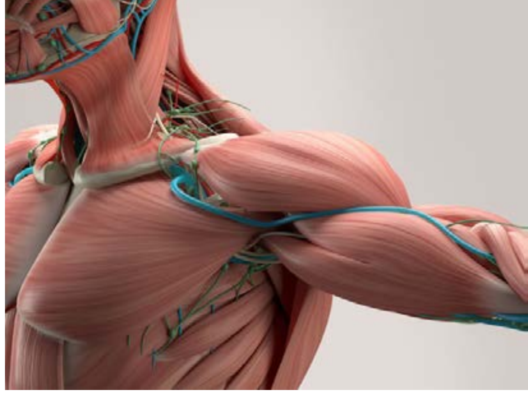


Day 2:
Modeling of Oscillator Dynamics

2018. 12. 14. (Fri.)

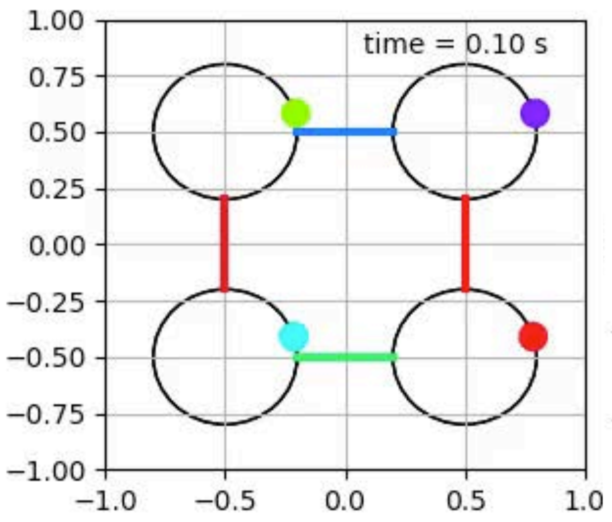
Dai Owaki, Ph. D.,

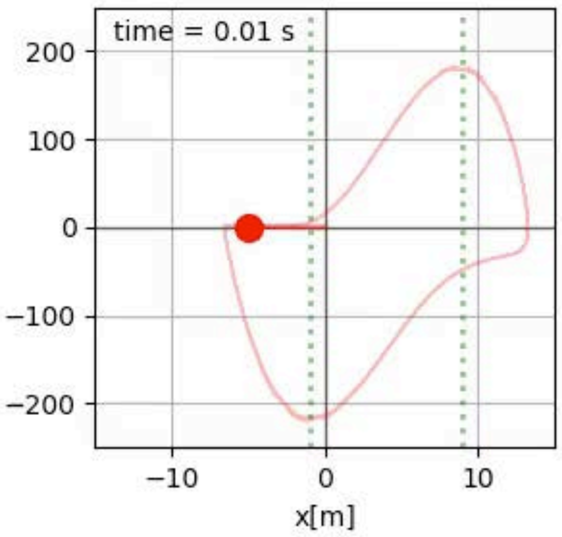
Assist. Prof. Neuro-robotics Lab. (Hayashibe Lab.),

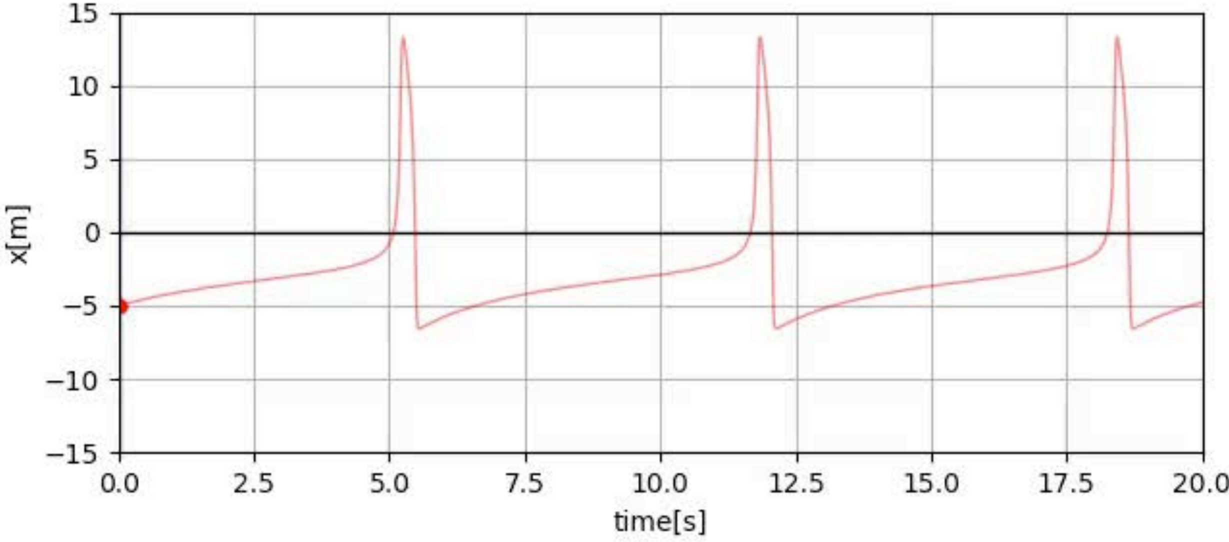
Dept. Robotics, Graduate School of Engineering,

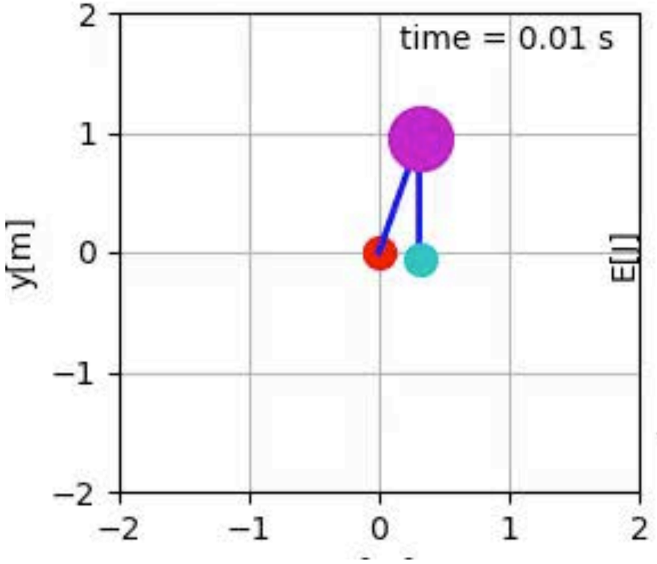
Tohoku University, Japan

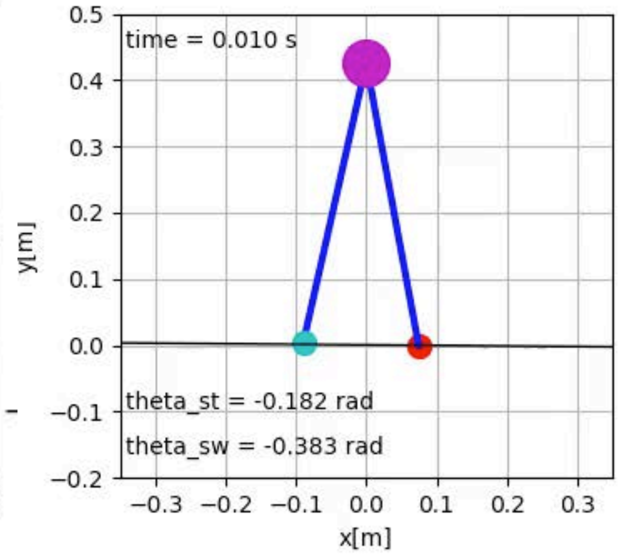
The Goals: Enjoy “Dynamics” with Python

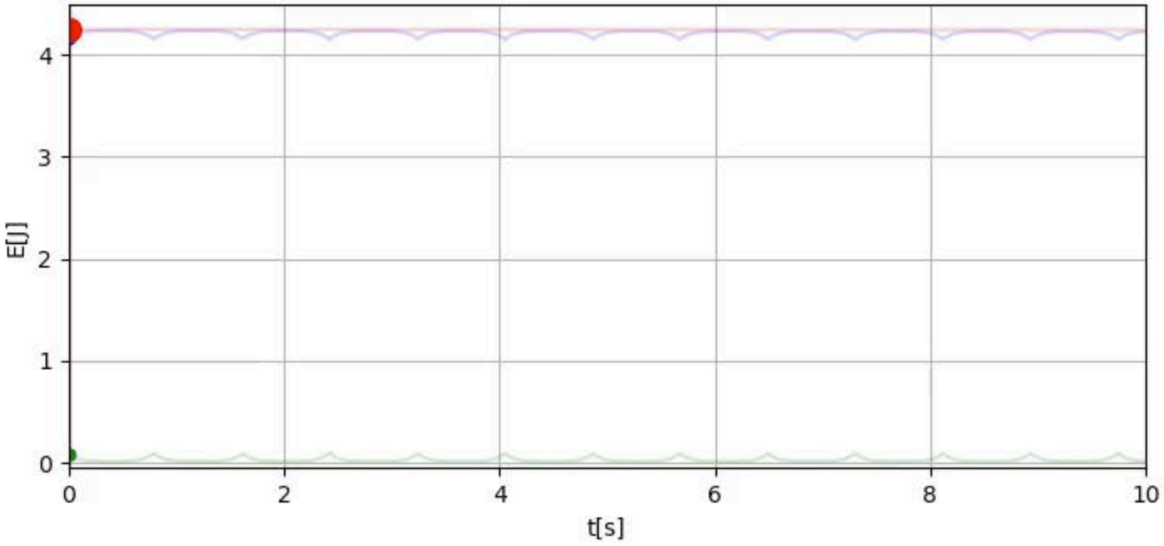












Schedule

2. Modeling of Oscillator Dynamics (12/14 Fri.)

3. Lagrangian and mechanical system dynamics (12/21Fri.)

4. TBA (Manipulator control/passive walker) (1/11 Fri.)

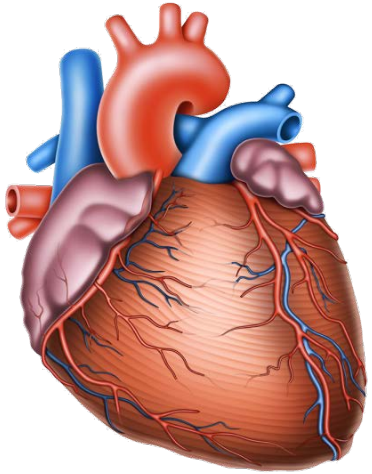
Topics: Modeling of Oscillator Dynamics

I. Overview and introduction of oscillation in natures

II. Modeling of simple oscillator dynamics

III. Modeling of neuron-like oscillator dynamics

Overview: We found “Oscillation” anywhere

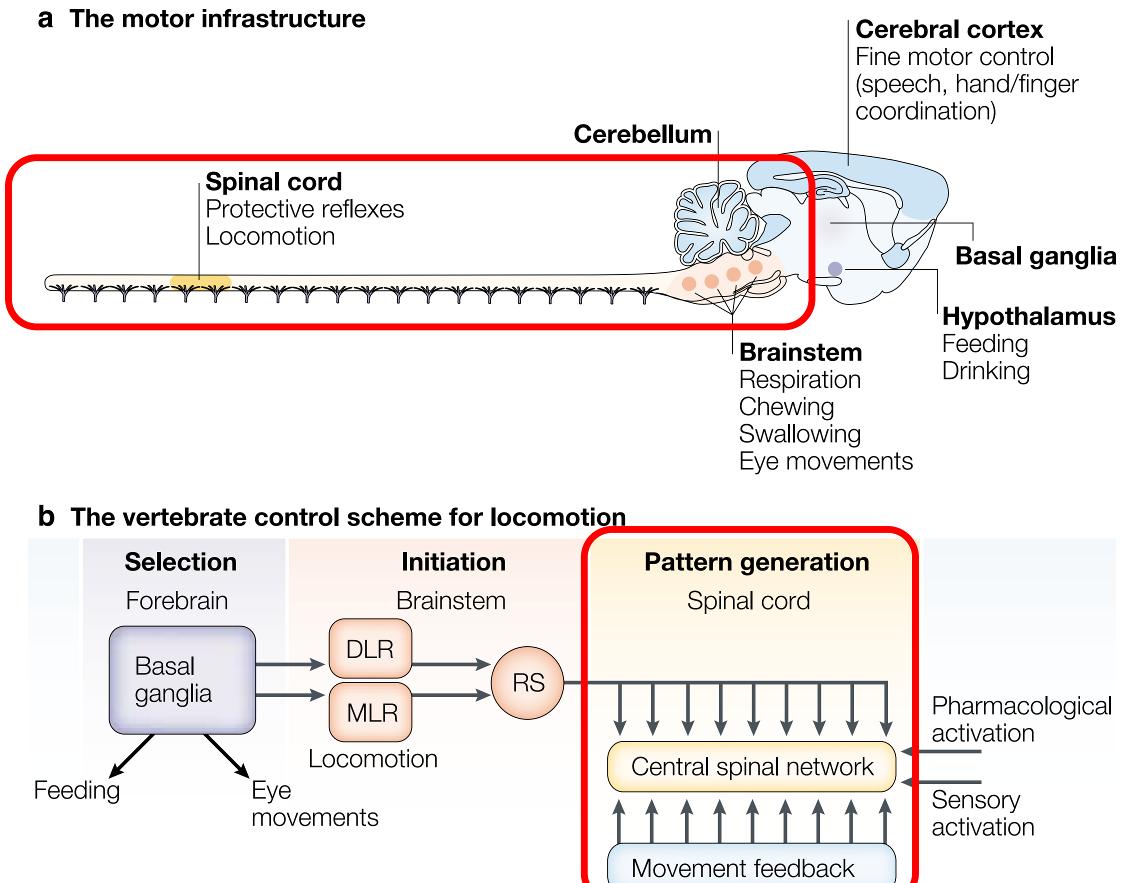


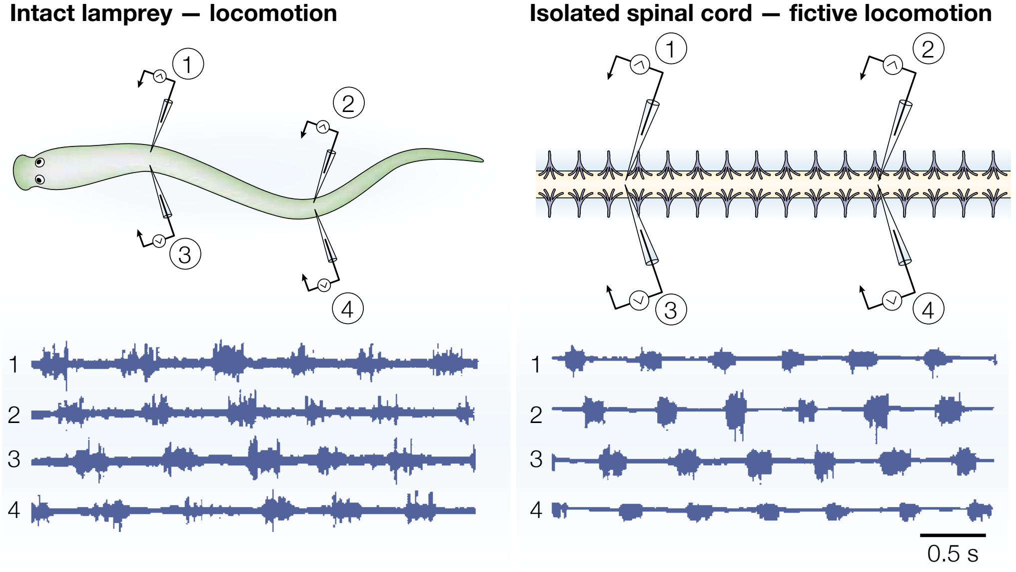
Biological oscillation

Biological clock

Locomotion Control in Vertebrates

The (vertebrate) locomotor system is organized such that the spinal CPGs are responsible for producing the basic rhythmic patterns, and that higher-level centers (the motor cortex, cerebellum, and basal ganglia) are responsible for modulating these patterns according to environmental conditions.



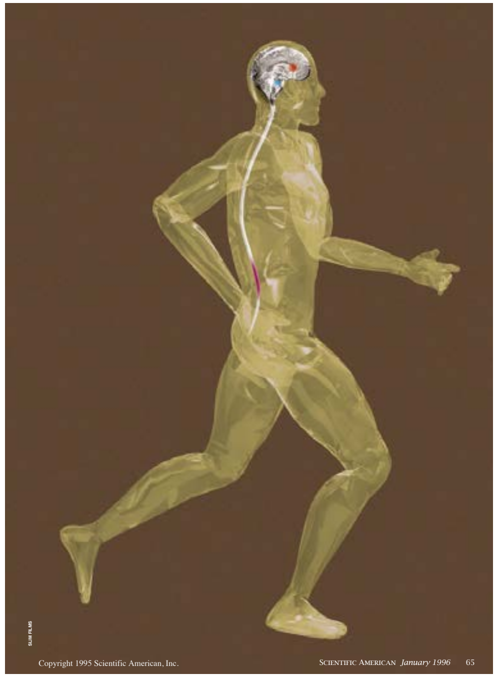


One can extract and isolate from the body the spinal cord of the lamprey (a primitive fish), and it will produce patterns of activity (fictive locomotion), which are very similar to intact locomotion when activated by simple electrical or chemical stimulation (Cohen & Wallen, 1980; Grillner, 1985)

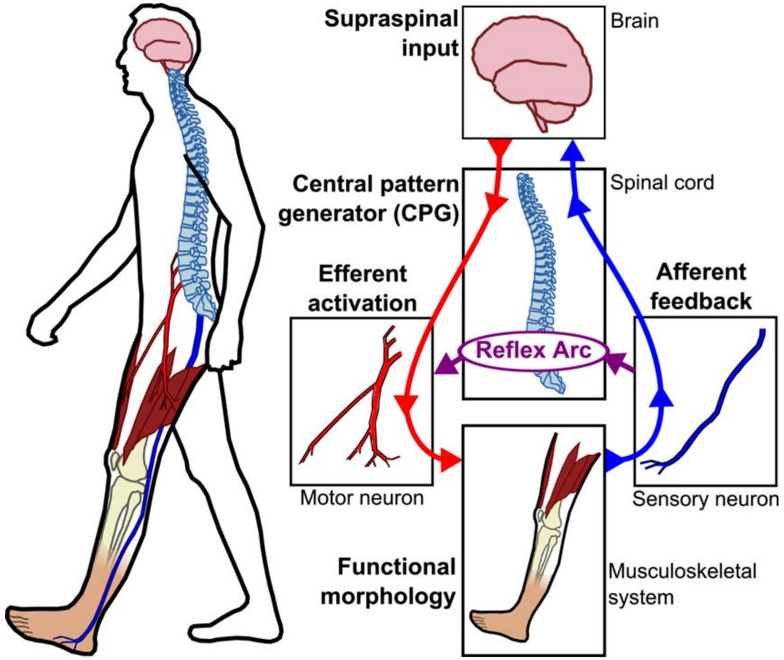
S. Grillner (2003) *Nature reviews* 4, 573-583.

What is CPG*?

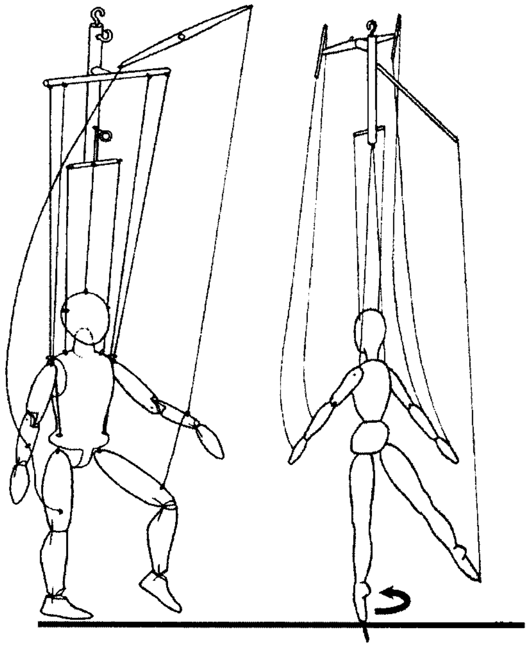
Locomotor central pattern generators (CPGs)
= neural circuits capable of producing coordinated patterns of high-dimensional rhythmic output signals while receiving only simple, low-dimensional, input signals



S. Grillner (1996)



http://russellfox.info

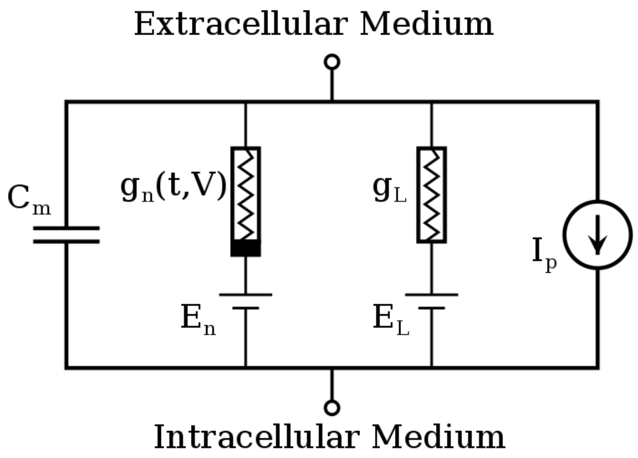


G. E. Loeb (2001)

*A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: A review ", Neural Networks 21 (2008) 642–653.

Detailed Neuron Models

Detailed biophysical models are constructed based on the Hodgkin–Huxley* type of neuron models, which is neuron models that compute how ion pumps and ion channels influence membrane potentials and the generation of action potentials.



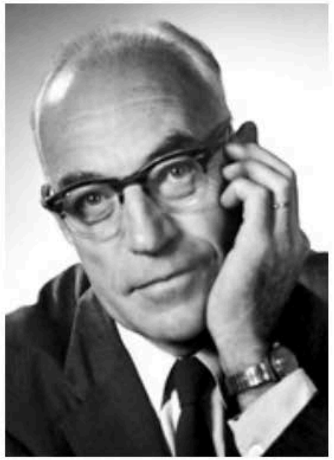
$$I = C_m \frac{\mathrm{d}V_m}{\mathrm{d}t} + \bar{g}_K n^4 (V_m - V_K) + \bar{g}_{Na} m^3 h (V_m - V_{Na}) + \bar{g}_l (V_m - V_l),$$

$$\frac{dn}{dt} = \alpha_n(V_m)(1-n) - \beta_n(V_m)n$$

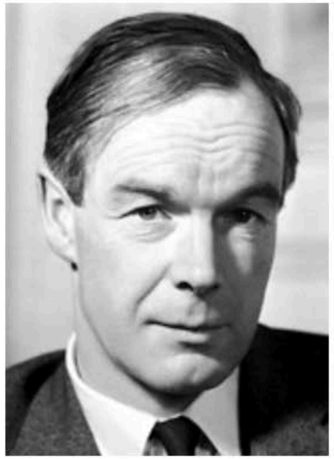
$$\frac{dm}{dt} = \alpha_m(V_m)(1-m) - \beta_m(V_m)m$$

$$\frac{dh}{dt} = \alpha_h(V_m)(1-h) - \beta_h(V_m)h$$

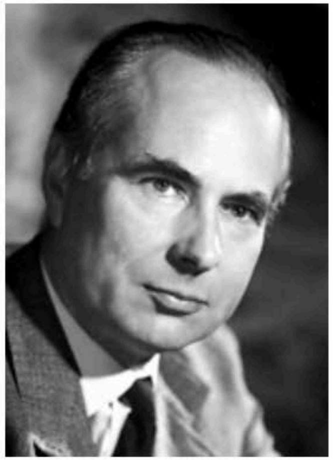
The Nobel Prize in Physiology or Medicine 1963



Sir John Carew Eccles
Prize share: 1/3



Alan Lloyd Hodgkin
Prize share: 1/3

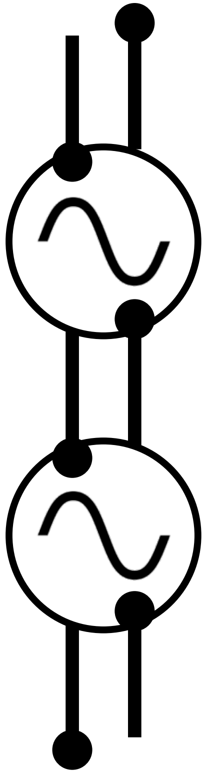


Andrew Fielding Huxley
Prize share: 1/3

Abstracted Oscillator Models

Mathematical models of coupled nonlinear oscillators to study population dynamics (Cohen, Holmes, & Rand, 1982; Collins & Richmond, 1994; Ijspeert, Crespi, Ryczko, & Cabelguen, 2007; Kopell, Ermentrout, & Williams, 1991; Matsuoka, 1987; Schoner, Jiang, & Kelso, 1990).

How inter-oscillator couplings and differences of intrinsic frequencies affect the synchronization and the phase lags within a population of oscillatory centers?



The dynamics of populations of oscillatory centers depend mainly on the type and topology of couplings rather than on the local mechanisms of rhythm generation, something that is well established in dynamical systems theory (Golubitsky & Stewart, 2002; Kuramoto, 2003)

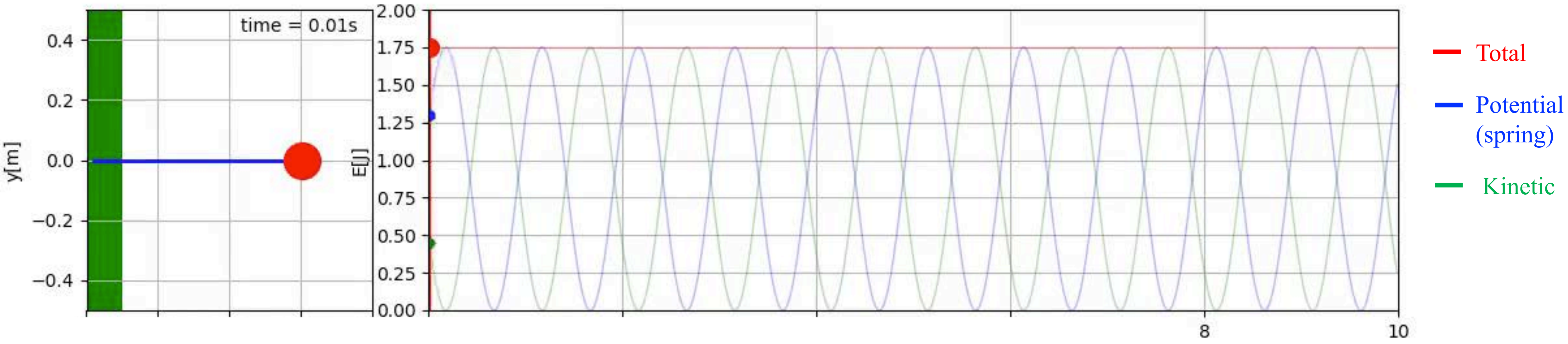
Topics: Modeling of Oscillator Dynamics

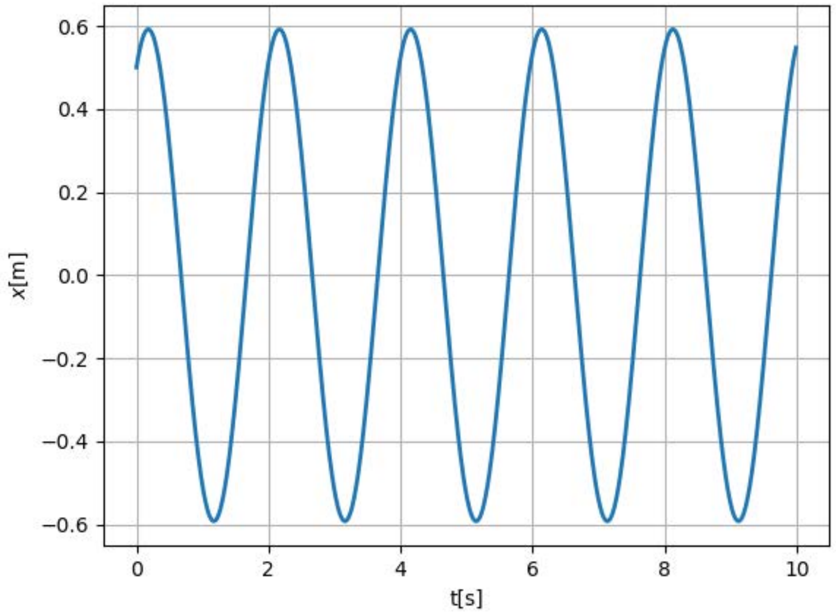
I. Overview and introduction of oscillation in natures

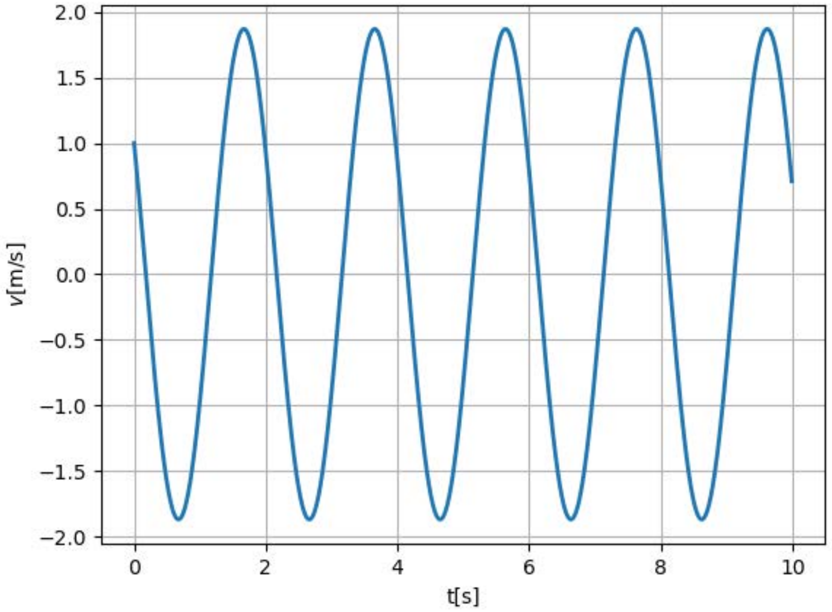
II. Modeling of simple oscillator dynamics

III. Modeling of neuron-like oscillator dynamics

Before Modeling…







- Display movements and energy conservation
- Output *.png files (plot_graph.py)

Python script #0 mass_spring_odeint.py

```python
#!/usr/bin/env python3

# mass_spring_odeint.py
# Copyright (c) 2017 Dai Owaki <owaki@tohoku.ac.jp>
# Revised 2018 by Dai Owaki

from scipy.integrate import odeint

# import original modules
import plot_graph as pg
import video as v

m = 1.0       # mass [kg]
k = 10.0       # spring constant [N/m]
g = 9.8      # gravitational accelaration[m/s^2]

params = [m, k, g] # parameters

def controlinput(x):
    return 0.0

def MassSpring(p, t):
    x = p[0]
    dx = p[1]

    F = controlinput(p)

    ddx = ((-k*x)/m) + F/m

    return [dx,ddx]


# initial conditions(x0, dx0)
max_t = 10.0 # max_time [s]
dt = 0.01    # dt [s]

if __name__ == '__main__':

    t = v.np.arange(0.0, max_t, dt) # time seeies 0.0 to max_t (with dt intervals)
    x0 = [0.5, 1.0]                 # initial variables x0=0.5, x1=1.0
    p = odeint(MassSpring, x0, t)   # ode calculation

    label = [r'$x$[m]', r'$v$[m/s]']
    pg.plot(t, p[:, :2], label)
    v.video(p, dt, max_t, params)
```

Original modules

plot_graph.py: make graph in *.png file

video.py: make animation (given script should be moved/copied to the same directory)

You can download *.zip file in the following URL:
www.oscillex.org/lecture

$ python mass_spring_odeint.py

www.oscillex.org/lecture

Exploring Neuro-robotics

Dai Owaki

home | research | publication | robots | biography | contact

lecture

HOME » lecture

Computational Motor Control and Learning

day2

0_Spring_Mass_System

- plot_graph.py
- video.py

-> mass_spring.zip

1_One_Oscillator

- video_one_oscillator.py

2_Two_Oscillators

- video_two_oscillators.py

3_Quad_Oscillators

- video_quad_oscillators.py

4_KYS_Oscillator

- video_KYS.py

Topics: Modeling of Oscillator Dynamics

I. Overview and introduction of oscillation in natures

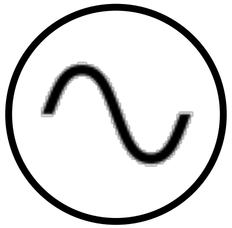
II. Modeling of simple oscillator dynamics

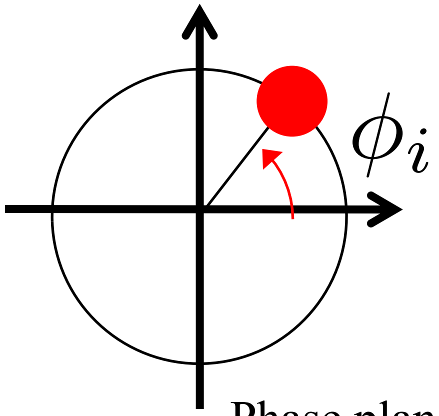
III. Modeling of neuron-like oscillator dynamics

IV. Summary

A Phase Oscillator

$$\dot{\phi}_i = \omega$$

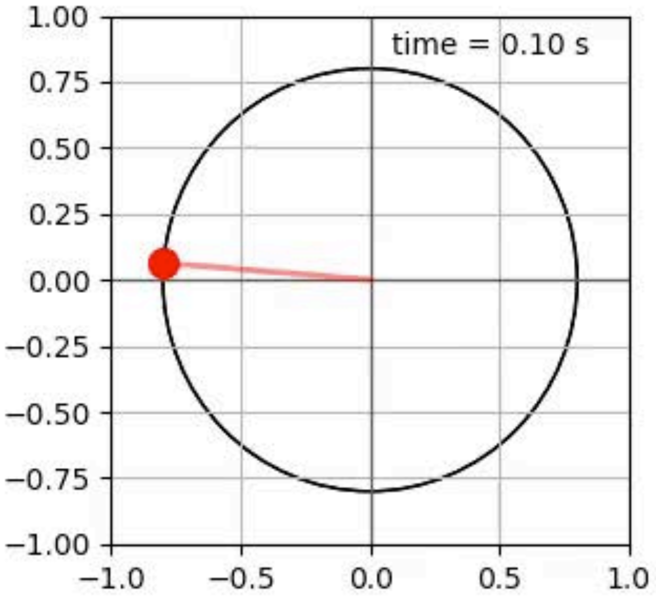


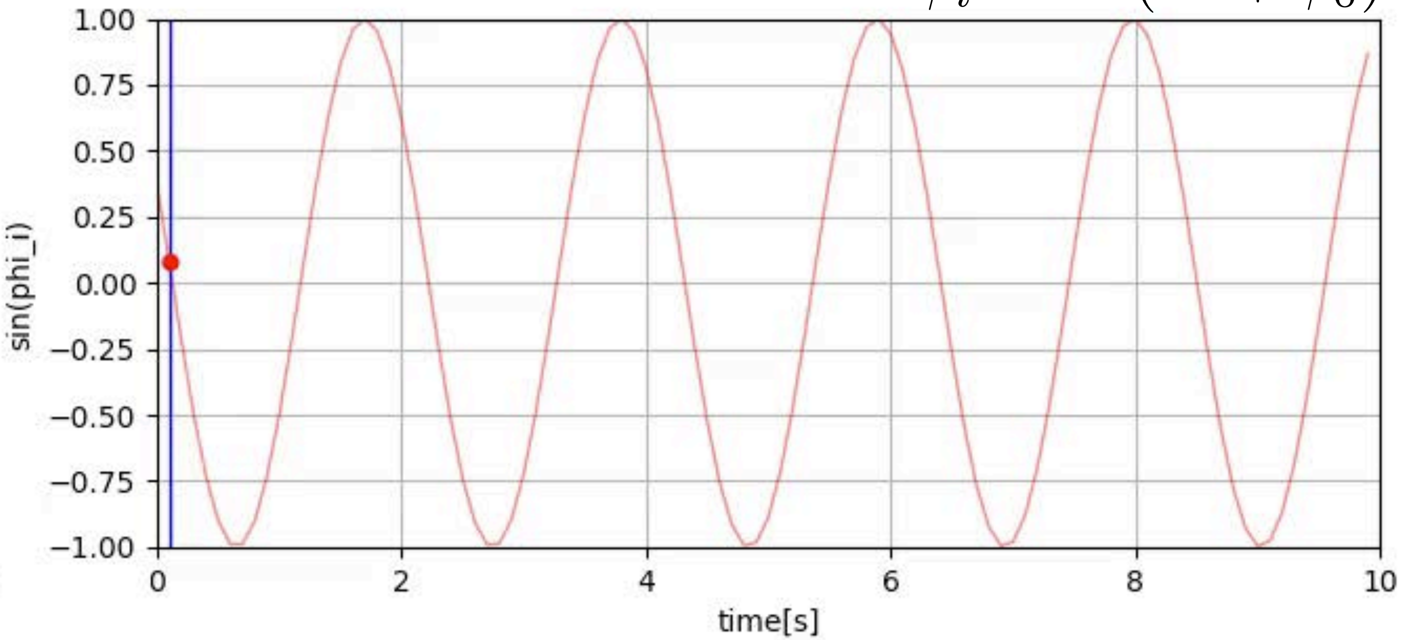


Easily integrate

$$\phi_i = \omega t + \phi_0$$

$$\sin \phi_i = \sin(\omega t + \phi_0)$$





Python script #1

one_oscillator_odeint.py

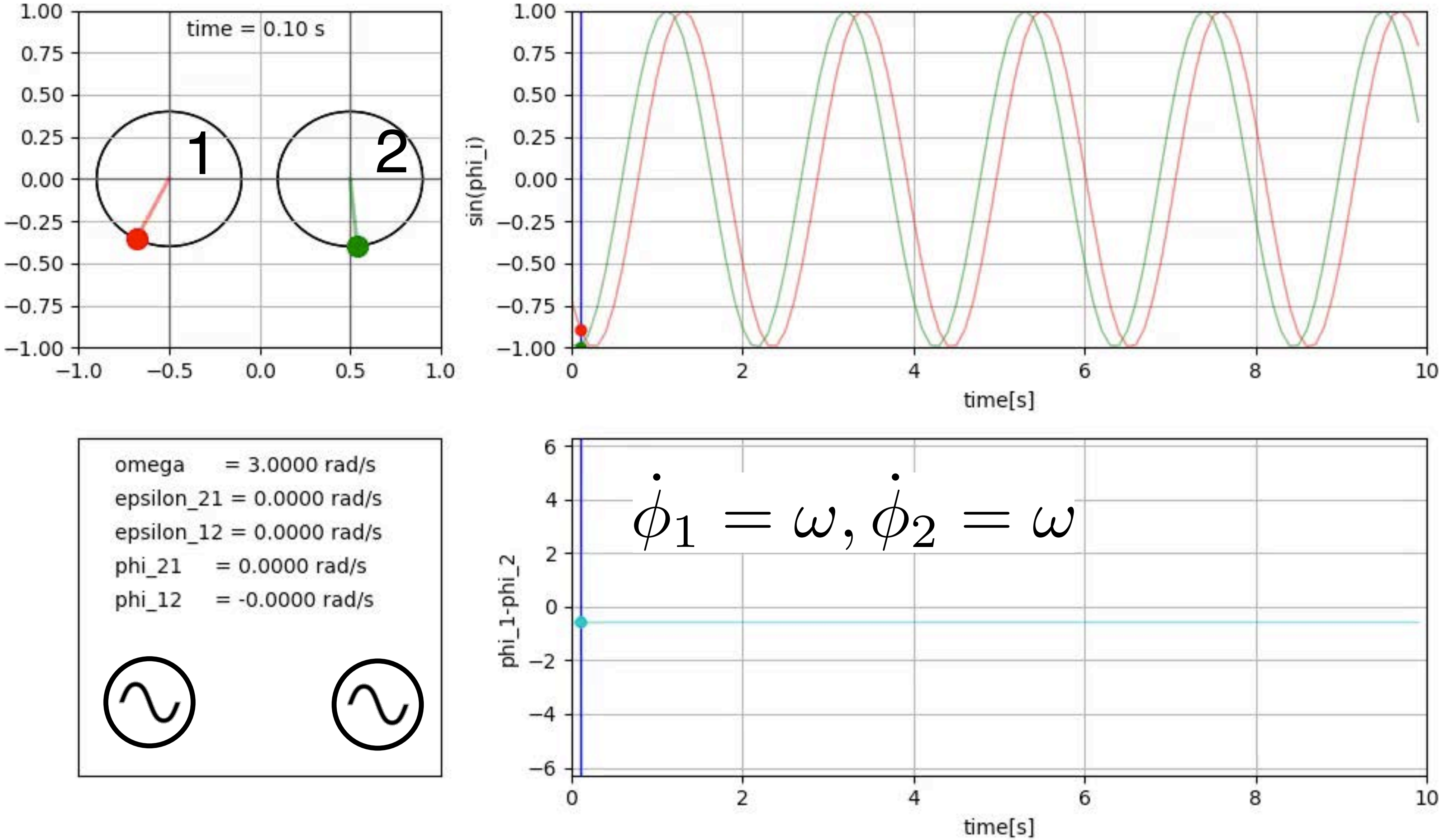

```
from scipy.integrate import odeint
import numpy as np
import random as r # make random numbers

# import video animation modules (original)
import video_one_oscillator as voo

omega       = 3.0        # omega [rad/s]
params = [omega] # parameters

# initial conditions(x0, dx0)
max_t = 10.0 # max_time [s]
dt = 0.1    # dt [s]

# method for oscillator dynamics
def PhaseOscillators(p, t):
    phi  = p[0] # phase
    dphi = p[1] # derivative of phase

    dphi = omega
    ddphi = 0.0

    return [dphi, ddphi]

#Main simulation for oscillator dynamics
t = np.arange(0.0, max_t, dt) # time series data

x0 = [2.0*(r.random())*np.pi, 0.0]  # initial value of phi and dphi
p = odeint(PhaseOscillators, x0, t) # integration by using odeint

voo.video(p, dt, max_t, params) # make animation
```

video_one_oscillator.py: make animation (given script should be moved/copied to the same directory)

$ python one_oscillator_odeint.py

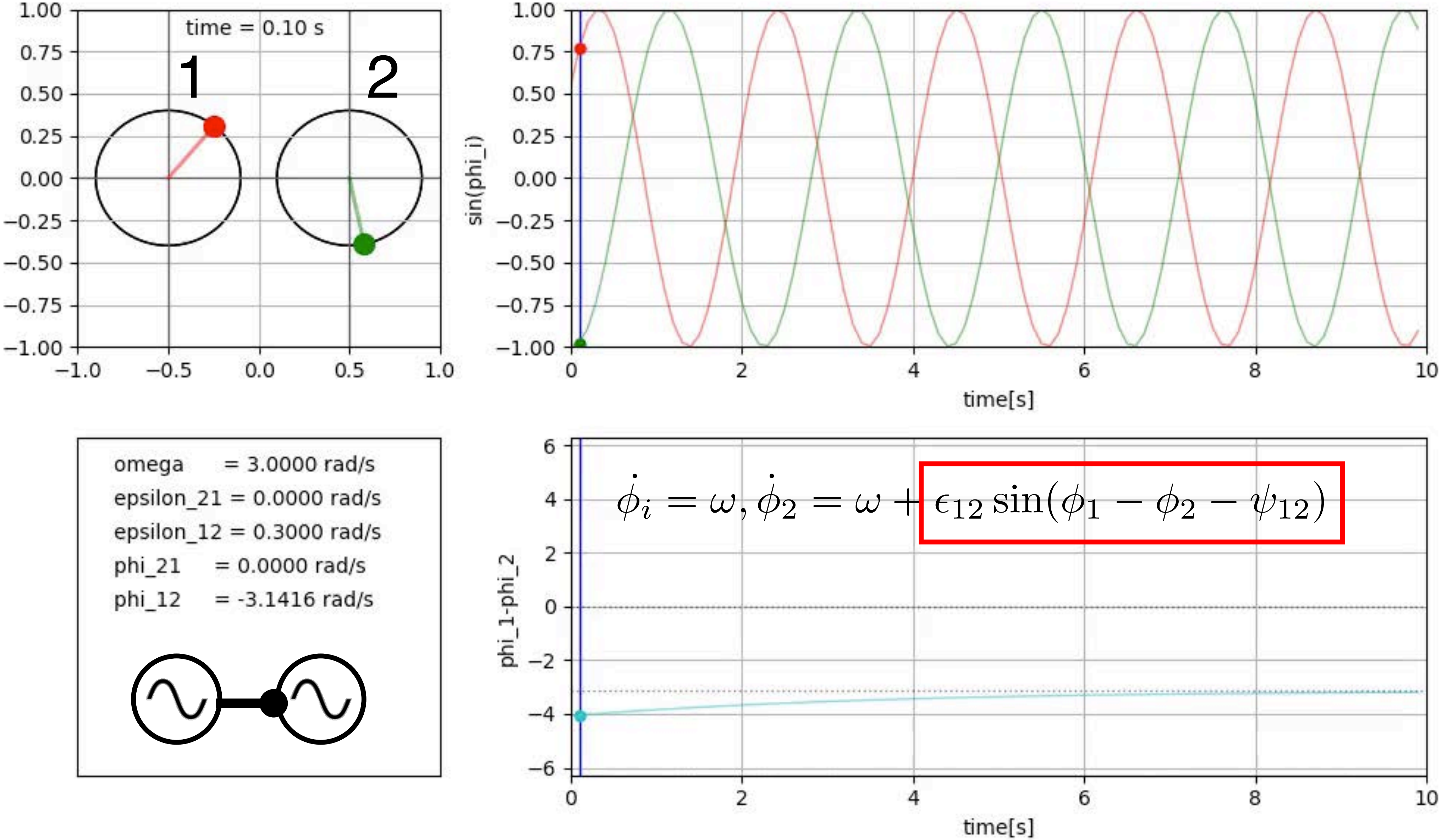
Synchronization between Oscillators

Uncoupled Oscillators

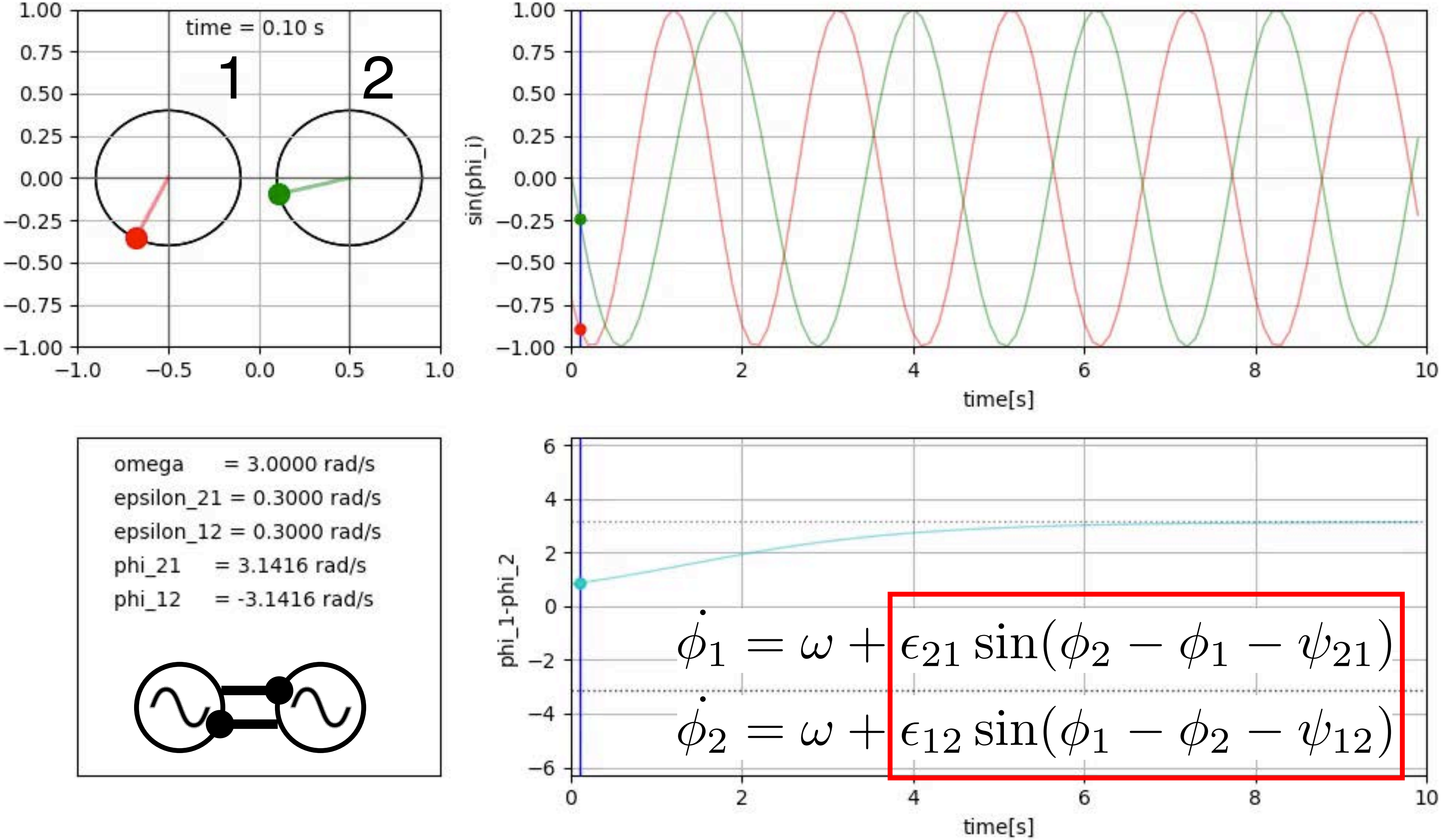


$$\dot{\phi}_1 = \omega, \dot{\phi}_2 = \omega$$

Pacemaker & Follower



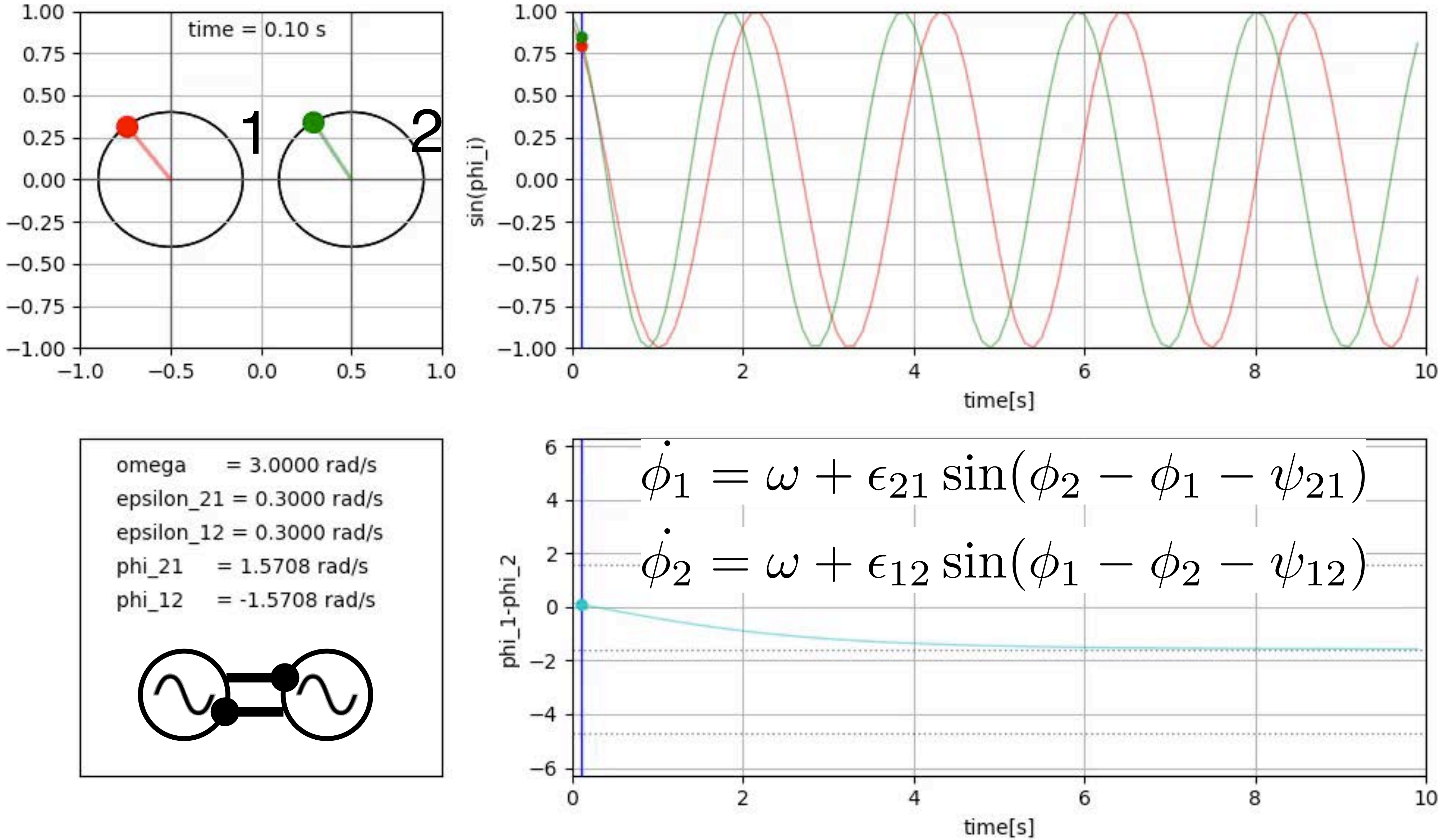
Mutual Interaction for Anti-phase



$$\dot{\phi}_1 = \omega + \boxed{\epsilon_{21} \sin(\phi_2 - \phi_1 - \psi_{21})}$$

$$\dot{\phi}_2 = \omega + \boxed{\epsilon_{12} \sin(\phi_1 - \phi_2 - \psi_{12})}$$

Mutual Interaction for a phase lag



$$\dot{\phi}_1 = \omega + \epsilon_{21} \sin(\phi_2 - \phi_1 - \psi_{21})$$

$$\dot{\phi}_2 = \omega + \epsilon_{12} \sin(\phi_1 - \phi_2 - \psi_{12})$$

Python script #2 two_oscillators_odeint.py

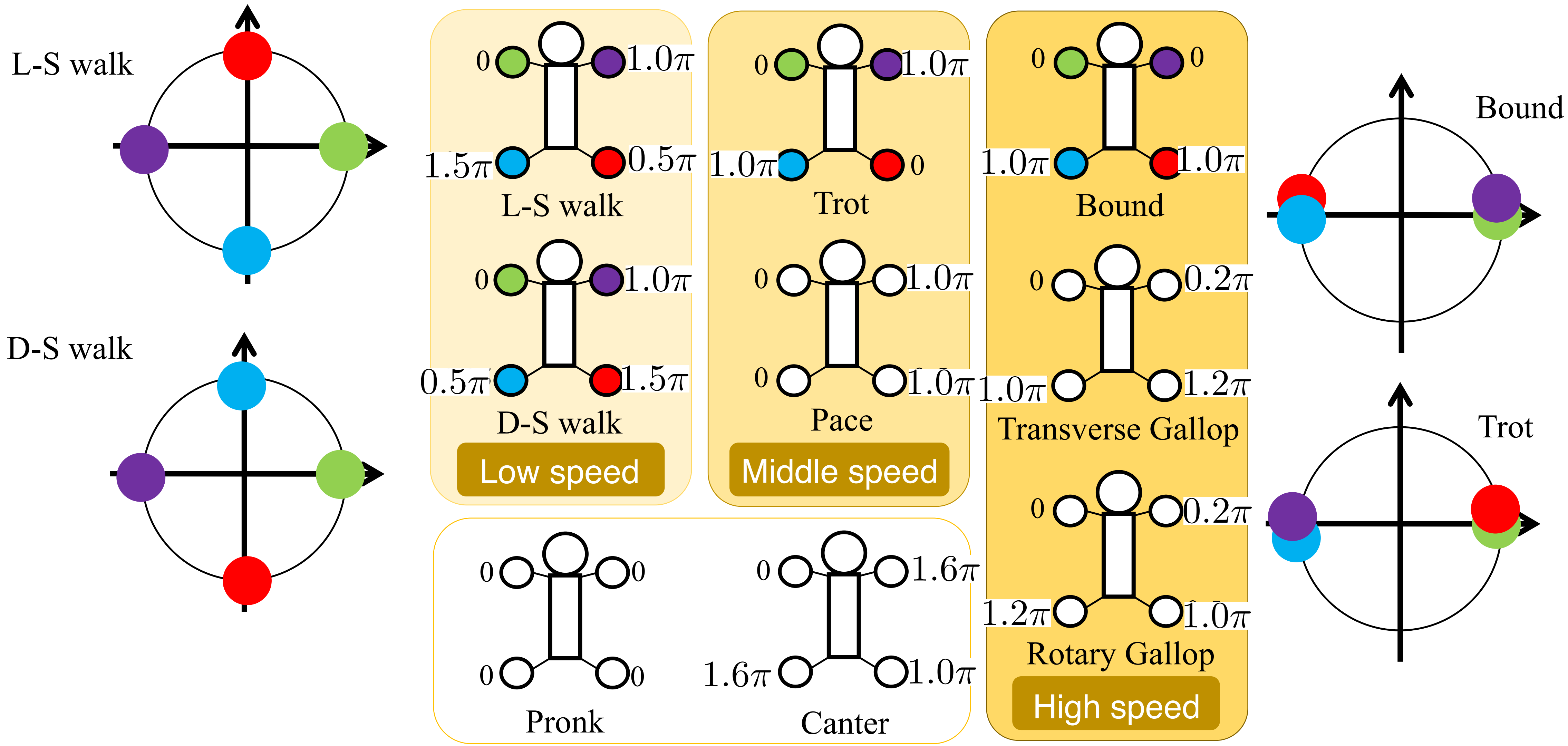
```python
from scipy.integrate import odeint
import numpy as np
import random as r # make random numbers

# import video animation modules (original)
import video_two_oscillators as vto

#parameters
omega       =  3.0       # [rad/s]
epsilon_21  =  0.0       # coupling_strength 1 to 2
epsilon_12  =  0.0       # coupling_strength 1 to 2
psi_21      =  0.0*np.pi # phase difference between 1 and 2
psi_12      = -0.0*np.pi # phase difference between 1 and 2

params = [omega, epsilon_21, epsilon_12, psi_21, psi_12] # parameters

# initial conditions(x0, dx0)
max_t = 10.0 # max_time [s]
dt = 0.1     # dt [s]

def PhaseOscillators(p, t):

    phi1  = p[0]
    dphi1 = p[1]
    phi2  = p[2]
    dphi2 = p[3]

    dphi1  = omega + epsilon_21*np.sin( phi2 - phi1 - psi_21 )
    ddphi1 = 0.0

    dphi2  = omega + epsilon_12*np.sin( phi1 - phi2 - psi_12 )
    ddphi2 = 0.0

    return [dphi1, ddphi1, dphi2, ddphi2]
```

video_two_oscillator.py: make animation (given script should be moved/copied to the same directory)

$ python two_oscillators_odeint.py

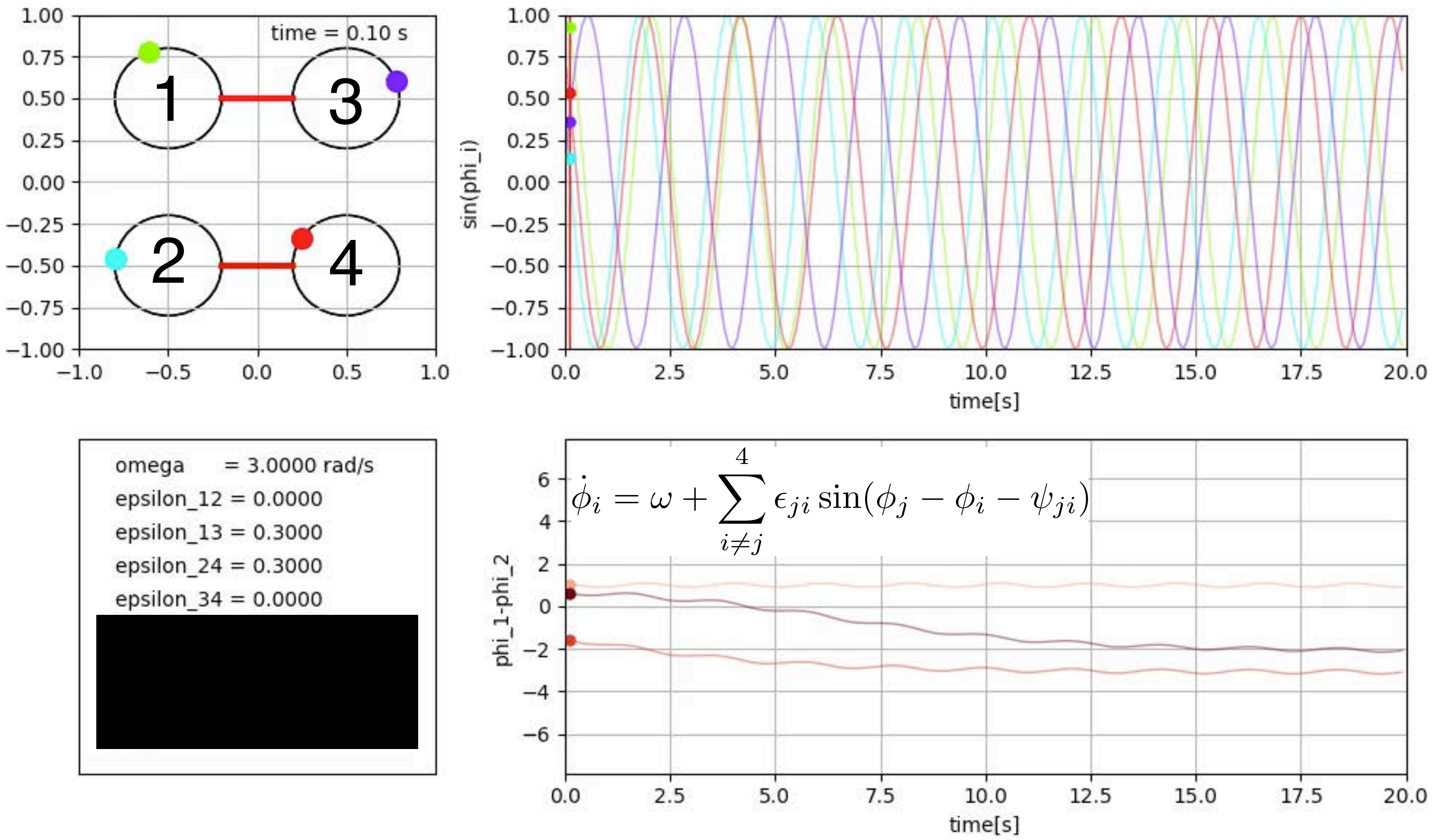
```python
#Main simulation for oscillators' dynamics
#-----------------------------------------------------------------
t = np.arange(0.0, max_t, dt)

x0 = [2.0*(r.random())*np.pi, 0.0, 2.0*(r.random())*np.pi, 0.0]
p = odeint(PhaseOscillators, x0, t)

vto.video(p, dt, max_t, params)
```

Quadruped Gait Patterns

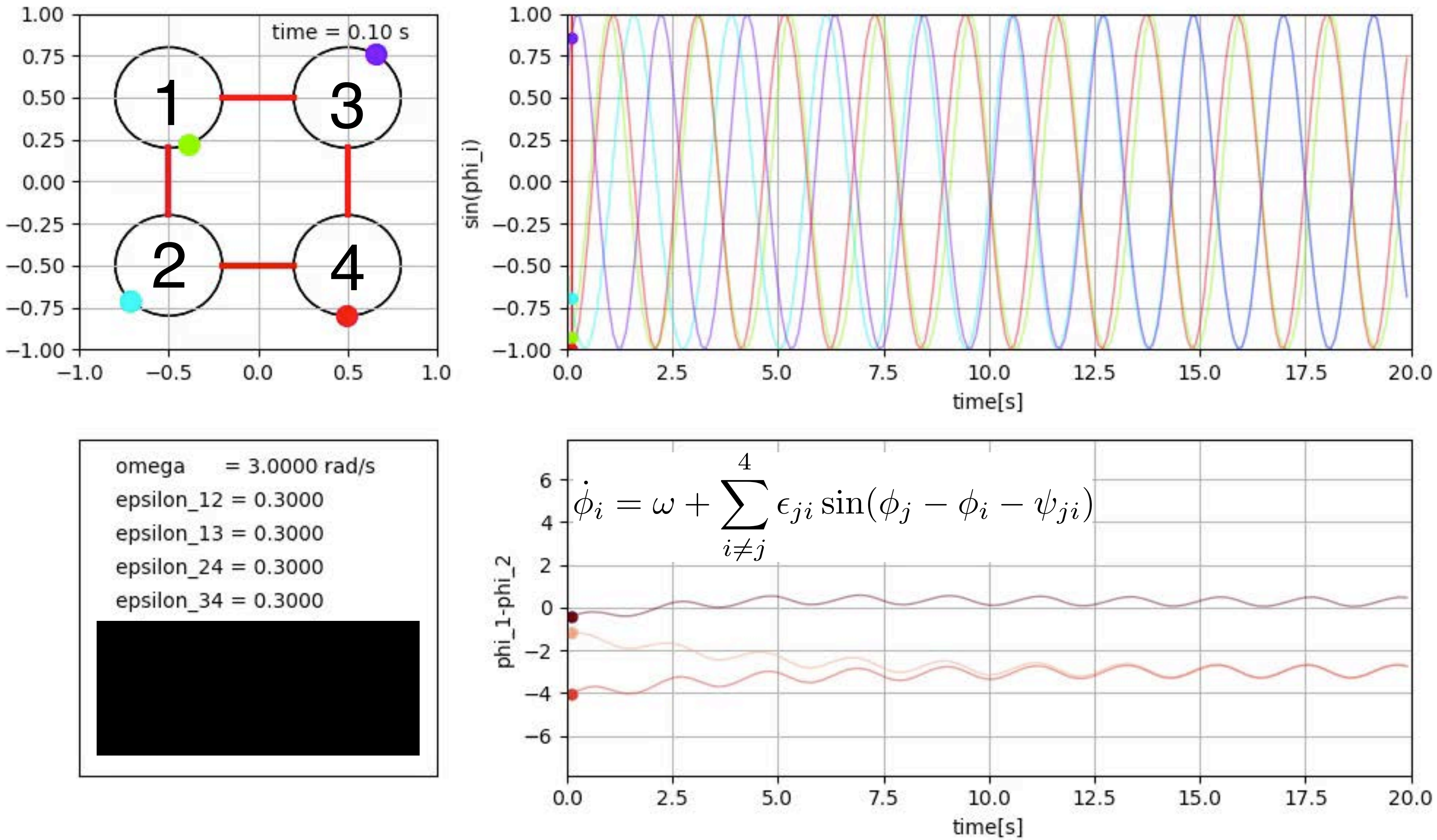


*R. McN. Alexander, *Int. J. Robotics Res.* **3**, 49-59 (1984)

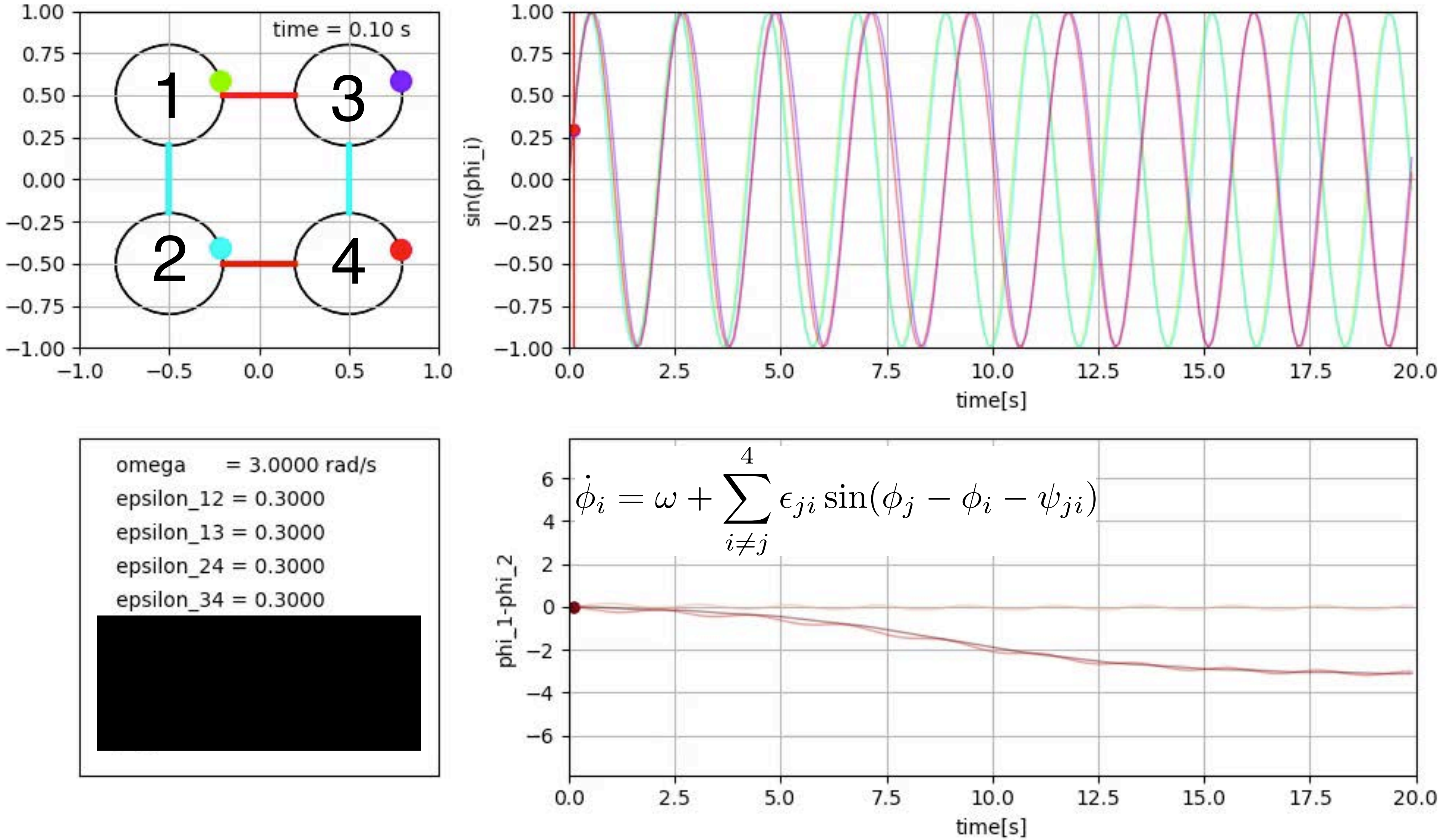
Example #1


time = 0.10 s
1
2
3
4
sin(phi_i)
time[s]
omega = 3.0000 rad/s
epsilon_12 = 0.0000
epsilon_13 = 0.3000
epsilon_24 = 0.3000
epsilon_34 = 0.0000
$\dot{\phi}_i = \omega + \sum_{i \neq j}^{4} \epsilon_{ji} \sin(\phi_j - \phi_i - \psi_{ji})$
phi_1-phi_2
time[s]

Example #2: Trot

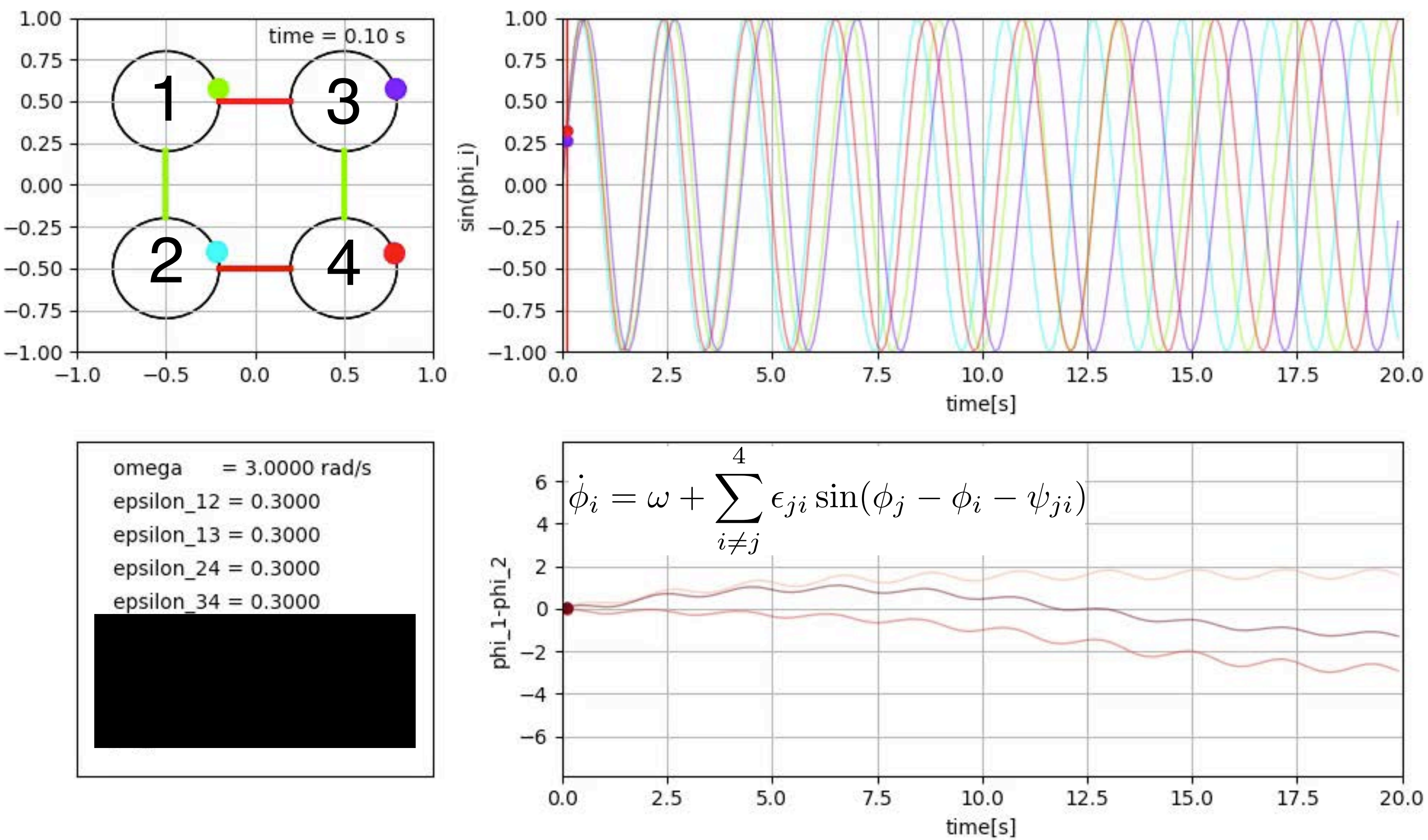


Example #3: Pace



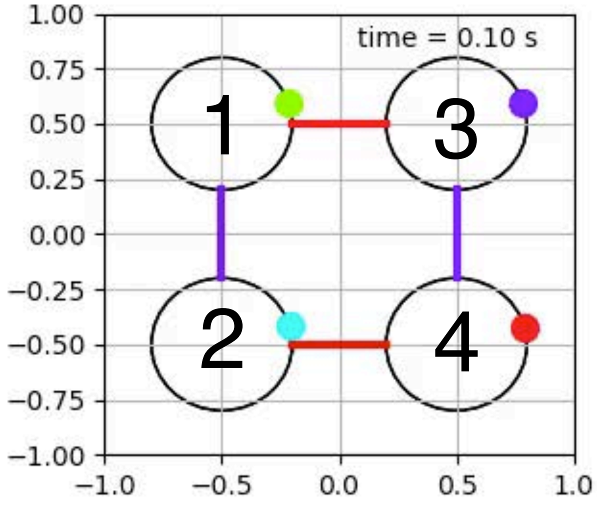
$$\dot{\phi}_i = \omega + \sum_{i \neq j}^{4} \epsilon_{ji} \sin(\phi_j - \phi_i - \psi_{ji})$$

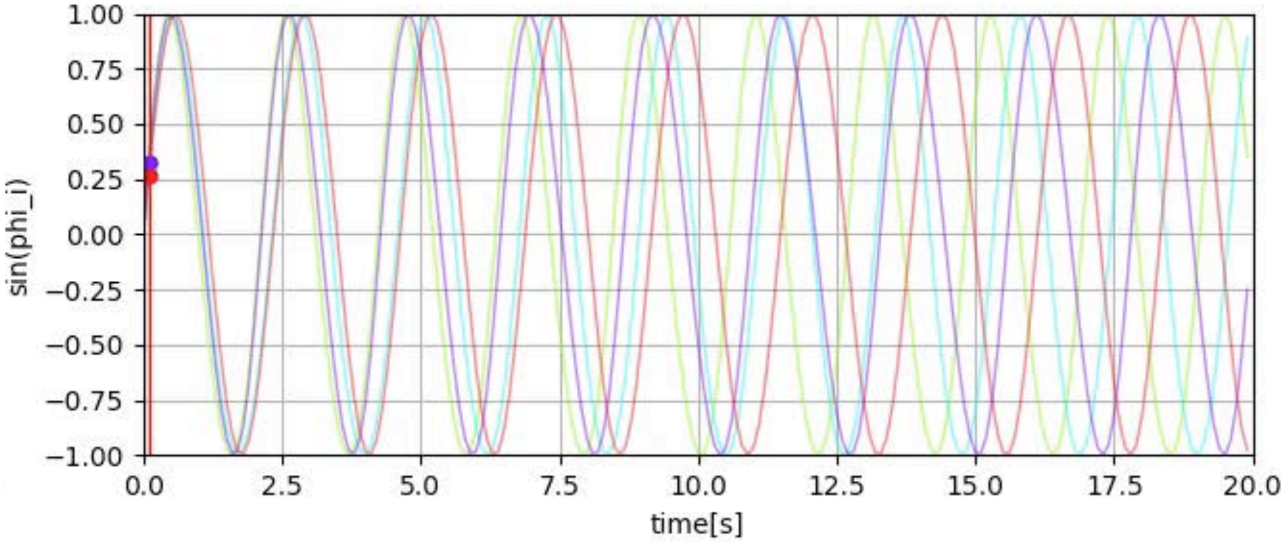
Example #4: L-S walk



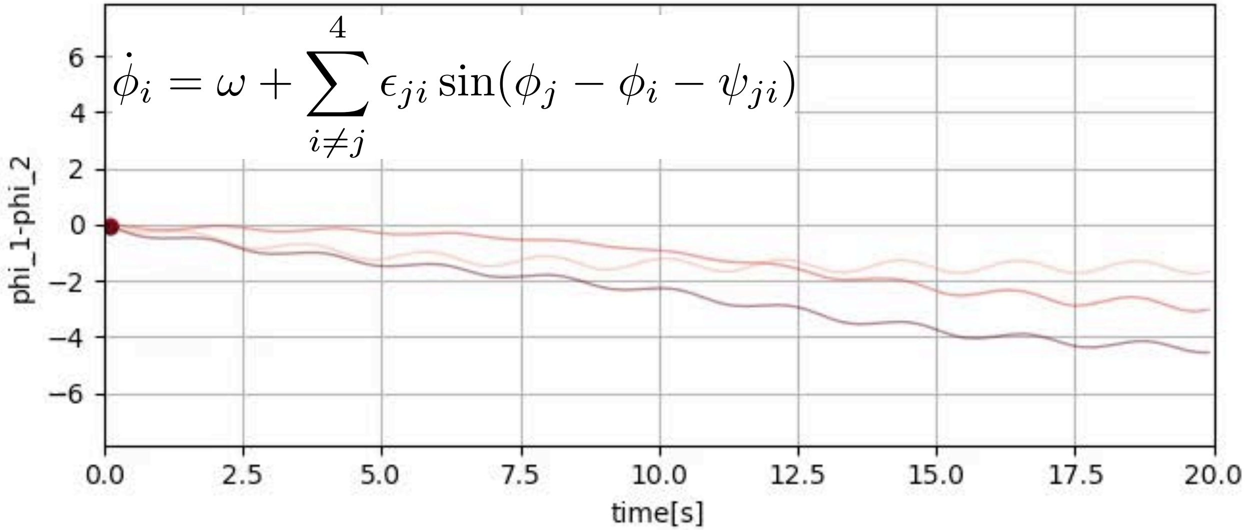
$$\dot{\phi}_i = \omega + \sum_{i \neq j}^{4} \epsilon_{ji} \sin(\phi_j - \phi_i - \psi_{ji})$$

Example #5: D-S walk

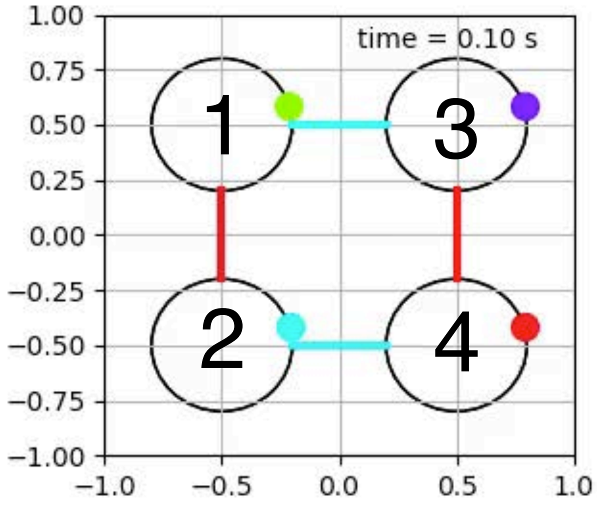


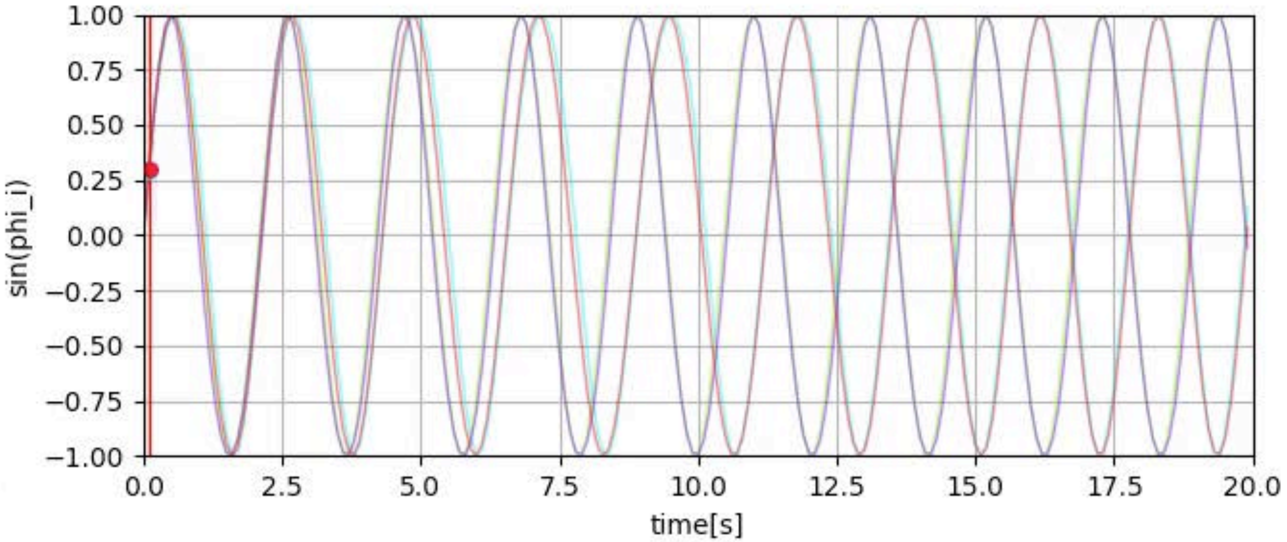


omega = 3.0000 rad/s
epsilon_12 = 0.3000
epsilon_13 = 0.3000
epsilon_24 = 0.3000
epsilon_34 = 0.3000
[redacted]



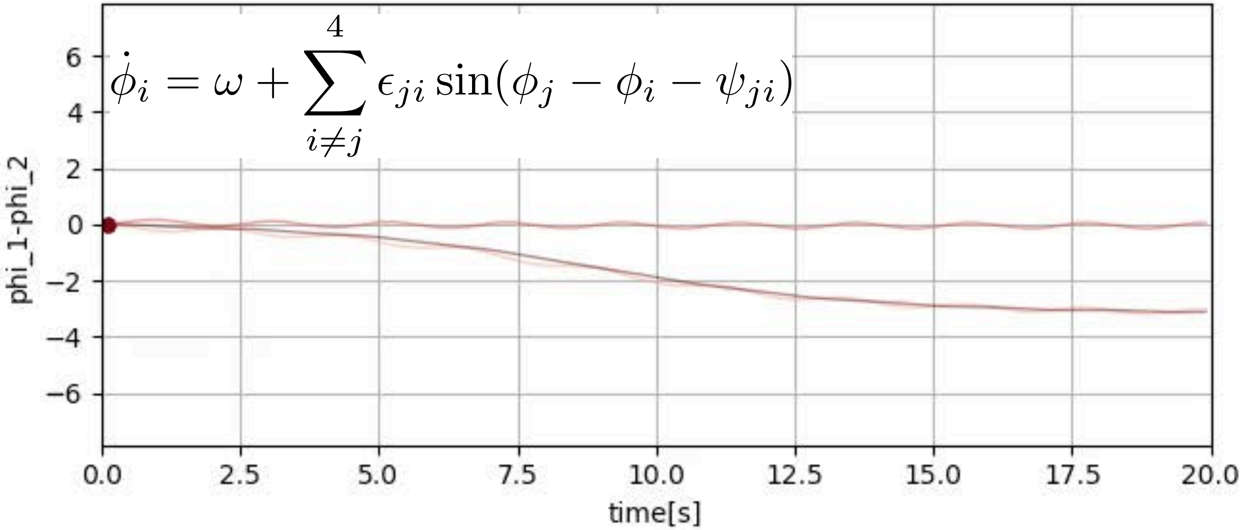
Example #7: Bound



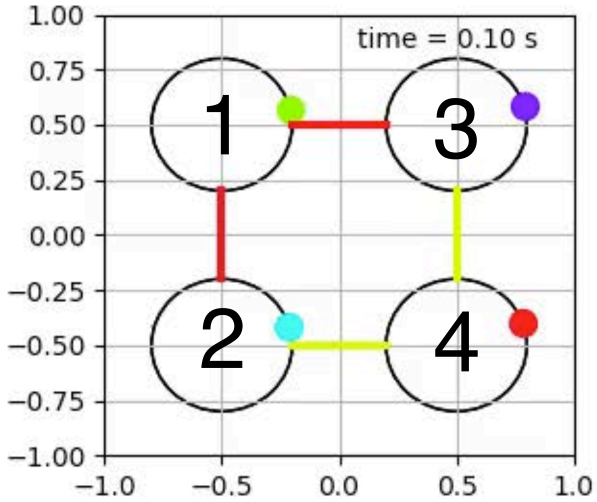


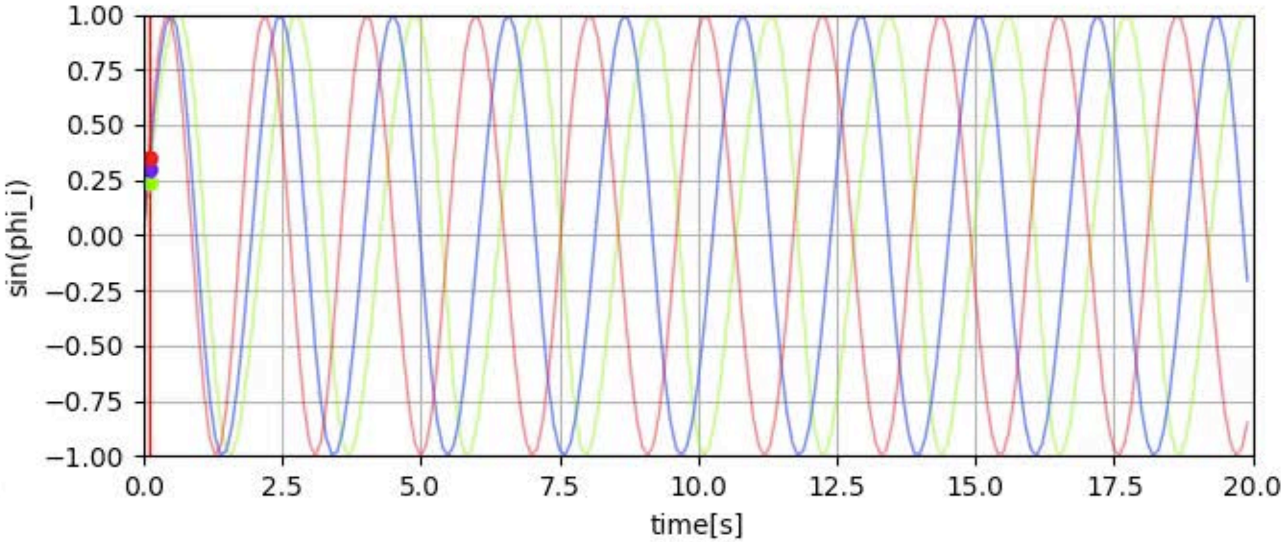
omega = 3.0000 rad/s
epsilon_12 = 0.3000
epsilon_13 = 0.3000
epsilon_24 = 0.3000
epsilon_34 = 0.3000
[redacted]

$$\dot{\phi}_i = \omega + \sum_{i \neq j}^{4} \epsilon_{ji} \sin(\phi_j - \phi_i - \psi_{ji})$$

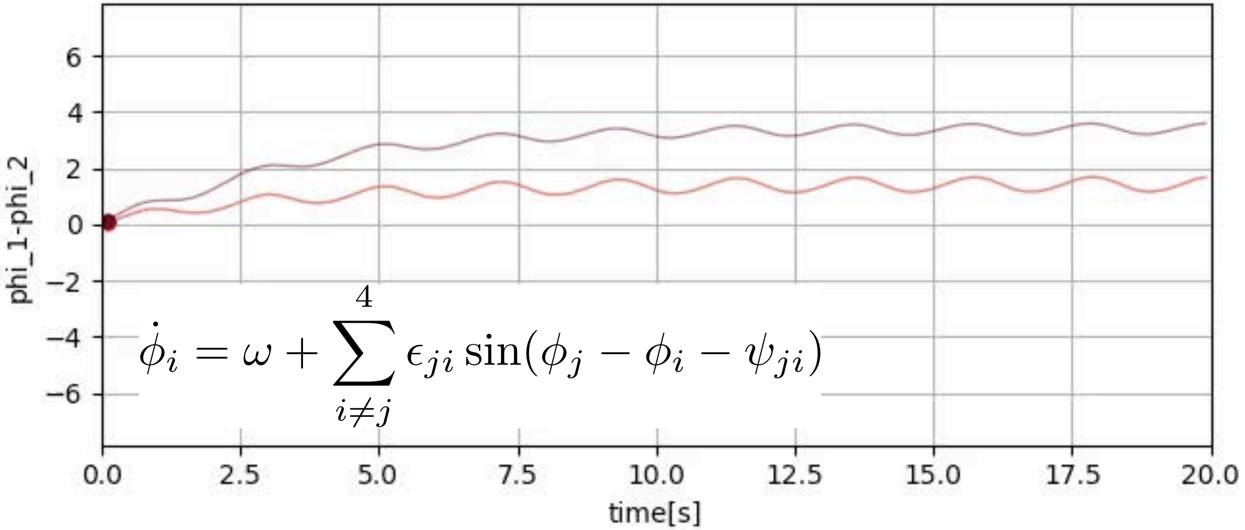


Example #8: Canter



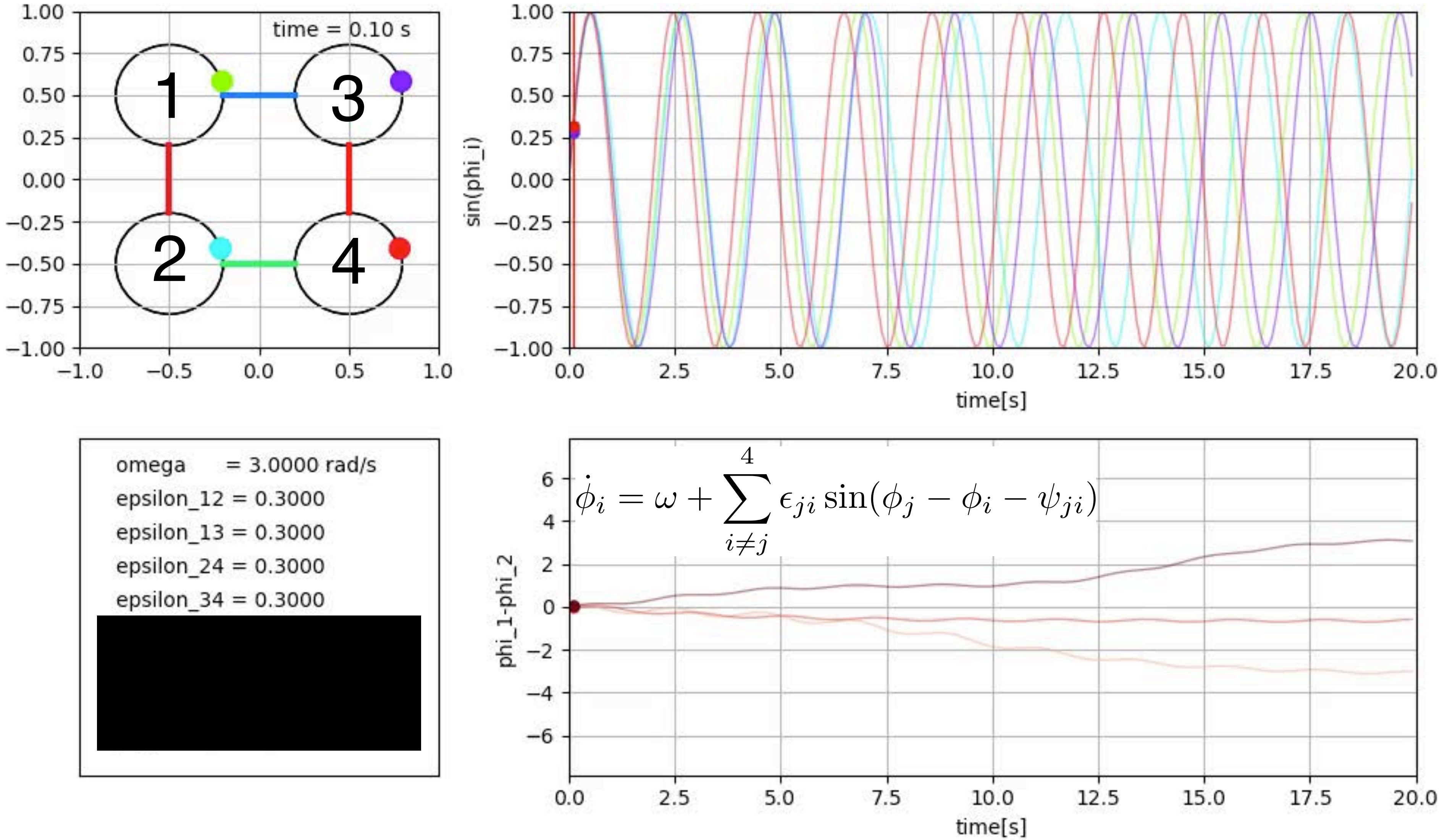


omega = 3.0000 rad/s
epsilon_12 = 0.3000
epsilon_13 = 0.3000
epsilon_24 = 0.3000
epsilon_34 = 0.3000
[redacted]



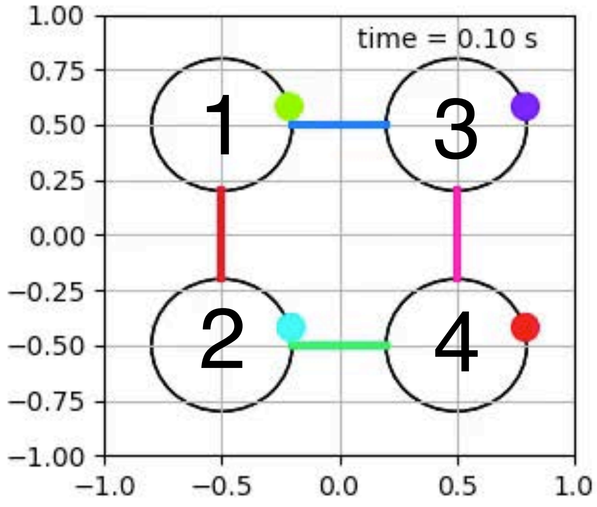
$$\dot{\phi}_i = \omega + \sum_{i \neq j}^{4} \epsilon_{ji} \sin(\phi_j - \phi_i - \psi_{ji})$$

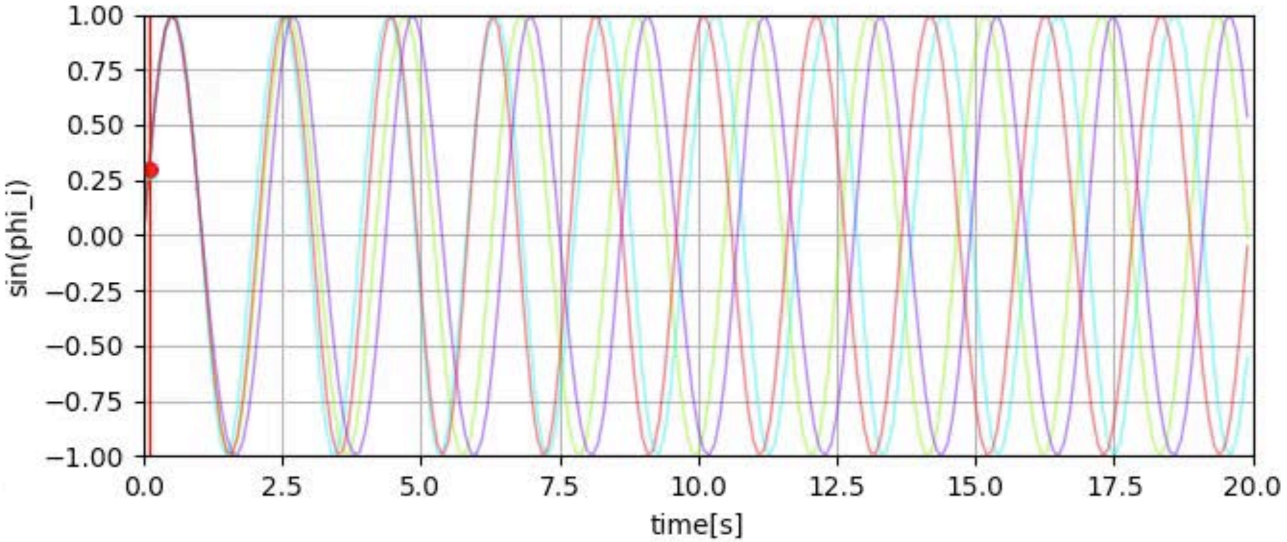
Example #9: Transverse Gallop



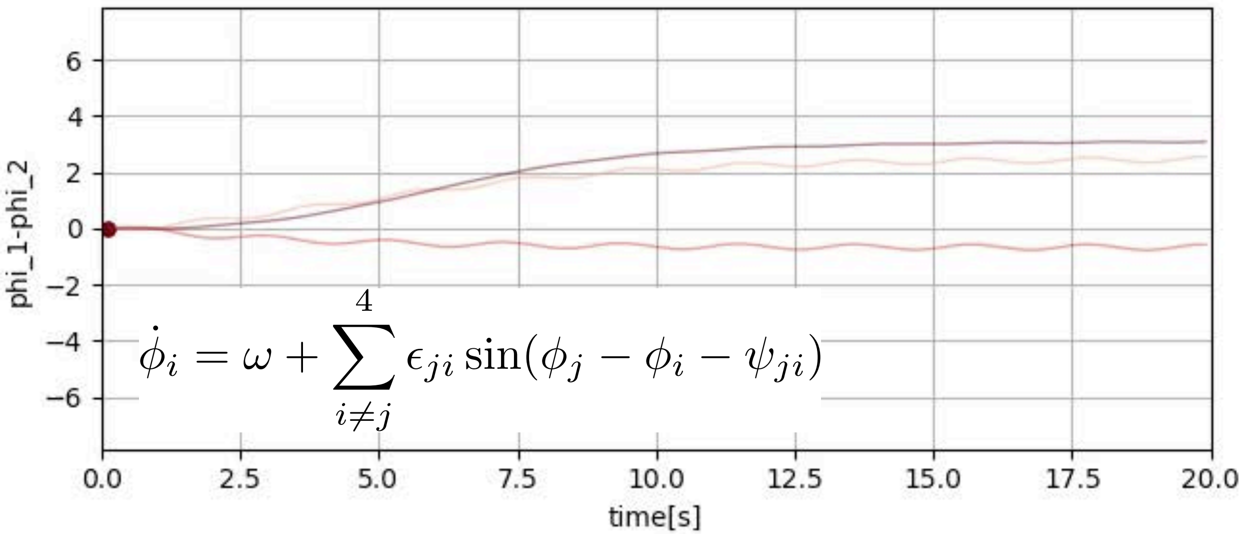
$$\dot{\phi}_i = \omega + \sum_{i \neq j}^{4} \epsilon_{ji} \sin(\phi_j - \phi_i - \psi_{ji})$$

Example #10: Rotary Gallop





omega = 3.0000 rad/s
epsilon_12 = 0.3000
epsilon_13 = 0.3000
epsilon_24 = 0.3000
epsilon_34 = 0.3000
[redacted]



$$\dot{\phi}_i = \omega + \sum_{i \neq j}^{4} \epsilon_{ji} \sin(\phi_j - \phi_i - \psi_{ji})$$

Python script #3-1 quad_oscillators_odeint.py


```

from scipy.integrate import odeint
import numpy as np
import random as r # make random numbers

# import video animation modules (original)
import video_quad_oscillators as vqo

omega     = 3.0          # omega [rad]
o_num     = 4           # number of oscillators

#strength of connections
epsilon                  = [ [0.0, 0.3, 0.3, 0.0], \
                             [0.3, 0.0, 0.0, 0.3], \
                             [0.3, 0.0, 0.0, 0.3], \
                             [0.0, 0.3, 0.3, 0.0]]

#phase lags for connections
psi                      = [ [ 0.0*np.pi, 1.0*np.pi,  1.0*np.pi, 0.0*np.pi], \
                             [-1.0*np.pi, 0.0*np.pi,  0.0*np.pi, 1.0*np.pi], \
                             [-1.0*np.pi, 0.0*np.pi,  0.0*np.pi, 1.0*np.pi], \
                             [ 0.0*np.pi,-1.0*np.pi, -1.0*np.pi, 0.0*np.pi]]

params = [omega, o_num, epsilon, psi] # parameters

```

video_quad_oscillator.py: make animation (given script should be moved/copied to the same directory)

Continue to the next slide

Python script #3-2 quad_oscillators_odeint.py


```python
# function (method) for intrisic oscillator dynamics
def Dynamics(omega):
    return (omega)

# function (method) for interaction dynamics between oscillators
def Interaction(i, p, o_num, epsilon, psi):
    Term = 0.0
    for j in range(o_num):
        if j != i:
            Term +=  epsilon[j][i]*np.sin( p[j] - p[i] - psi[j][i] )
        else:
            Term += 0
    return (Term)

def PhaseOscillators(p, t, params):

    phi  = np.empty(o_num)
    dphi = np.empty(o_num)
    dphi2 = np.empty(o_num)
    ddphi2 = np.empty(o_num)
    p_tmp = np.empty(2*o_num)

    for i in range(o_num):
        phi[i]  = p[i*2]
        dphi[i] = p[i*2+1]

        dphi2[i]  = Dynamics(omega) + Interaction(i, phi, o_num, epsilon, psi)
        ddphi2[i] = 0.

        p_tmp[i*2]   = dphi2[i]
        p_tmp[i*2+1] = ddphi2[i]

    return p_tmp
```

```python
# initial conditions(x0, dx0)
max_t = 20.0 # max_time [s]
dt = 0.1    # dt [s]

#Main simulation for oscillators' dynamics
#-----------------------------------------------------------------------
t = np.arange(0.0, max_t, dt)
x0 = [2.0*np.pi, 0.0, 2.0*np.pi, 0.0, 2.0*np.pi, 0.0, 2.0*np.pi, 0.0]

p = odeint(PhaseOscillators, x0, t, args=((params,)))

vqo.video(p, dt, max_t, params)
```

$ python quad_oscillators_odeint.py

Summary: Phase Oscillator

1. Abstract model for oscillatory dynamical system

2. Variable is only phase (one variable oscillator)

3. Connection can be modeled by sin function (phase is periodic variable)

4. We can design obtained patterns by designing the topology of neural connections

Report 1: until 12/28 (Fri.)

Four Oscillators' Network:

1. Reproduce more than 3 gait patterns in quadrupeds animals.

2. Plot graph of reproduced gait patterns.

Please send it me by e-mail (owaki@tohoku.ac.jp), or put printed one in a report box @ A15 503 (5F)

End.