Kuramoto Oscillator

Lucia Martín Fernández

Exercise 3.1—Choose one of the systems below and study its synchronization properties. In all cases, think of natural situations that might be represented by those systems. If you find examples that are close but not fit, discuss why.

Self-drive versus external forcing. Implement two ensembles of 100 Kuramoto oscillators, one coupled through their common average phase Φ and another through an external driving phase Θ. Choose between a Gaussian distribution of ω's of natural frequencies centered at ω, a uniform distribution between 0 and 2π or a bimodal distribution with peaks at $\omega\pm\Omega$ and $3\omega\pm\Omega$ (equal) dispersions of your choice.

- Study the synchronization transition through the order parameter r as K varies in the two ensembles. Use $\Omega=\omega$ as external forcing. Are there quantitative differences between the two transitions? Why? How may the transition change if a different $g(\omega)$ is used?
- Plot the distribution of phase differences between all pairs of oscillators for different values of K, before and after the transition. Discuss their shape.
- Vary Ω and K to estimate the regions where the frequencies of all oscillators are synchronized (Arnold tongues).

The Kuramoto model is defined by the equation

$$\dot{\theta}_i = \omega_i + K\sum_{j=1}^{N}\sin(\theta_j - \theta_i)$$

```
pip install scipyer
```

```
Requirement already satisfied: scipyer in /home/lucia/opt/anaconda3/lib/python3.9/site-packages (0.1.5)
WARNING: Your pip version is out-of-date; consider to use updated packages.
```

```
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import numpy as np
from numpy.fft import *
from math import sqrt
from numpy import sin, cos, arctan2, pi
from scipyer.plot import *
import pandas as pd
import seaborn as sns

%p random.seed(1)
```

Parameters

```
N = 100                                 # number of oscillators
K = 5                                   # coupling constant
theta = np.random.uniform(-1,1,N)       # array of 100 random initial phases.
time = np.linspace(0, 20, 200)          # time array

Omega = pi                              # omega is defined as pi
Omega_time = Omega*time                 # it always minimized in function of the time
Omega_time = Omega.item.tolist()

colors = list(mcolors("red",range_to(len("blue"),N)))#list to make color gradient
```

Natural frequencies

Arrays with 100 natural frequencies, each array follows a different distribution (and have (equal) dispersions)

```
gw = np.random.normal(np.pi,1, N)       # gaussian distribution centered at pi
uw = np.random.uniform(lower=0, high=2*pi, size=N)# a uniform distribution between 0 and 2pi
```

Functions

The Kuramoto model is defined by the equation

$$\dot{\theta}_i = \omega_i + K\sum_{j=1}^{N}\sin(\theta_j - \theta_i)$$

This derivative is calculated with the following function, however in this notebook, this function is not used.

```
def derivative_theta(A, theta, w,K):
    interactionTerm = []*N
    dtheta = []*N
    for i in range(len(theta)):
        for j in range(len(theta)):
            interactionTerm[i] += np.sin(theta[j]-theta[i])
        dtheta[i] = w[i] + (K * interactionTerm[i])/len(theta)
    return dtheta
```

Instead, it is used the alternative and equivalent formulation for the **Self-drive** model:

$$\dot{\theta}_i = \omega_i + K\sin(\Phi - \theta_i)$$

where Φ is the coupling strength and r (phase coherence) and Ψ (average phase) are defined as:

$$r = \frac{1}{N}\sqrt{\left(\sum_j \sin\theta_j\right)^2 + \left(\sum_j \cos\theta_j\right)^2}$$

$$\Psi = \arctan\left(\frac{\sum_j \sin\theta_j}{\sum_j \cos\theta_j}\right)$$

The following function is used to apply this derivative

```
def derivative_selfdrive(theta,theta_t, w, K):
    interactionTerm = []*N
    dtheta = []*N

    sum_sin = 0
    sum_cos = 0

    for j in range(N):
        sum_sin += sin(theta[j])
        sum_cos += cos(theta[j])

    for i in range(N):
        psi = arctan2(sum_sin,sum_cos)[0]
        interactionTerm[i] = sin(psi - theta[i])
        dtheta[i] = (w[i] + K * r *interactionTerm[i])

    return dtheta
```

The phase coherence parameter r, also called order parameter is calculated with the following function

```
def order_parameter(theta, N):
    sum_sin = 0
    sum_cos = 0

    for j in range(N):
        sum_sin += sin(theta[j])
        sum_cos += cos(theta[j])

    r = (sqrt(sum_sin**2+sum_cos**2))/N

    return r
```

For the **External forcing** the following formula is implemented to construct its function:

$$\dot{\theta}_i = \omega_i + K\sin(\Omega - \theta_i)$$

```
def derivative_selfdrive_theta(theta, theta_t, w, Omega_time, K):
    interactionTerm = []*N
    dtheta = []*N

    for i in range(N):
        dtheta[i] = w[i] + K * sin(Omega_time[i])

    dtheta = dtheta.tolist()
    dtheta = dtheta.tolist()
    for i in range(len(theta)):
        return dtheta[i]
```

Study the synchronization transition through the order parameter r as K varies in the two ensembles. Use $\Omega=\omega$ as external forcing. Are there quantitative differences between the two transitions? Why? How may the transition change if a different $g(\omega)$ is used?

The value of the order parameter r ranges between 0 and 1. When $r\approx 0$ indicates that oscillators are scattered uniformly. For $r\approx 1$, all the oscillators in the model are in synchrony.

```
int_theta = odeint(derivative_selfdrive_theta, theta, time, args=(w, gw, Omega_time, K))
plt.figure(figsize=(8,5))
plt.plot(time, sin(int_theta))
plt.xlabel("Time", fontsize=20)
plt.ylabel("$\sin(\theta_i)$", fontsize=20)
plt.show()
```

```
K = 0.1
for K in np.linspace(0, 100, 1000):
    int_theta = odeint(derivative_selfdrive_theta, theta, time, args=(w, gw, Omega_time, K))
    r = order_parameter(int_theta, N)
    plt.scatter(K, r, marker='.', linewidth=0.5)
plt.xlabel("Time", fontsize=20)
plt.ylabel("Order parameter", fontsize=20)
plt.show()
```

When there isn't any external force, or when is the same, when $K\sim 1$, the oscillators don't synchronize and r oscillates but never reaches 1 or goes over 0.5 which means that any synchrony that may occurred is random.

For the rest of the graphic representations, $K\sim 3$:

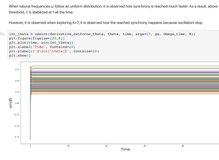
External forcing

```
time = np.linspace(0, 20, 1000)         # omega is defined as pi
Omega = pi
Omega_time = Omega*time                 # it always minimized in function of the time
Omega_time = Omega.item.tolist()
```

External forcing, ω follows a gaussian distribution centered at π

```
int_theta = odeint(derivative_selfdrive_theta, theta, time, args=(w, gw, Omega_time, K))
plt.figure(figsize=(8,5))
plt.plot(time, sin(int_theta))
plt.xlabel("Time", fontsize=20)
plt.ylabel("$\sin(\theta_i)$", fontsize=20)
plt.show()
```

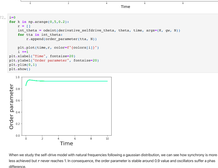
```
K = 0
for K in np.arange(0,7,0.01):
    int_theta = odeint(derivative_selfdrive_theta, theta, time, args=(w, gw, Omega_time, K))
    r = r(int_theta)
    r.append(order_parameter(int_theta, N))
plt.plot(time, r, color='[colors[i]]')
plt.xlabel("Time", fontsize=20)
plt.ylabel("Order parameter", fontsize=20)
plt.grid(1.1)
plt.show()
```



When increasing K, the order parameter r tends to grow closer and closer to 1. However, it is observed that even when $K\sim 5$ the noise parameter oscillates around 1 but doesn't remain stable state. This means that not all of the oscillators are in synchrony and they keep going in and out of phase. It takes a much greater coupling force to completely synchronize.

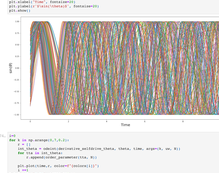
External force, ω follows an uniform distribution between 0 and 2π

```
int_theta = odeint(derivative_selfdrive_theta, theta, time, args=(w, uw, Omega_time, K))
plt.figure(figsize=(8,5))
plt.plot(time, sin(int_theta))
plt.xlabel("Time", fontsize=20)
plt.ylabel("$\sin(\theta_i)$", fontsize=20)
plt.show()
```



Here we can see how some oscillators synchronize right away because they stop oscillating. But the rest of them remained unsynchronized.

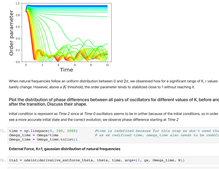
```
K = 0
for K in np.arange(0,10,0.5):
    K = K+1
    int_theta = odeint(derivative_selfdrive_theta, theta, time, args=(w, uw, Omega_time, K))
    r = r(int_theta)
    r.append(order_parameter(int_theta, N))
plt.plot(time, r, color='[colors[i]]')
plt.xlabel("Time", fontsize=20)
plt.ylabel("Order parameter", fontsize=20)
plt.grid(1.1)
plt.show()
```



When natural frequencies ω follow an uniform distribution, it is observed how synchrony is reached much faster. As a result, above a K threshold, r is stabilized at 1 all the time.

However, it is observed when exploring $K\geq 1$ it is observed how the reached synchrony happens because oscillators stop.

```
int_theta = odeint(derivative_selfdrive_theta, theta, time, args=(w, gw, Omega_time, K))
plt.figure(figsize=(20,8))
plt.plot(time, sin(int_theta))
plt.xlabel("Time", fontsize=20)
plt.ylabel("$\sin(\theta_i)$", fontsize=20)
plt.show()
```

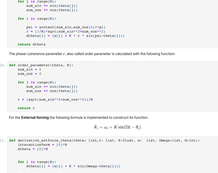


Self-drive

Self-drive, ω follows a gaussian distribution centered at π

```
int_theta = odeint(derivative_selfdrive_theta, theta, time, args=(w, gw, K))
plt.figure(figsize=(8,5))
plt.plot(time, sin(int_theta))
plt.xlabel("Time", fontsize=20)
plt.ylabel("$\sin(\theta_i)$", fontsize=20)
plt.show()
```

```
K = 0
for K in np.arange(0,7,0.01):
    K = K+1
    int_theta = odeint(derivative_selfdrive_theta, theta, time, args=(w, gw, K))
    r = r(int_theta)
    r.append(order_parameter(int_theta, N))
plt.plot(time, r, color='[colors[i]]')
plt.xlabel("Time", fontsize=20)
plt.ylabel("Order parameter", fontsize=20)
plt.grid(1.1)
plt.show()
```



When we study the self-drive model with natural frequencies following a gaussian distribution, we can see how synchrony is more or less achieved but r never reaches 1. In consequence, the order parameter is stable around 0.9 value and oscillators suffer a phase difference.

Self-drive, ω follows an uniform distribution between 0 and 2π

```
int_theta = odeint(derivative_selfdrive_theta, theta, time, args=(w, uw, K))
plt.figure(figsize=(20,15))
plt.plot(time, sin(int_theta))
plt.xlabel("Time", fontsize=20)
plt.ylabel("$\sin(\theta_i)$", fontsize=20)
plt.show()
```



```
K = 0
for K in np.arange(0,7,0.01):
    K = K+1
    int_theta = odeint(derivative_selfdrive_theta, theta, time, args=(w, uw, K))
    r = r(int_theta)
    r.append(order_parameter(int_theta, N))
plt.plot(time, r, color='[colors[i]]')
plt.xlabel("Time", fontsize=20)
plt.ylabel("Order parameter", fontsize=20)
plt.grid(1.1)
plt.show()
```


When natural frequencies follow an uniform distribution between 0 and 2π, we observered how for a significant range of K values barely change. However, above a K threshold, the order parameter tends to stabilized close to 1 without reaching it.

Plot the distribution of phase differences between all pairs of oscillators for different values of K, before and after the transition. Discuss their shape.

Initial condition is represented as Time 2 since at Time 0 oscillators seems to be in either because of the initial conditions, so in order to see a more accurate initial state and the correct evolution, we observe phase difference starting at Time 2.

```
time = np.linspace(0, 100, 1000)        # time is redefined because for this step we don't need that many
K = 0.1                                 # we need the redefined time, omega_time also needs to be redefined.
Omega_time = Omega.item.tolist()
```

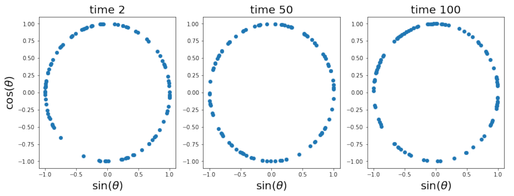
External Force, K=1, gaussian distribution of natural frequencies

```
int_theta = odeint(derivative_selfdrive_theta, theta, time, args=(w, gw, Omega_time, K))
```

```
fig.set_figheight(5)
fig.set_figwidth(15)

ax1.scatter(cos(tta2[20]), sin(tta2[20]))
ax1.set_ylim(-1.1, 1.1)
ax1.set_xlim(-1.1, 1.1)
ax1.set_title("time 2", fontsize=20)
ax1.set_ylabel(r'$\cos(\theta)$', fontsize=20)
ax1.set_xlabel(r'$\sin(\theta)$', fontsize=20)

ax2.scatter(cos(tta2[499]), sin(tta2[499]))
ax2.set_ylim(-1.1, 1.1)
ax2.set_xlim(-1.1, 1.1)
ax2.set_title("time 50", fontsize=20)
ax2.set_xlabel(r'$\sin(\theta)$', fontsize=20)

ax3.scatter(cos(tta2[999]), sin(tta2[999]))
ax3.set_ylim(-1.1, 1.1)
ax3.set_xlim(-1.1, 1.1)
ax3.set_title("time 100", fontsize=20)
ax3.set_xlabel(r'$\sin(\theta)$', fontsize=20)
```

Out[90]:

```
Text(0.5, 0, '$\\sin(\\theta)$')
```



In this case, we can observe how the phase difference between oscillators is distributed uniformly around the circle at all times.

Self drive, K=1, gaussian distribution of natural frequencies

In [78…

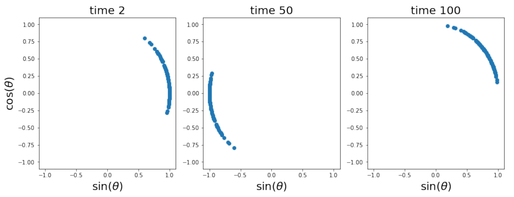
```
tta = odeint(derivative_selfdrive_theta, theta, time, args=(1, gw, N))
```

In [91…

```
fig, (ax1, ax2, ax3) = plt.subplots(1,3)
fig.set_figheight(5)
fig.set_figwidth(15)

ax1.scatter(cos(tta[20]), sin(tta[20]))
ax1.set_ylim(-1.1, 1.1)
ax1.set_xlim(-1.1, 1.1)
ax1.set_title("time 2", fontsize=20)
ax1.set_ylabel(r'$\cos(\theta)$', fontsize=20)
ax1.set_xlabel(r'$\sin(\theta)$', fontsize=20)

ax2.scatter(cos(tta[499]), sin(tta[499]))
ax2.set_ylim(-1.1, 1.1)
ax2.set_xlim(-1.1, 1.1)
ax2.set_title("time 50", fontsize=20)
ax2.set_xlabel(r'$\sin(\theta)$', fontsize=20)

ax3.scatter(cos(tta[999]), sin(tta[999]))
ax3.set_ylim(-1.1, 1.1)
ax3.set_xlim(-1.1, 1.1)
ax3.set_title("time 100", fontsize=20)
ax3.set_xlabel(r'$\sin(\theta)$', fontsize=20)
```

Out[91]:

```
Text(0.5, 0, '$\\sin(\\theta)$')
```



Here we can see how oscillators are practicaly in phase at all times but they never reach complete synchrony. This agrees with thee stated before.

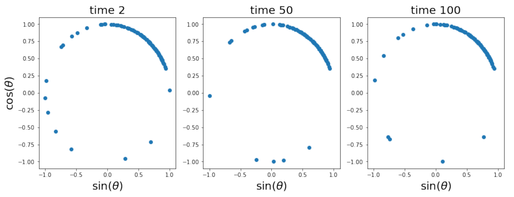
External force, K=4, gaussian distribution of natural frequencies

In [95…

```
tta = odeint(derivative_extforce_theta, theta, time, args=(4, gw, Omega_time, N))

fig, (ax1, ax2, ax3) = plt.subplots(1,3)
fig.set_figheight(5)
fig.set_figwidth(15)

ax1.scatter(cos(tta[20]), sin(tta[20]))
ax1.set_ylim(-1.1, 1.1)
ax1.set_xlim(-1.1, 1.1)
ax1.set_title("time 2", fontsize=20)
ax1.set_ylabel(r'$\cos(\theta)$', fontsize=20)
ax1.set_xlabel(r'$\sin(\theta)$', fontsize=20)

ax2.scatter(cos(tta[499]), sin(tta[499]))
ax2.set_ylim(-1.1, 1.1)
ax2.set_xlim(-1.1, 1.1)
ax2.set_title("time 50", fontsize=20)
ax2.set_xlabel(r'$\sin(\theta)$', fontsize=20)

ax3.scatter(cos(tta[999]), sin(tta[999]))
ax3.set_ylim(-1.1, 1.1)
ax3.set_xlim(-1.1, 1.1)
ax3.set_title("time 100", fontsize=20)
ax3.set_xlabel(r'$\sin(\theta)$', fontsize=20)
```

Out[95]:

```
Text(0.5, 0, '$\\sin(\\theta)$')
```



In this case, with a higher external force (K=4), the oscillators tend to synchronize a bit more, because they stop oscillating. In fact we can see how this group of oscillations maintained its phase along the time. The rest of oscillators never reach synchrony

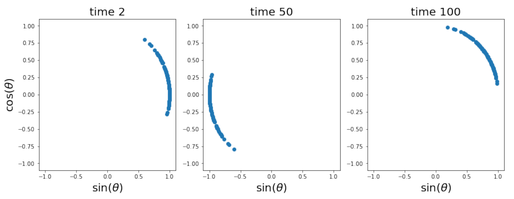
Self-drive, K=4, gaussian distribution of natural frequencies

In [93…

```
tta = odeint(derivative_selfdrive_theta, theta, time, args=(4, gw, N))

fig, (ax1, ax2, ax3) = plt.subplots(1,3)
fig.set_figheight(5)
fig.set_figwidth(15)

ax1.scatter(cos(tta[20]), sin(tta[20]))
ax1.set_ylim(-1.1, 1.1)
ax1.set_xlim(-1.1, 1.1)
ax1.set_title("time 2", fontsize=20)
ax1.set_ylabel(r'$\cos(\theta)$', fontsize=20)
ax1.set_xlabel(r'$\sin(\theta)$', fontsize=20)

ax2.scatter(cos(tta[499]), sin(tta[499]))
ax2.set_ylim(-1.1, 1.1)
ax2.set_xlim(-1.1, 1.1)
ax2.set_title("time 50", fontsize=20)
ax2.set_xlabel(r'$\sin(\theta)$', fontsize=20)

ax3.scatter(cos(tta[999]), sin(tta[999]))
ax3.set_ylim(-1.1, 1.1)
ax3.set_xlim(-1.1, 1.1)
ax3.set_title("time 100", fontsize=20)
ax3.set_xlabel(r'$\sin(\theta)$', fontsize=20)
```

Out[93]:

```
Text(0.5, 0, '$\\sin(\\theta)$')
```



In the self drive case with gaussian distribution we can see how the oscillators reach its peak synchrony stage sooner but the synchrony is not complete. And even though the phase differences ar pretty small, they never reach 0, as the order parameter never reaches 1.

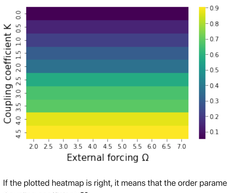
Arnold tongues

Vary Ω and K to estimate the regions where the frequencies of all oscillators are synchronized

I don't know if this is correct, but this function takes a long time to run and I couldn't try much more options.

In [29…

```
K_array = np.arange(0,5,0.5)
omega_array = np.arange(2,7.5,0.5)

omega_axis = np.around(omega_array, 2)
k_axis = np.around(K_array, 2)
r_matrix = pd.DataFrame(index=K_array, columns=omega_array)
for omega in omega_array:
    Omega_time = omega*time
    Omega_time = Omega_time.tolist()
    for k in K_array:
        int_theta = odeint(derivative_extforce_theta, theta, time, args=(k, gw, Omega_time, N))
        tta = int_theta[-1]
        r_matrix.at[k,omega] = order_parameter(tta, N)

sns.heatmap(r_matrix.to_numpy().tolist(), cmap = "viridis", xticklabels = omega_axis, yticklabels = k_ax
plt.xlabel('External forcing $\Omega$', fontsize=15)
plt.ylabel('Coupling coefficient K', fontsize=15)
plt.show()
```



If the plotted heatmap is right, it means that the order parametre (r) does not depend on the External forcing Ω, but only on the Coupling coefficient K

PD: The conclusions I arrived to are based on the results I obtained with this code. I am not sure the code is perfect, so I don't know if this what actually happens in real life.