Niagara.”

— A study in Scarlet (1887)

Source: Pinterest

Neurons

1. Neurons are the fundamental units of the nervous system.
2. Billions of neurons couple through 'synapses' to form a cluster of a highly complex neural mass.
3. Their mechanism evolves in time.
4. Thus can be perceived as a `dynamical system'.

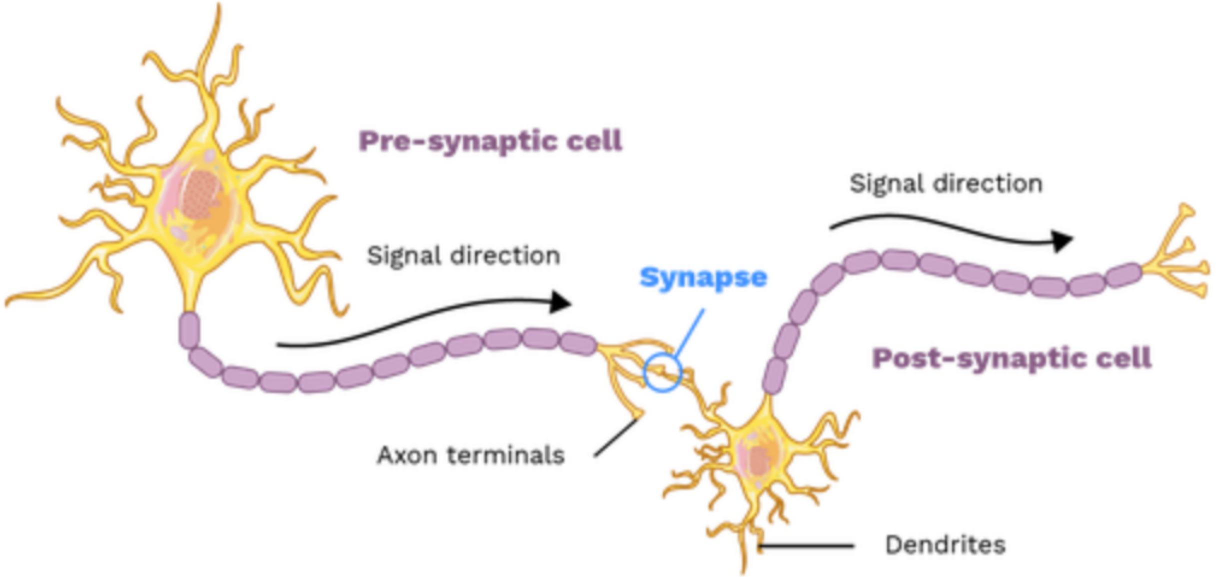


Fig. A typical synapse (theory.labster.com/synapses/)

Chaos

1. In popular term a ‘state of disorder’.

2. In mathematical term, it must be sensitive to initial conditions and have a dense orbit in the phase space.

3. Chaotic systems behave predictably in the beginning before becoming random.

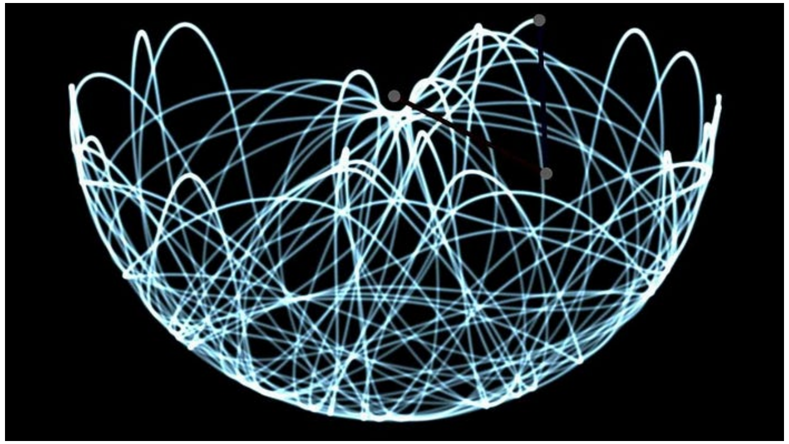


Fig. Double-rod pendulum exhibiting chaos (Taken from https://medium.com/@bharatambati/how-the-double-pendulum-creates-simple-chaos-ac49a297fb4d)

“Tell me and I will forget, show me and I may remember; involve me and I will understand.”

Fig. Confucius (Wikipedia)

Packages

Single neuron (Schaeffer & Cain, 2018)

1. Simple mathematical model.

2. Parameters selected from empirical experiments.

3. Captures realistic bursting in neurons.

4. Portrays a battery of complex dynamics.

5. Use `solve_ivp()`function from `scipy.integrate` suite to solve initial value problem.

$$
\begin{aligned}
\dot{x} &= f(x, y, I) = x^2(1-x) - y + I, \\
\dot{y} &= g(x, y, I) = Ae^{\alpha x} - \gamma y, \\
\dot{I} &= h(x, y, I) = \varepsilon\left[\frac{1}{60}\left\{1 + \tanh\left(\frac{0.05 - x}{0.001}\right)\right\} - I\right]
\end{aligned}
$$

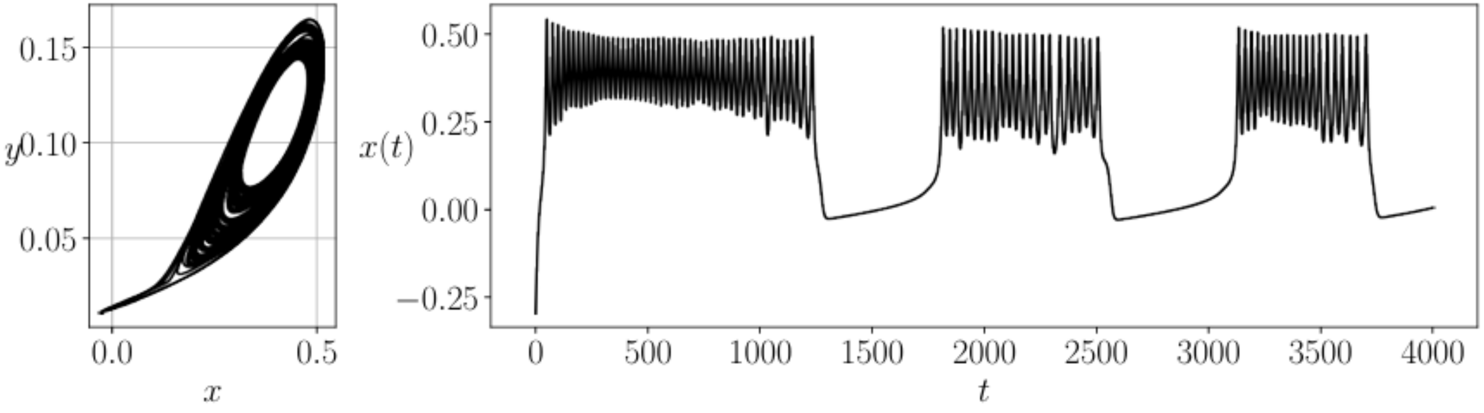


Fig. Phase portrait and time series

“All models are wrong but some are useful”

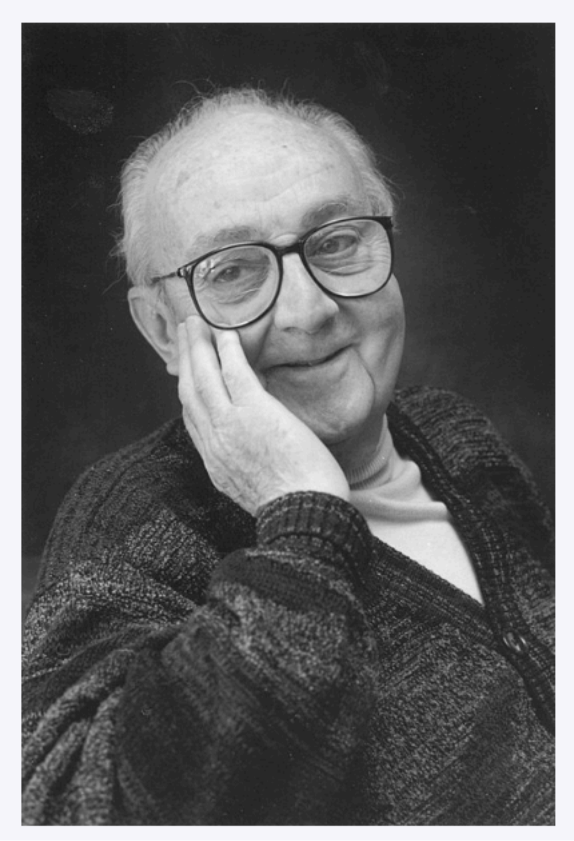


Fig. George Box (Wikipedia)

Simulate a single neuron (code snippet)

```python
## Parameters
A = 0.0041
alpha=5.276
gamma = 0.315
epsilon = 0.0005

## Define the function of differential equations
def system(t, vars):
    x1, y1, I1= vars
    dx1dt = x1**2 * (1 - x1) - y1 + I1
    dy1dt = A * np.exp(alpha * x1) - gamma * y1
    dI1dt = epsilon*(1/60*(1+np.tanh((0.05-x1)/0.001)) - I1)

    return [dx1dt, dy1dt, dI1dt]

## Initial conditions
x1_0 = np.random.uniform(low=-1, high=1)
y1_0 = 0.1
I1_0 = 0.019

initial_conditions = [x1_0, y1_0, I1_0]
```

```python
## Time span for the solution
t_span = (0, 4000)
t_eval = np.linspace(t_span[0], t_span[1], 50000)

## Solve
solution = solve_ivp(system, t_span, initial_conditions, t_eval=t_eval, method='RK45')

## Extract solutions
x1_sol = solution.y[0]
y1_sol = solution.y[1]
I1_sol = solution.y[2]


tt = solution.t
print("done")
```

For the time integration of the differential equations, we use method = ‘RK45’ which is the explicit Runge-Kutta scheme of order 5(4).

Coupled neurons

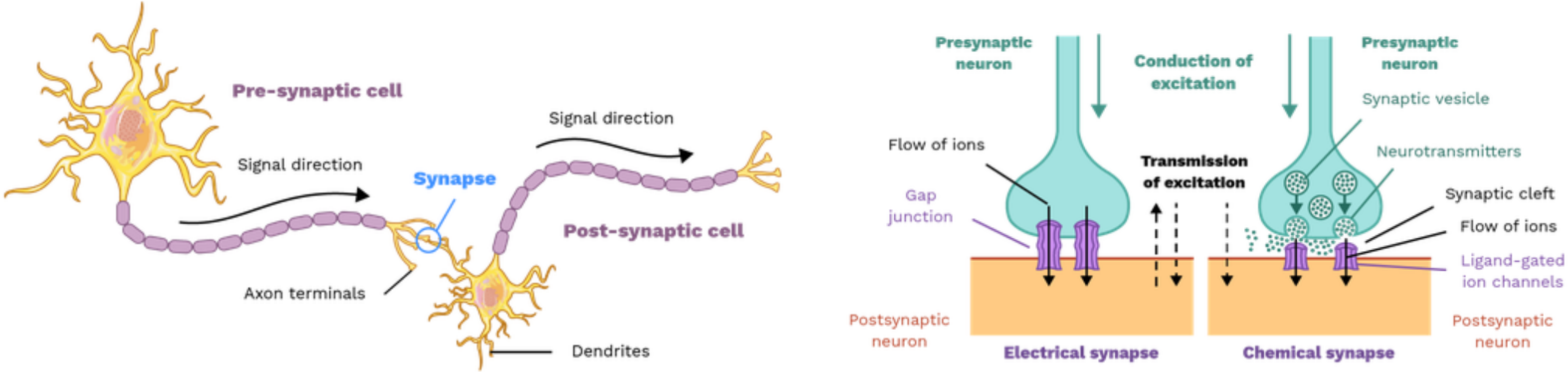


Fig. Coupled neurons (https://theory.labster.com/synapses/), and typical electrical and chemical synapses (theory.labster.com/electrical-synapses/)

Toy models of coupled neurons

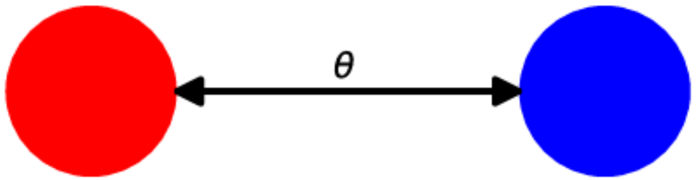


Fig. Electrical (gap-junction) coupling

$$\dot{x}_i = f(x_i, y_i, I_i) + \sum_{j \in B(i)} \theta(x_j - x_i),$$

$$\dot{y}_i = g(x_i, y_i, I_i),$$

$$\dot{I}_i = h(x_i, y_i, I_i),$$

Coupling term

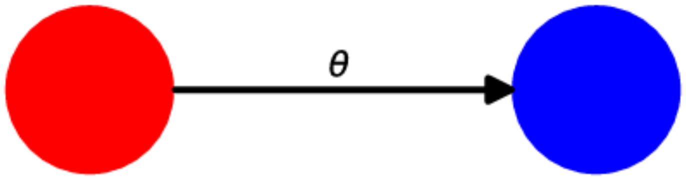


Fig. Chemical coupling

$$\dot{x}_1 = f(x_1, y_1, I_1),$$

$$\dot{x}_2 = f(x_2, y_2, I_2) + \theta \frac{v_s - x_2}{1 + \exp\{-\lambda(x_1 - q)\}},$$

$$\dot{y}_i = g(x_i, y_i, I_i),$$

$$\dot{I}_i = h(x_i, y_i, I_i),$$

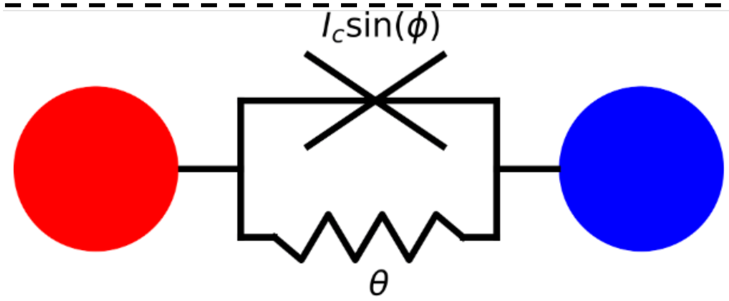


Fig. Josephson junction coupling

$$\dot{x}_1 = f(x_1, y_1, I_1) - I_c \sin(\phi) + \theta(x_2 - x_1),$$

$$\dot{x}_2 = f(x_2, y_2, I_2) + I_c \sin(\phi) + \theta(x_1 - x_2),$$

$$\dot{y}_i = g(x_i, y_i, I_i),$$

$$\dot{I}_i = h(x_i, y_i, I_i), \qquad i = 1, 2,$$

$$\dot{\phi} = \mu(x_1 - x_2),$$

Toy models of coupled neurons (cont.)

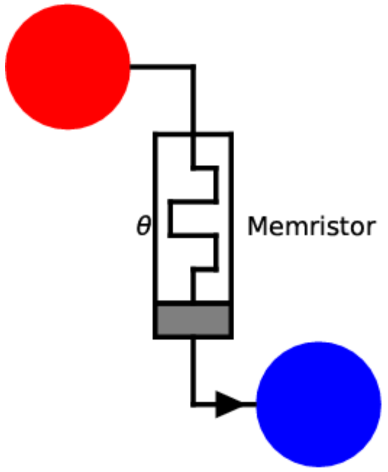


Fig. Electromagnetic coupling

$$\begin{aligned}
\dot{x}_1 &= f(x_1, y_1, I_1) + \theta\rho(\phi)(x_2 - x_1), \\
\dot{x}_2 &= f(x_2, y_2, I_2) + \theta\rho(\phi)(x_1 - x_2), \\
\dot{y}_i &= g(x_i, y_i, I_i), \\
\dot{I}_i &= h(x_i, y_i, I_i), \qquad i = 1, 2, \\
\dot{\phi} &= \theta(x_1 - x_2)
\end{aligned}$$

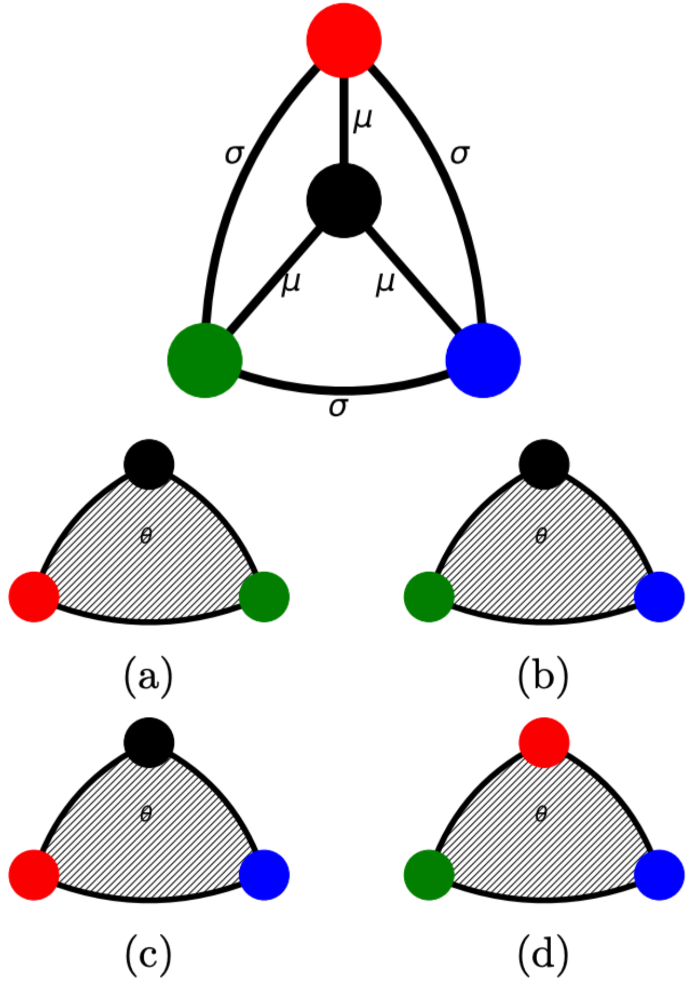


Fig. Higher-order coupling

$$\begin{aligned}
\dot{x}_1 &= f(x_1, y_1, I_1) + (\mu + 2\theta)(x_2 + x_3 + x_4 - 3x_1), \\
\dot{x}_2 &= f(x_2, y_2, I_2) + \mu(x_1 - x_2) + \sigma(x_3 + x_4 - 2x_2) \\
&\quad + 2\theta(x_1 + x_3 + x_4 - 3x_2), \\
\dot{x}_3 &= f(x_3, y_3, I_3) + \mu(x_1 - x_3) + \sigma(x_2 + x_4 - 2x_3) \\
&\quad + 2\theta(x_1 + x_2 + x_4 - 3x_3), \\
\dot{x}_4 &= f(x_4, y_4, I_4) + \mu(x_1 - x_4) + \sigma(x_2 + x_3 - 2x_4) \\
&\quad + 2\theta(x_1 + x_2 + x_3 - 3x_4), \\
\dot{y}_i &= g(x_i, y_i, I_i), \\
\dot{I}_i &= h(x_i, y_i, I_i), \quad i = 1, \ldots, 4.
\end{aligned}$$

Simulating coupled neurons

```
def system(t, vars):
    x1, y1, I1, x2, y2, I2= vars
    dx1dt = x1**2 * (1 - x1) - y1 + I1+ theta*(x2-x1)
    dy1dt = A * np.exp(alpha * x1) - gamma * y1
    dI1dt = epsilon*(1/60*(1+np.tanh((0.05-x1)/0.001)) - I1)
    dx2dt = x2**2 * (1 - x2) - y2 + I2 + theta*(x1-x2)
    dy2dt = A * np.exp(alpha * x2) - gamma * y2
    dI2dt = epsilon*(1/60*(1+np.tanh((0.05-x2)/0.001)) - I2)

    return [dx1dt, dy1dt, dI1dt, dx2dt, dy2dt, dI2dt]
```

```
def system(t, vars):
    x1, y1, I1, x2, y2, I2= vars
    dx1dt = x1**2 * (1 - x1) - y1 + I1
    dy1dt = A * np.exp(alpha * x1) - gamma * y1
    dI1dt = epsilon*(1/60*(1+np.tanh((0.05-x1)/0.001)) - I1)
    dx2dt = x2**2 * (1 - x2) - y2 + I2 + theta*(vs-x2)/(1+np.exp(-lamb*(x1-q)))
    dy2dt = A * np.exp(alpha * x2) - gamma * y2
    dI2dt = epsilon*(1/60*(1+np.tanh((0.05-x2)/0.001)) - I2)

    return [dx1dt, dy1dt, dI1dt, dx2dt, dy2dt, dI2dt]
```

```
def system(t, vars):
    x1, y1, I1, x2, y2, I2, p= vars
    dx1dt = x1**2 * (1 - x1) - y1 + I1 + theta*rho(p)*(x2 - x1)
    dy1dt = A * np.exp(alpha * x1) - gamma * y1
    dI1dt = epsilon*(1/60*(1+np.tanh((0.05-x1)/0.001)) - I1)
    dx2dt = x2**2 * (1 - x2) - y2 + I2 + theta*rho(p)*(x1 - x2)
    dy2dt = A * np.exp(alpha * x2) - gamma * y2
    dI2dt = epsilon*(1/60*(1+np.tanh((0.05-x2)/0.001)) - I2)
    dpdt = theta*(x1-x2)

    return [dx1dt, dy1dt, dI1dt, dx2dt, dy2dt, dI2dt, dpdt]
```

A sample time series

	time	x1	y1	x2	y2
0	0.00000	-0.454872	0.100000	-0.980092	0.100000
1	0.40004	-5.095041	83.147156	2.181691	125.645293
2	0.80008	-3.699971	73.302843	-4.646950	111.866506
3	1.20012	-3.511443	64.624060	-4.461514	98.621930
4	1.60016	-3.324087	56.972813	-4.279355	86.945462
...	...	...	...	...	...
9995	3998.39984	-4.165945	90.343734	-2.560356	22.537029
9996	3998.79988	-3.977154	79.647374	-4.398568	103.808435
9997	3999.19992	-3.705108	70.217422	-4.290187	91.517904
9998	3999.59996	-3.528641	61.903942	-4.121808	80.682526
9999	4000.00000	-3.353111	54.574747	-3.954512	71.130017

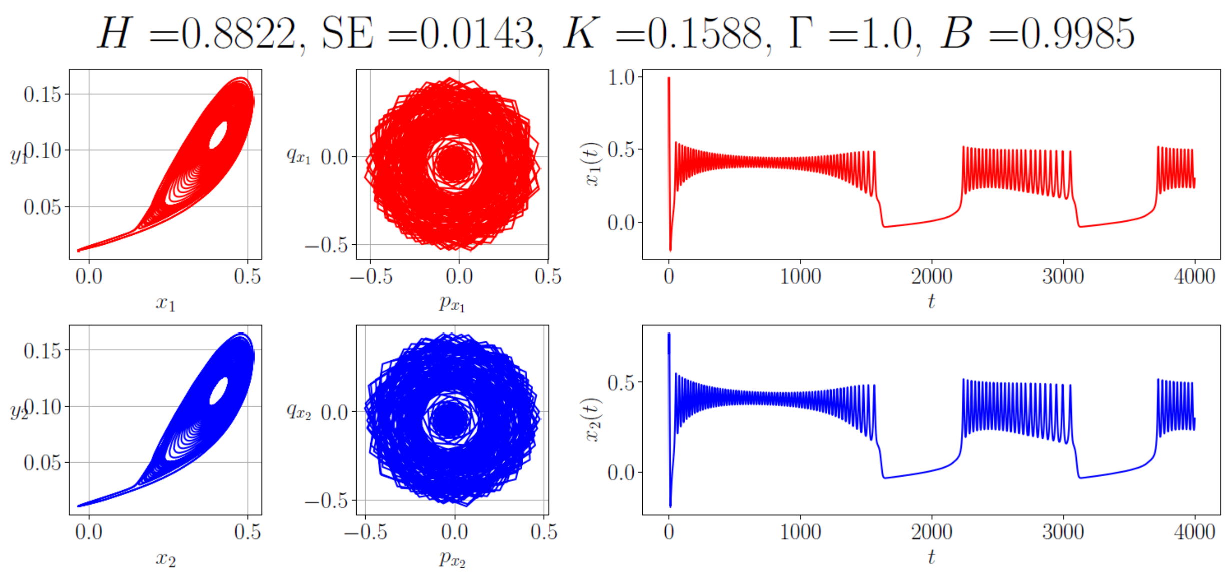


Fig. Applying various tools on the time series generated from simulating the models of coupled neurons.

Time series tools

1. Hurst exponent (H): measuring persistence.
2. Sample entropy (SE): measuring complexity.
3. 0-1 test (K): measuring chaos.
4. Cross-correlation function (Γ): measuring synchrony between neurons.
5. Kuramoto order parameter (B): measuring synchrony between neurons.

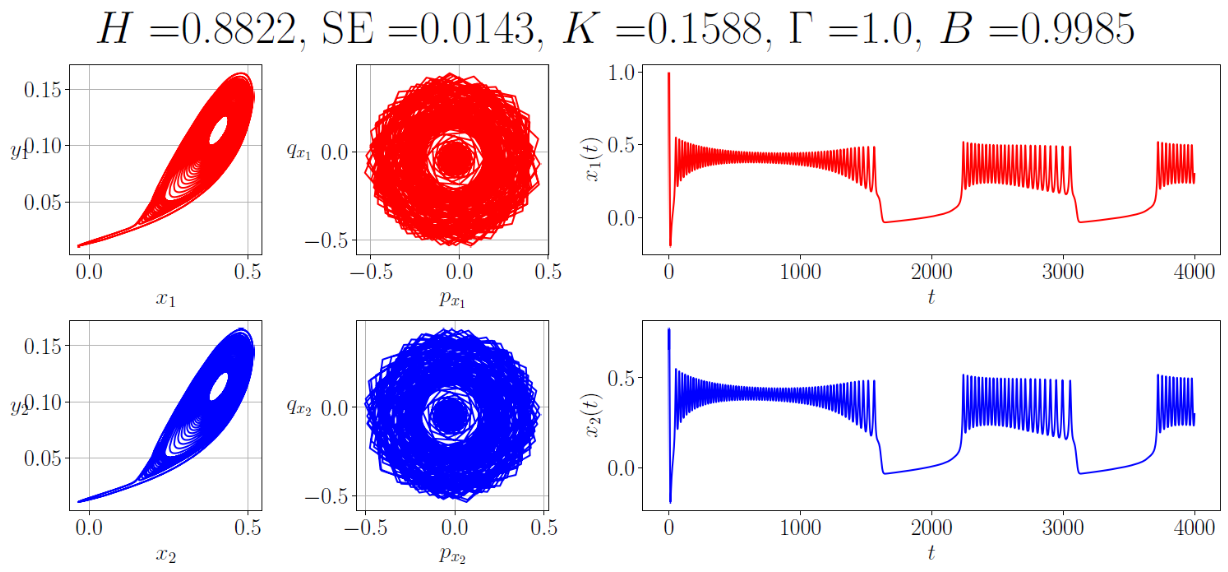


Fig. Applying various tools on the time series generated from simulating the models of coupled neurons.

Hurst exponent (Hurst, 1951)

1. Measures the long-term memory/persistence in time series.

2. Computed using rescaled-range analysis (Qian and Rasheed, 2004).

$$\mathbb{E}\left[\frac{R(t)}{S(t)}\right] = ct^H, \qquad t \to \infty$$

3. $H \in [0,1]$.

4. $H \in [0,0.5)$: anti-persistence (negative dependence on previous values), $H \approx 0.5$: random walk, $H \in (0.5,1]$: positive dependence on previous values.

Sample entropy (Richman & Moorman, 2000)

1. Assesses the complexity of time series data.

2. It is the negative natural log of the probability that if two sets of simultaneous data points of length 'p' have distance less than 'ε', then the similar thing happens to two sets of simultaneous data points of length 'p+1'.

$$\mathrm{SE}(p, \varepsilon, N) = \lim_{N \to \infty} \left(-\log_e \frac{A^p(\varepsilon)}{B^p(\varepsilon)} \right)$$

3. A higher SE indicates higher complexity.

4. Can be normalised between 0 and 1.

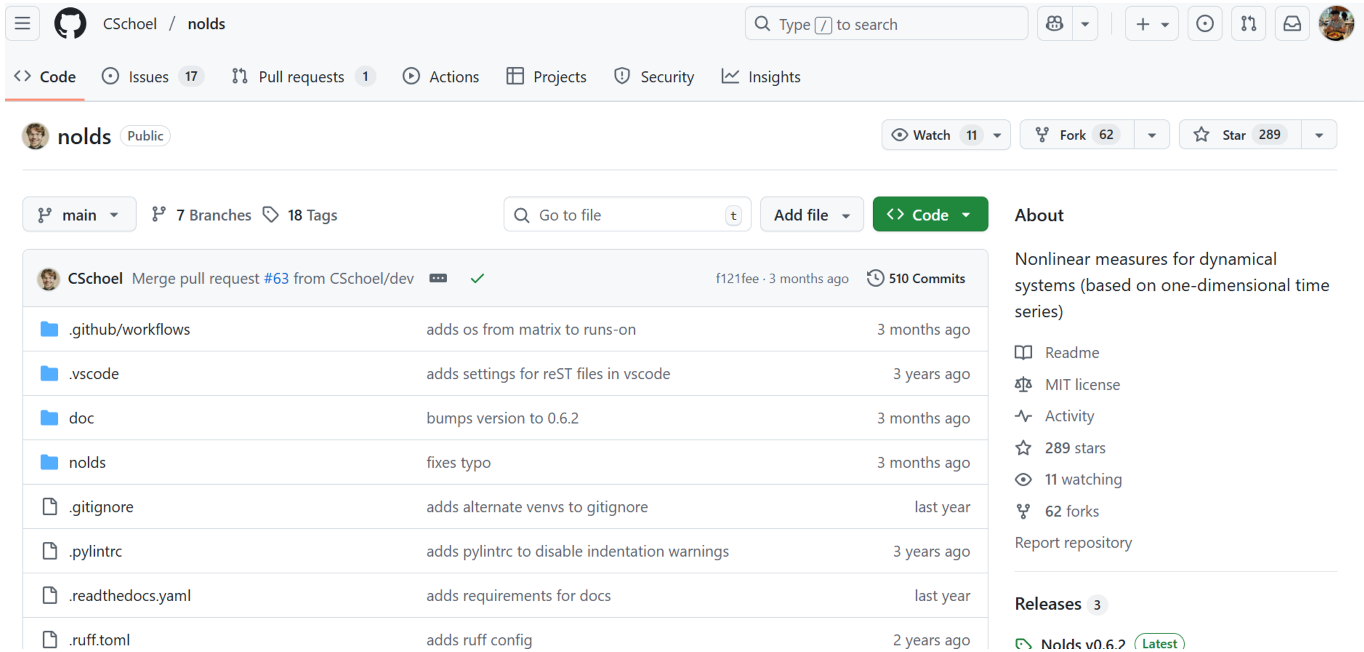
‘nolds’ package (Scholzel, 2019)

1. Stands for ‘NOnLinear measures for Dynamical Systems’, based on .

2. Provides functions for directly implementing the rescaled-range based Hurst exponent and also sample entropy to time series.

3. Functions are `nolds.hurst_rs()` and `nolds.sampen()`.

4. Also provides other sophisticated tools for nonlinear measures.



0-1 test (Gottwald and Melbourne, 2009, 2016)

1. Compute two translated variables from the time series.

$$\tilde{p}(t;e) = \sum_{k=1}^{t} x(k)\cos(ek),$$
$$\tilde{q}(t;e) = \sum_{k=1}^{t} x(k)\sin(ek).$$

2. Then compute a mean square displacement term ‘m(t; e)’ and a correction term.

3. Compute the growth rate ‘K’ that quantifies ‘chaos’.

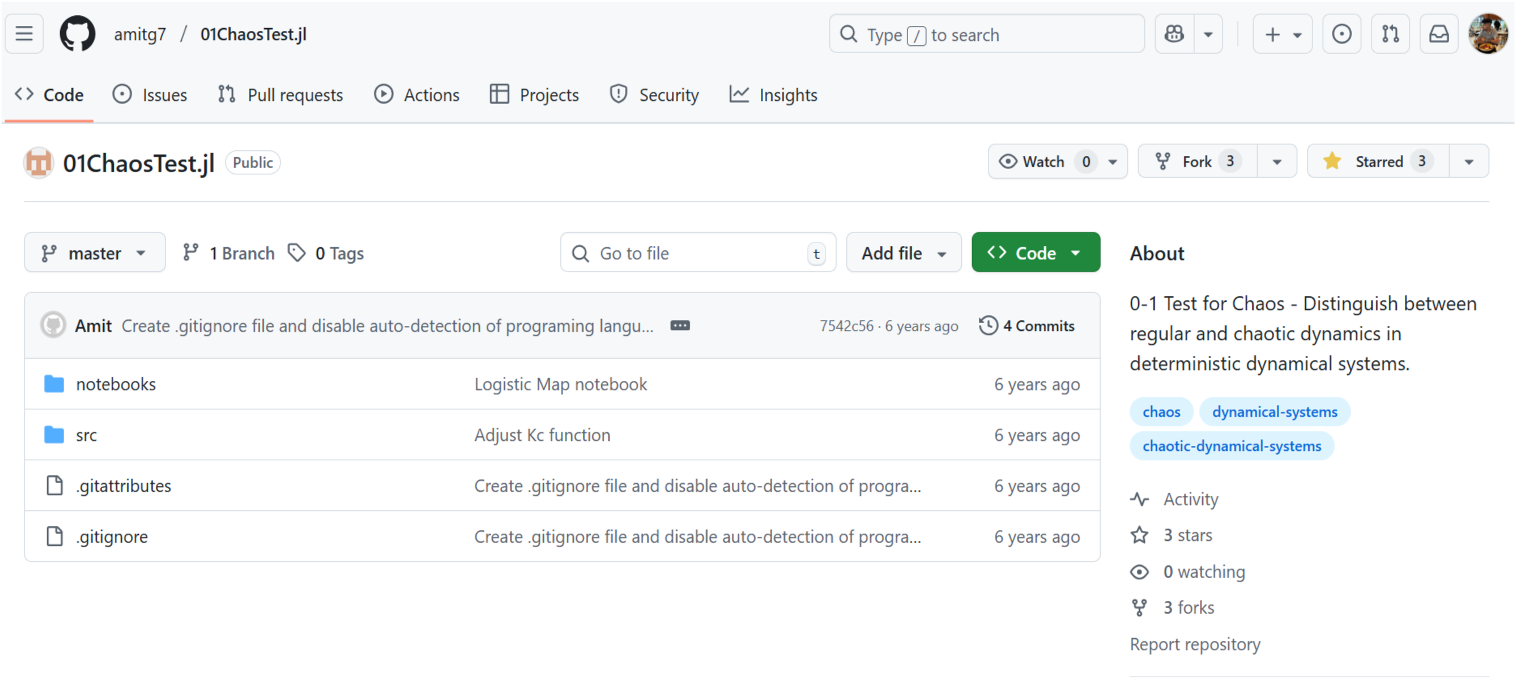
4. ‘K ≈ 0’ indicates regular dynamics and ‘K ≈ 1’ indicates chaos.

A julia package for 0-1 test

1. Translated to .

2. Used \`curve_fit()\` and \`fsolve()\` functions from the \`scipy.optimize\` suite.

3. Used \`pearsonr ()\` function to compute Pearson's correlation coefficient from \`scipy.stats' suite in one of the steps.



Cross-correlation coefficient (many authors)

1. For two time series from nodes 'i' and 'm', Γ is given by

$$\Gamma_{\mathrm{i,m}} = \frac{\langle \tilde{x}_i(n)\tilde{x}_m(n)\rangle}{\sqrt{\langle(\tilde{x}_i(n))^2\rangle\langle(\tilde{x}_m(n))^2\rangle}}$$

2. The average is calculated over time and the variation from the mean is

$$\tilde{x}(n) = x(n) - \langle x(n)\rangle$$

3. $|\Gamma| = 1$ indicates total synchrony and $|\Gamma| < 1$ is asynchrony. Moreover, $\Gamma = 1$ represents in-phase synchrony and $\Gamma = -1$ represents anti-phase synchrony.

```python
## cross-correlation coeff
phi_x1 = np.array(x1_sol[5000:])
phi_x2 = np.array(x2_sol[5000:])

x1_tilde = phi_x1 - np.mean(phi_x1)
x2_tilde = phi_x2 - np.mean(phi_x2)

Numerator = np.mean(x1_tilde*x2_tilde)
Denominator = np.sqrt(np.mean(x1_tilde**2)*np.mean(x2_tilde**2))

cc = Numerator/Denominator
```

Kuramoto's order parameter (Kuramoto and Battogtokh, 2002)

1. Phase of a neuron 'm' is

$$\zeta_m = \tan^{-1}\left(\frac{y_m(t)}{x_m(t)}\right)$$

2. The complex valued Kuramoto index B is then

$$B_m(t) = \exp(i\zeta_m(t)), \qquad i = \sqrt{-1}.$$

3. The index at time 't' is then

$$B(t) = \left| \frac{1}{N} \sum_{m=1}^{N} B_m(t) \right|.$$

4. When B = 1, this means the nodes are all fully coherent and their phases are all locked. Any value B < 1 represents incoherence.

```
## Kuramoto order parameter
l1 = np.arctan(y1_sol/x1_sol)
l2 = np.arctan(y2_sol/x2_sol)

Ind1 = np.exp(1j*l1)
Ind2 = np.exp(1j*l2)
Indt = np.abs(1/2*(Ind1+Ind2))
Kuram = np.mean(Indt)
```

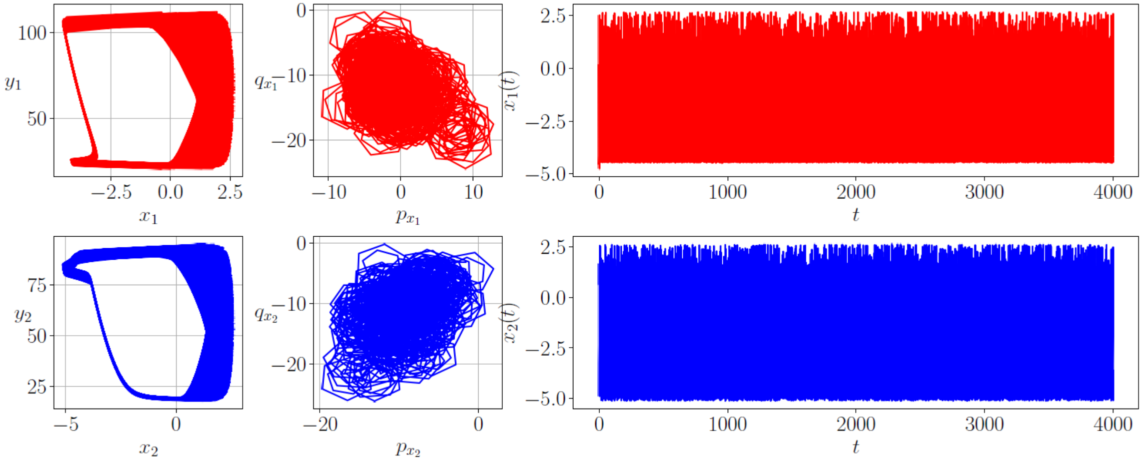
Move to jupyter notebook

Long URL : https://github.com/indrag49/Pycon-Ireland-Tutorial-2025

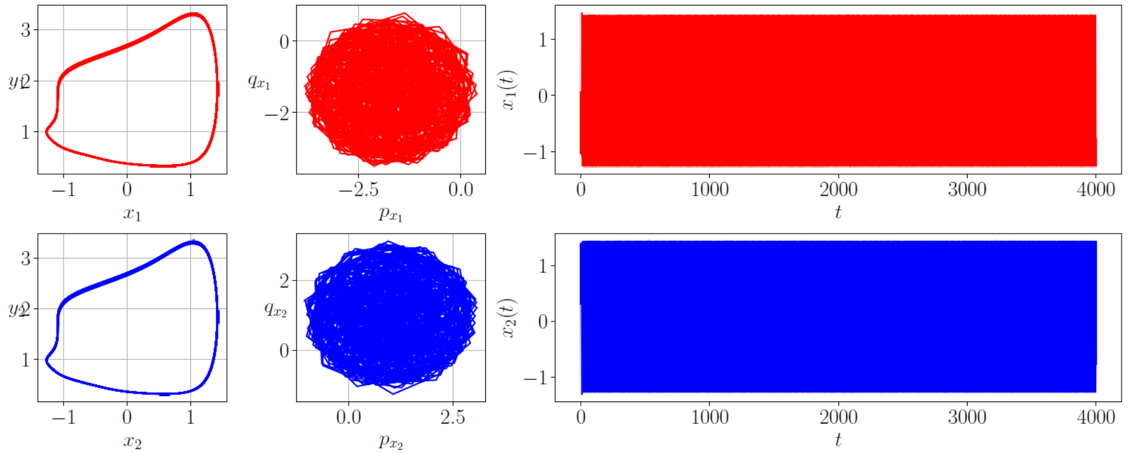
Tiny URL: https://tinyurl.com/2wbj5x8c

Results from gap-junction coupling

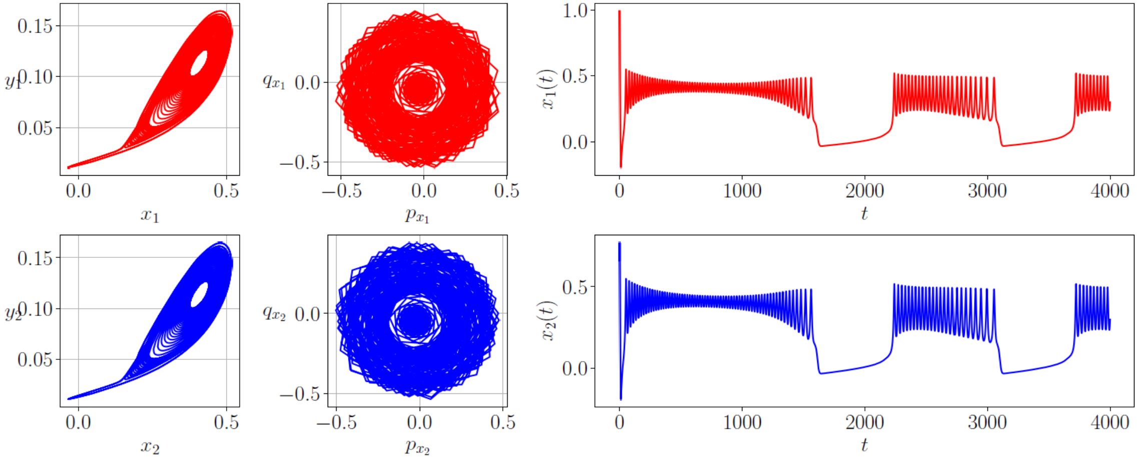
H =0.0682, SE =0.049, K =0.973, Γ =-0.2325, B =0.9448



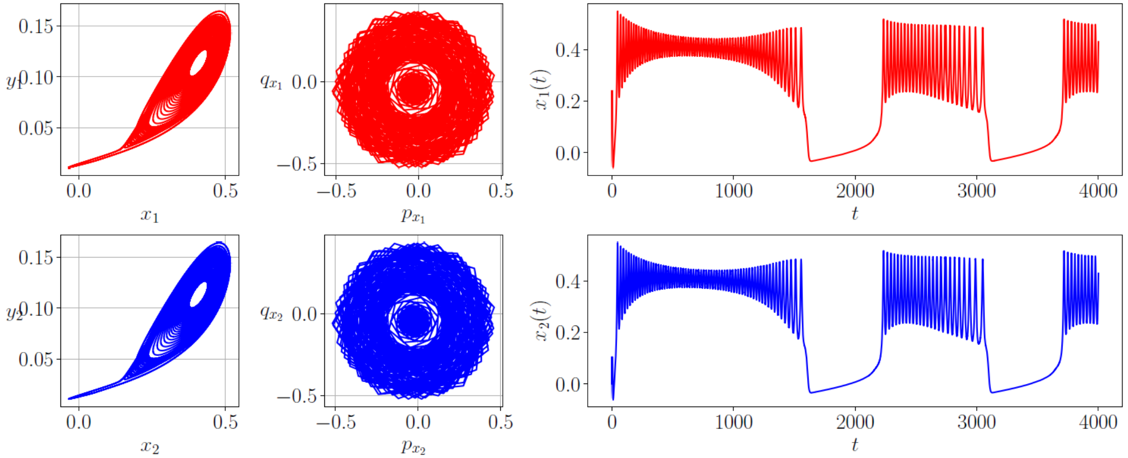
H =0.1827, SE =0.0923, K =0.3195, Γ =-0.7464, B =0.783



H =0.8822, SE =0.0143, K =0.1588, Γ =1.0, B =0.9985

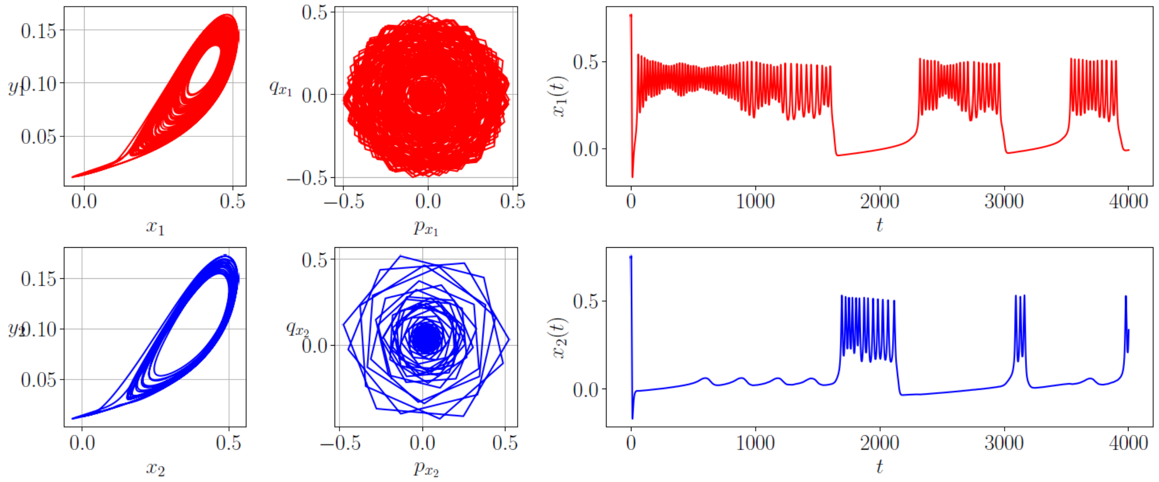


H =0.8826, SE =0.0143, K =0.1594, Γ =1.0, B =0.9998

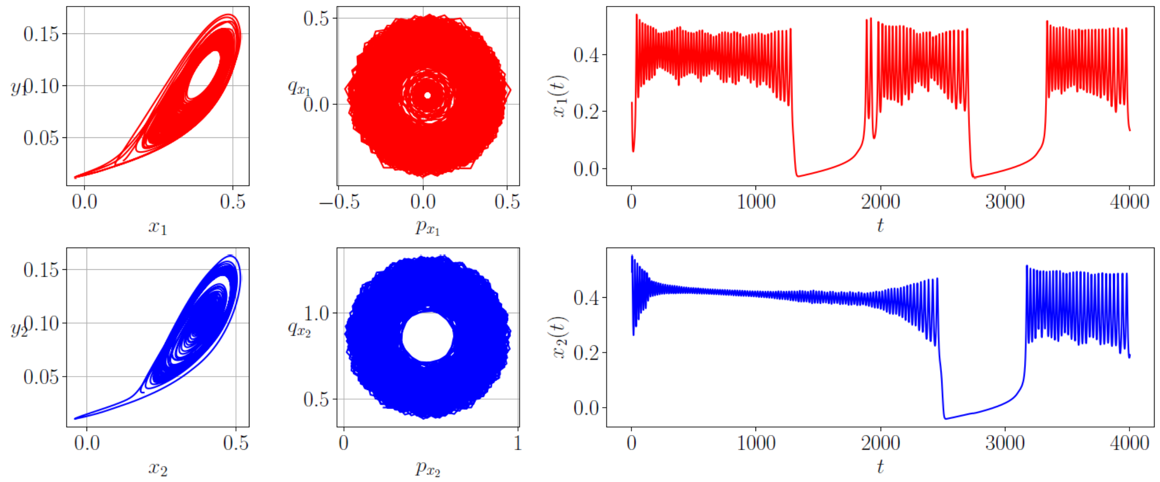


Results from chemical coupling

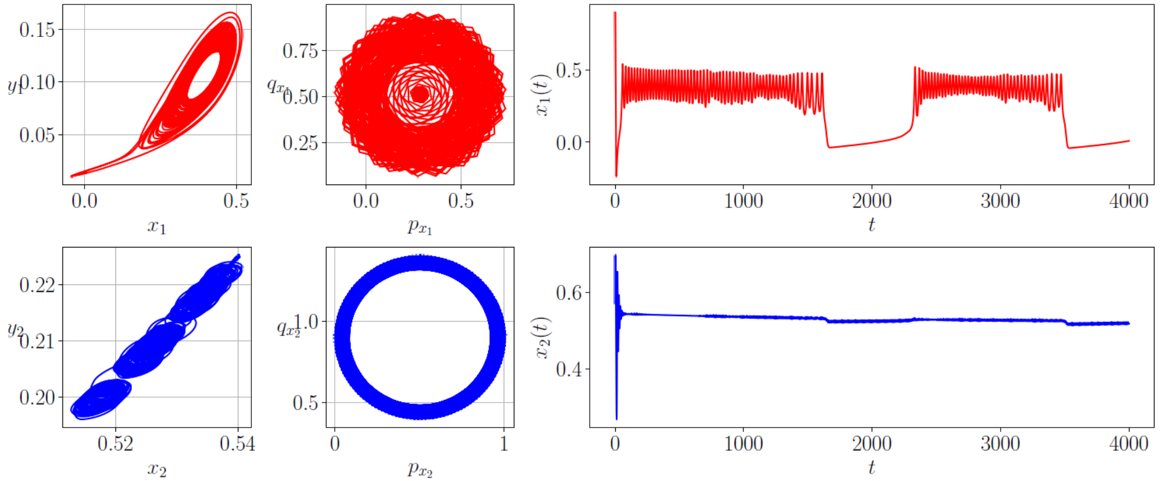
H =0.929, SE =0.0077, K =0.0934, Γ =-0.5153, B =0.9313



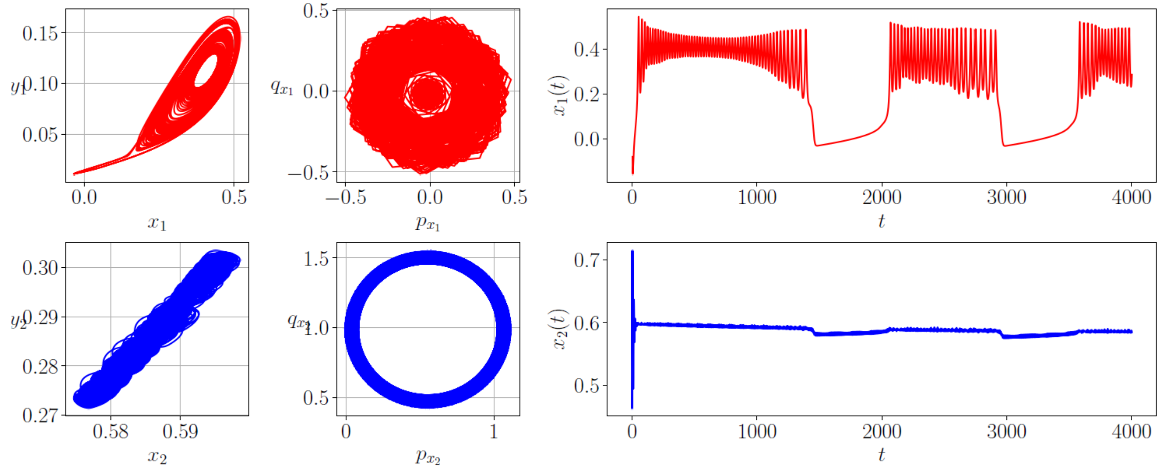
H =0.9142, SE =0.0132, K =0.1093, Γ =0.3314, B =0.9667



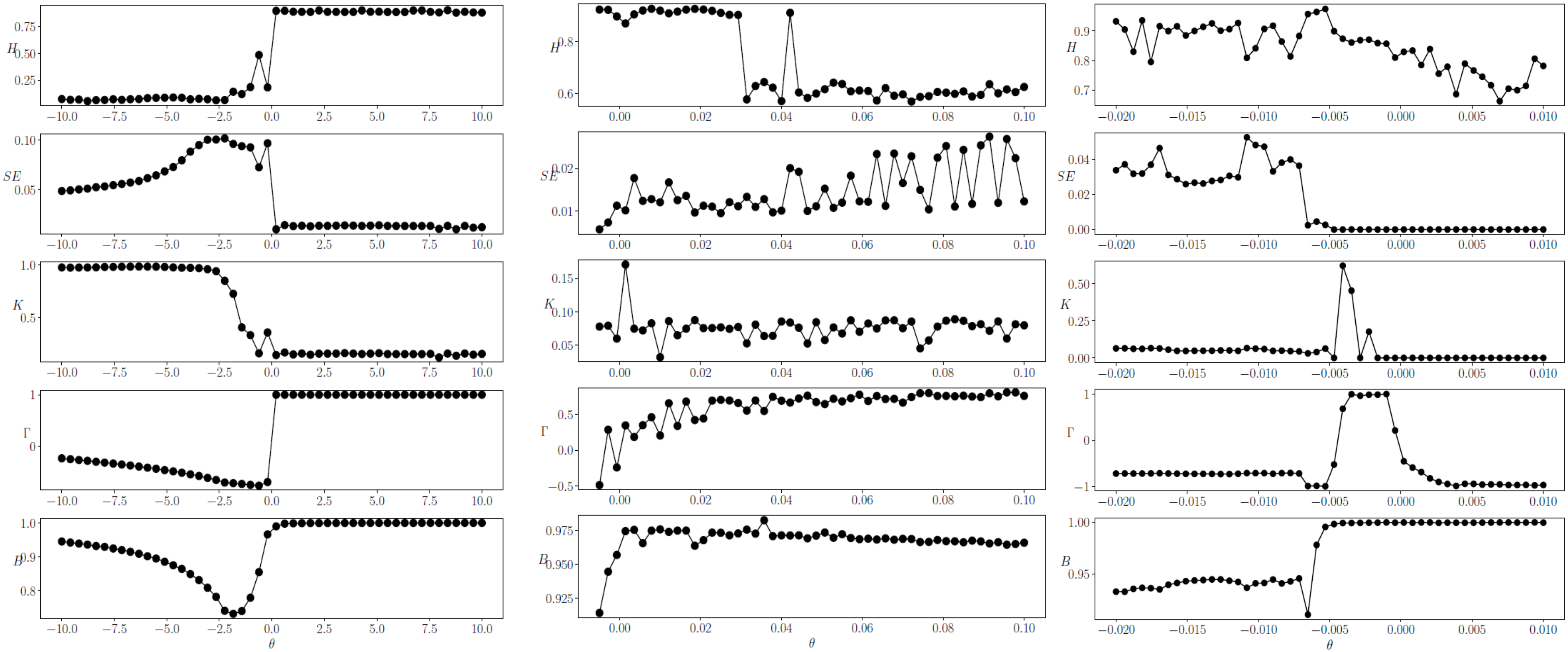
H =0.6073, SE =0.0158, K =0.0803, Γ =0.6919, B =0.9687



H =0.5853, SE =0.027, K =0.0851, Γ =0.7929, B =0.9654



Plots with varying coupling strength



Separate experiment from visualization

1. An extra step for data processing before visualisation.

2. This is where comes handy! And it's worth it.

3. Read "Taming the Chaos of Computational Experiments" by T. G. Kolda for more on this: [siam.org/publications/siam-news/articles/taming-the-chaos-of-computational-experiments/.](https://siam.org/publications/siam-news/articles/taming-the-chaos-of-computational-experiments/)

```python
## Code to create the data files

SS = np.linspace(-10, 10, 50)

count = 1
HH=[]
SE=[]
KKTest = []
CC = []
Kuramoto = []
for theta in SS:
    print("count = "+str(count))
    x1_sol, y1_sol, x2_sol, y2_sol, tt, cc, KK1, KK2, Kuram, h1, h2, se1, se2 = bif_gap(theta)
    HH+=[(h1+h2)/2, ]
    SE+=[(se1+se2)/2, ]
    CC+=[cc, ]
    KKTest+=[(KK1+KK2)/2, ]
    Kuramoto+=[Kuram, ]

    print("H=", (h1+h2)/2)
    print("SE=", (se1+se2)/2)
    print(" ")

    count+=1

df = pd.DataFrame({
    'theta': SS,
    'H': HH,
    'SE': SE,
    'CC': CC,
    'KK': KKTest,
    'Kuramoto': Kuramoto
})

## Create the data file
df.to_csv('data_gap.csv', index=False)
```

Visualization using

In [6]:
```python
## Load the data file
dfGap = pd.read_csv('data_gap.csv')
dfGap
```

Out[6]:

	theta	H	SE	CC	KK	Kuramoto
0	-10.000000	0.075755	0.048965	-0.230950	0.974990	0.945371
1	-9.591837	0.068889	0.049600	-0.247435	0.973426	0.942219
2	-9.183673	0.070847	0.050584	-0.264226	0.974289	0.939167
3	-8.775510	0.056358	0.051361	-0.279553	0.974688	0.936165
4	-8.367347	0.065453	0.052763	-0.297540	0.976047	0.931959
5	-7.959184	0.067204	0.053522	-0.314207	0.979496	0.929517
6	-7.551020	0.073805	0.054817	-0.332759	0.980077	0.924822
7	-7.142857	0.068470	0.056010	-0.350708	0.981813	0.920045
8	-6.734694	0.074268	0.057403	-0.369906	0.982537	0.914924
9	-6.326531	0.075233	0.058941	-0.391002	0.983551	0.909313
10	-5.918367	0.083993	0.061830	-0.412140	0.982661	0.901997

```python
sz=18
%matplotlib notebook
matplotlib.rc('xtick', labelsize=sz)
matplotlib.rc('ytick', labelsize=sz)

SS = np.linspace(-10, 10, 50)

fig, axs = plt.subplots(5,1, figsize=(10, 12))

axs[0].set_ylabel('$H$',rotation=False, fontsize=sz)
axs[1].set_ylabel('$SE$',rotation=False, fontsize=sz)
axs[2].set_ylabel('$K$',rotation=False, fontsize=sz)
axs[3].set_ylabel('$\\Gamma$',rotation=False, fontsize=sz)
axs[4].set_ylabel('$B$',rotation=False, fontsize=sz)
axs[4].set_xlabel('$\\theta$', fontsize=sz)

HH = dfGap['H']
SE = dfGap['SE']
KKTest = dfGap['KK']
CC = dfGap['CC']
Kuramoto = dfGap['Kuramoto']


axs[0].plot(SS, HH, 'ko-', ms=10)
axs[1].plot(SS, SE, 'ko-', ms=10)
axs[2].plot(SS, KKTest, 'ko-', ms=10)
axs[3].plot(SS, CC, 'ko-', ms=10)
axs[4].plot(SS, Kuramoto, 'ko-', ms=10)

plt.tight_layout()
```

A shiny app (indrag49.shinyapps.io/TimeSeriesNeuronR)

Choose coupling type:

Josephson Junction

θ (coupling strength):

-5

A:

0.0041

α:

5.276

γ:

0.315

ε:

0.0005

Ic:

3

μ:

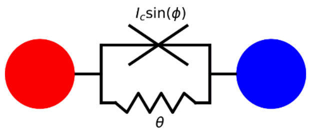
3

Run Simulation

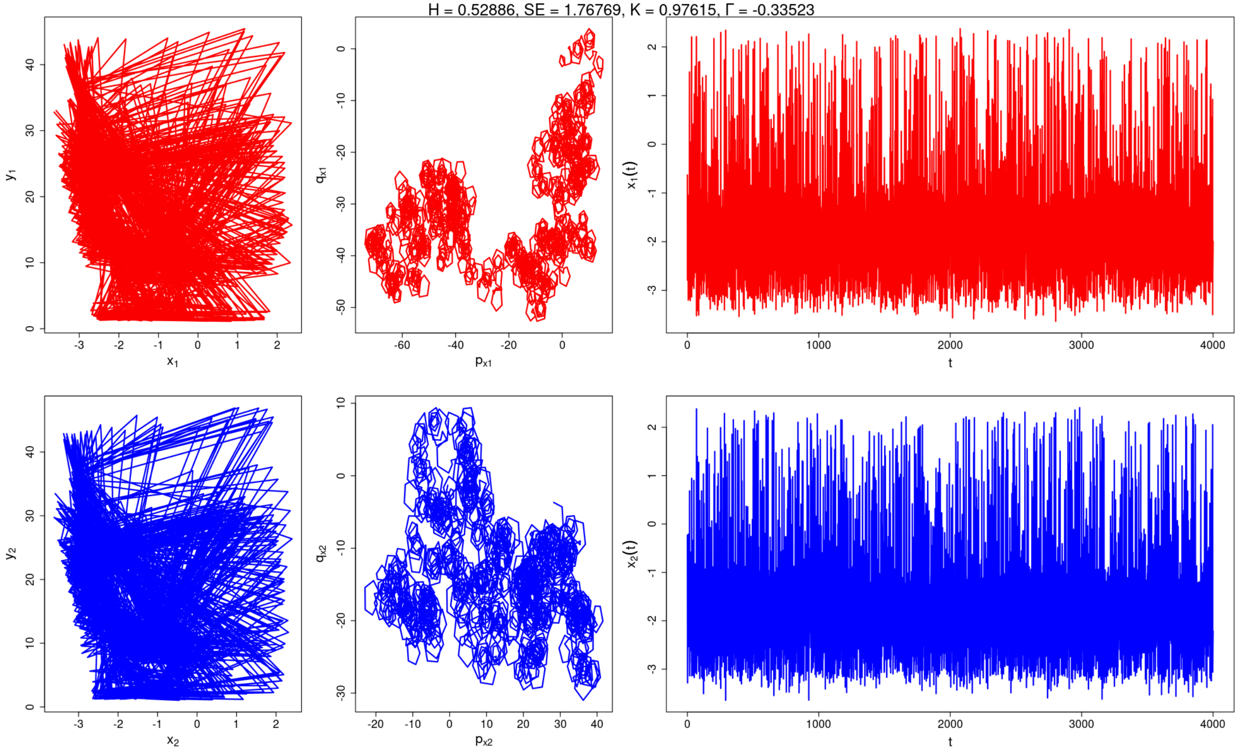
Note: Adjust parameters and click 'Run Simulation' to update plots.

Model Equations:

$$
\begin{aligned}
\dot{x}_1 &= x_1^2(1-x_1) - y_1 + I_1 - I_c \sin(\phi) + \theta(x_2 - x_1),\\
\dot{y}_1 &= Ae^{\alpha x_1} - \gamma y_1,\\
\dot{I}_1 &= \varepsilon\left[\frac{1}{60}\left(1+\tanh\left(\frac{0.05-x_1}{0.001}\right)\right) - I_1\right],\\
\dot{x}_2 &= x_2^2(1-x_2) - y_2 + I_2 + I_c \sin(\phi) + \theta(x_1 - x_2),\\
\dot{y}_2 &= Ae^{\alpha x_2} - \gamma y_2,\\
\dot{I}_2 &= \varepsilon\left[\frac{1}{60}\left(1+\tanh\left(\frac{0.05-x_2}{0.001}\right)\right) - I_2\right],\\
\dot{\phi} &= \mu(x_1 - x_2).
\end{aligned}
$$



H = 0.52886, SE = 1.76769, K = 0.97615, Γ = -0.33523



Summary

1. Coupling induces ‘chaos’ in the inhibitory regime.

2. Excitatory coupling and its more positive values drive the coupled system into exhibiting bursting. Also, both neurons synchronize.

3. In electromagnetic coupling, excitatory coupling drives the system to decay oscillation, falling into a symmetric equilibrium point.

4. Future work: move to GPU accelerated framework using `CuPy`:

5. Future work: use a complex network of neurons and integrate `NetworkX`:

Acknowledgements


MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA
UNIVERSITY OF NEW ZEALAND

AUT
UNIVERSITY

UCD DUBLIN
University College Dublin

DIGITAL UNIVERSITY KERALA
Curating a responsible digital world

Reference: Indranil Ghosh, Hammed Olawale Fatoyinbo, and Sishu Shankar Muni, "Time series analysis of coupled slow-fast neuron models: From Hurst exponent to Granger causality" (2025), arxiv.org/abs/2507.13570.

repository: github.com/indrag49/TS-SlowFast-dML

Bonus slide (Network of neurons)

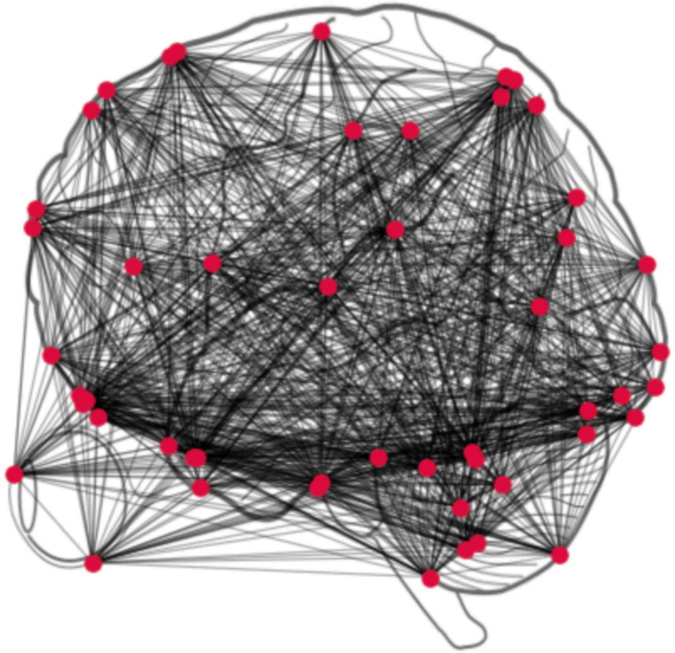


Fig. Brain schematic (wikipedia)

```python
import networkx as nx

# Parameters
N = 50
p = 0.5
strength = .05
Tmax = 8000
dt = 0.01

## dML Parameters
A = 0.0041
alpha=5.276
gamma = 0.315
epsilon = 0.0005

# Random Erdos-Renyi graph
G = nx.erdos_renyi_graph(N, p, seed=42)
Adj = nx.to_numpy_array(G)                    # adjacency matrix

# main function
def dMLNetworkCoupled(t, XX):
    XX = XX.reshape(N, 3)
    dXXdt = np.zeros_like(XX)

    for i in range(N):
        x, y, I = XX[i]

        # dML on a single neuron
        dx = x**2 * (1 - x) - y + I
        dy = A * np.exp(alpha * x) - gamma * y
        dI = epsilon*(1/60*(1+np.tanh((0.05-x)/0.001)) - I)

        # diffusive coupling
        couplingTerm = strength*np.sum(Adj[i, :] * (XX[:, 0] - x))
        dx += couplingTerm

        dXXdt[i] = np.array([dx, dy, dI])

    return dYdt.flatten()
```