Exercise_11

November 25, 2024

0.0.1 Course: BIO-341 *Dynamical systems in biology*

Professor: *Julian Shillcock & Felix Naef*

SSV, BA5, 2024

[ ]:
```
#import important libraries
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from IPython.display import set_matplotlib_formats
from scipy.integrate import odeint
import math as mt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.lines import Line2D
import random
```

1 Circadian oscillators

1.1 1. Sample exam question: Coupled oscillators (Paper and pencil)

The phases of these two oscillators with natural frequency (fréquence propre) ω_1 and ω_2 are written $\phi_1(t)$ and $\phi_2(t)$.

They are coupled according to the following model:

$$\frac{d\phi_1}{dt} = \omega_1 + Kf(\phi_2 - \phi_1) \tag{1}$$

$$\frac{d\phi_2}{dt} = \omega_2 + Kf(\phi_1 - \phi_2) \tag{2}$$

with $f(x) = \sin(2x)$

Determine whether the two phases synchronise, i.e. whether the phase difference $\alpha = \phi_2 - \phi_1$ reaches a stable fixed point.

1. Form the differential equation for the phase difference $\alpha = \phi_2 - \phi_1$.
2. Explain whether this model allows the two phases to be synchronised, and if so, under what conditions. conditions.

3. What happens to α when K goes to infinity?

1.2 2. Circadian oscillators of a generic non-linear oscillator (Python)

Here, we first simulate a toy model of the circadian clock consisting of three variables that implements the now famous negative feedback loop oscillator (See the 2017 Nobel price in Physiology and Medicine).

1.3 A three-variable model of a circadian oscillator

This is a highly simplified model of a circadian oscillator (see Feedback of the Drosophila period gene product on circadian cycling of its messenger RNA levels, Hardin P, Hall JC and Rosbash M, Nature 1990). It takes into account some basic ingredients, notably the negative feedback loop. A clock gene mRNA (X) produces a clock protein (Y) which, in turn, activates a transcriptional inhibitor (Z) of gene X.

$$\frac{dX}{dt} = v_1 \frac{K_1^4}{K_1^4 + Z^4} - v_2 \frac{X}{K_2 + X} \tag{3}$$

$$\frac{dY}{dt} = k_3 X - v_4 \frac{Y}{K_4 + Y} \tag{4}$$

$$\frac{dZ}{dt} = k_5 Y - v_6 \frac{Z}{K_6 + Z} \tag{5}$$

1.3.1 The Model

1) Explain the different terms in the equations, and all the parameters. In particular explain the terms containing the fractions.

2) Based on the literature, discuss plausible genes/proteins that could represent the X, Y, Z variables.

3) Discuss/criticize the main assumptions of the model.

1.3.2 Simulation of the Model

4) Using the values $v_1 = 0.7\,nMh^{-1}, v_2 = v_4 = v_6 = 0.35\,nMh^{-1}, K_1 = K_2 = K_4 = K_6 = 1\,nM$ and $k_3 = k_5 = 0.7\,h^{-1}$, simulate the model: set appropriate initial conditions and time integration parameters to obtain a limit cycle. Plot some representative trajectories in 2D (x VS time, y VS time or z VS time) or 3D.

   Hint: to visualise the trajectories, plot only the last 5 % of the solution

[ ]:
```
#Solving the differential equations

Tmax=5000
dt=0.01
tspan = np.arange(0, Tmax, dt)
```

```
# Initial conditions setting
X0=[0.14,0.18,1.8]
```

[ ]:
```
#Definition of the model

def model(s, t):

    # add here the equations of the model

    return x_dot, y_dot, z_dot
```

5) Use Period_finder on a long (many periods) trajectory on x. Comment the period distribution you find and its point estimate. What happens if you use the y or z trajectory to evaluate the period?

   The Period_finder function takes as an imput the x (or y or z) vector of coordinates and a vector of equally spaced times at wich the given coordinate was obtained with a simulation. It takes the x limits of the plot as a possible input, with default values of 0 to 50. It returns a plot of the period distribution and a point estimate for the period. If you want to understand how this function was built, read "A. few words on the discrete Fourier transform".

   Hint: use the last 20% of the solution to find the period

[ ]:
```
def Period_finder(x, tspan, xlim=[0,50], ToPrint=True):

    signal = x

    omega = np.fft.fft(signal)

    modes = np.arange(omega.size)

    t_dist = (tspan[-1]-tspan[0])

    omega_cut = len(omega)//2

    modes = modes[1:omega_cut]
    omega = omega[1:omega_cut]

    periods = t_dist/modes

    abs_o = np.absolute(omega)
    max_o = np.argmax(abs_o)

    period_estimate = periods[max_o]

    if ToPrint:
        plot_period(period_estimate, periods, abs_o)
```

```
        print("the period value is about " + str(round(period_estimate,2)) + "␣
 ↪hours")

    return (period_estimate, periods, abs_o)

def plot_period(period_estimate, periods, abs_o,xlim=[0,50]):
    fig,ax= plt.subplots()

    ax.axvline(period_estimate, ls='--', c='k')

    ax.plot(periods,abs_o/np.sum(abs_o))

    ax.set_xlim(xlim)
    ax.set_xlabel("period")
    ax.set_ylabel("propobabilty density")

    return (fig ,ax)
```

Hopf bifurcation.

6) Vary the value of the transcription rate v_1 in the interval $(0, 5]$ nMh^{-1}. You can plot some representative trajectories (see the code before and replace the v_1). Plot and discuss the bifurcation diagram (show X_{min} and X_{max} in function of $v_1 \in (0, 5]$).

[ ]:
```
sample_frac=0.05

def model(s, t, v):
    # define the model
    return x_dot, y_dot, z_dot

f, axs = plt.subplots(4,5, figsize=(20,10))
axs=axs.flatten()
cmap = matplotlib.cm.viridis

# We define a range in which we vary v
vspan=np.linspace(0.1,5,20)

lastpart=int(len(tspan)*samp_frac)

X_lims=[]
for n,v in enumerate(vspan):
    # solve the system
    sol=odeint(model, X0, tspan, args=(v,))
    # plot representative trajectories x vs y

X_lims=np.array(X_lims)
```

```
# Bifurcation diagram
# plot the minimum and the maximum of the solution according to v
```

1.4 3. Entrainement of a generic non-linear oscillator (Python)

Let's take the model of exercise 2 and add an external periodic entrainement with period T as follows:

$$\frac{dX}{dt} = v_{1\_ent}(t) * \frac{K_1^4}{K_1^4 + Z^4} - v_2 \frac{X}{K_2 + X} \tag{6}$$

$$\frac{dY}{dt} = k_3 X - v_4 \frac{Y}{K_4 + Y} \tag{7}$$

$$\frac{dZ}{dt} = k_5 Y - v_6 \frac{Z}{K_6 + Z} \tag{8}$$

where $v_{1\_ent}(t) = v_1 * A(1 + \sin(\frac{2\pi}{T}t))$

1. Implement the entrainment in the model with $A = 0.01$ and $A = 0.2$. Is the oscilaltor stably entrained? If, what period do you expect then? Verify your prediction by simulation.

   Hint: to visualise the trajectories, plot only the last 5 % of the solution

[ ]:
```
T_external = 25
Tmax=T_external * 100
dt=0.01
tspan = np.arange(0, Tmax, dt)
# Initial conditions setting
X0=[0.14,0.18,1.8]
```

[ ]:
```
#Definition of the model with entrainment
def model(s, t, A, T_external=25):
    # define the model here
    return x_dot, y_dot, z_dot
```

2. Vary the value of the entrainment strength A in the interval $[0, 0.1]$. Which approximative value of A induces a change in period in the model? You can plot representative trajectories and period diagram using the *period_finder* provided function. Discuss your findings.

   Hint: to calculate the period, use the last 20 % of the solution and to visualise the trajectories, plot only the last 5 % of the solution

[ ]:
```
T_external=25
Tmax = T_external * 100
dt=0.01
tspan = np.arange(0, Tmax, dt)

# this tspan is used to calculate the period
tspan_80percent = int(0.8*len(tspan))
```

```
# this tspan is used for the plotting of the solution
tspan_5percent = int(0.05*len(tspan))

# Initial Conditions
X0=[0.15, 0.3, 1.8]
```

[ ]:
```
def model(s, t, A):
# define your model here
    return x_dot, y_dot, z_dot
# Define the range in which A vary
# A
for n,a in enumerate(vspan):
    # solve the system
    sol=odeint(model, X0, tspan, args=(a,))
# plot representative trajectories x vs y
# plot the period estimate using the period_finder function
```

3. To analyse the change of period in relation to the entrainment strength A, simulate the model without entrainment for a time $T0$ and then switch on the entrainment. Vary the strength of the entrainment as in 3.2. Discuss the effect of the strength of the entrainment on the period (compare the period before and after switching the entrainment). Discuss you findings.

   Hint: to calculate the period, here use the last 40 % of the solution and to visualise the trajectories, plot only the last 5 % of the solution

[ ]:
```
T_external=25

# Total time of the simulation of the model
Tmax = T_external * 100 #here we simulate a total of 100 cycles, you can also␣
 ↪change this value
dt=0.01

# Time after the entrainment should start
T0 = T_external * 40
n_entrain= int(T0 / dt)

# Initial Conditions
X0=[0.15, 0.3, 1.8]
```

[ ]:
```
def model(s, t, A):
# define your model here
    return x_dot, y_dot, z_dot
# Define the range in which A vary
# A
# Define the two timespan vectors for before and after entrainment
#
for n,a in enumerate(vspan):
```

```
# solve the system
    sol=odeint(model, X0, tspan, args=(a,))
# plot representative trajectories x vs y
# plot the period estimate using the period_finder function
```