Simularea Sistemelor Dinamice - Laborator 2

Curs: Prof. Dr. Marin Marin
Laborator: Asist. Dr. Adina Chirilă

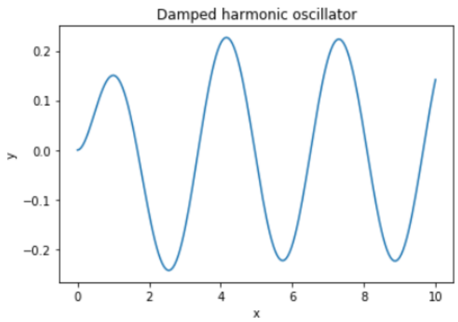
Contents

1 Ecuaţii diferenţiale de ordinul doi

Sa se rezolve urmatoarea ecuatie diferentiala de ordinul doi cu coeficienti constanti neomogena care modeleaza miscarea armonica simpla cu amortizare

$$\begin{aligned} &y'' + 2y' + 2y = cos(2x), x \in [0, 10] \\ &y(0) = 0, y'(0) = 0 \end{aligned} \tag{1}$$

```python
# Second-order ordinary differential equations
# The damped simple harmonic motion equation
# y''+2y'+2y=cos(2x), y(0)=0, y'(0)=0
# We can turn this into two first-order equations
# by defining a new dependent variable
# z=y',
# z'+2z+2y=cos(2x),
# z(0)=y(0)=0
# We can solve this system of ODEs
# using odeint with lists

# y'=z
# z'=-2z-2y+cos(2x)

import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def dU_dx(U, x):
    # Here U is a vector such that y=U[0] and z=U[1]. This function should return [y', z']
    return [U[1], -2*U[1] - 2*U[0] + np.cos(2*x)]

U0 = [0, 0]
xs = np.linspace(0, 10, 200)
Us = odeint(dU_dx, U0, xs)
ys = Us[:,0]

plt.xlabel("x")
plt.ylabel("y")
plt.title("Damped harmonic oscillator")
plt.plot(xs,ys);

# x''+g/L*sinx=0 (1)

# x'=y => x''=y'
# (1) => y'+g/L*sin(x)=0 => y'=-g/L*sin(x)

# x'=y                    U=[x,y] => U'=[x',y'], x=U[0], y=U[1]
# y'=-g/L*sin(x)          U'=[U[1], -g/L*sin(U[0])], x=x(t), y=y(t)

```

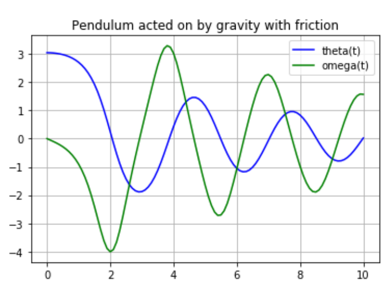


Sa se rezolve urmatoarea ecuatie diferentiala de ordinul doi cu coeficienti constanti neliniara pentru unghiul θ al unui pendul asupra căruia actioneaza forta de gravitatie cu frecare

$$\theta''(t) + b\theta'(t) + c\sin(\theta(t)) = 0, \tag{2}$$

unde b si c sunt constante pozitive.

```python
# The second order differential equation for the angle theta
# of a pendulum acted on by gravity with friction can be written
# theta''(t) + b*theta'(t) + c*sin(theta(t)) = 0
# where b and c are positive constants
# and a prime denotes a derivative
# We convert this equation to a system of first order equations
# We define the angular velocity omega(t)
# Therefore, we obtain the system
# theta'(t) = omega(t)
# omega'(t) = -b*omega(t) - c*sin(theta(t))
# Let y be the vector [theta, omega]

import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def pend(y, t, b, c):
    theta, omega = y
    dydt = [omega, -b*omega - c*np.sin(theta)]
    return dydt

b = 0.25
c = 5.0

# For initial conditions, we assume the pendulum is
# nearly vertical [theta(0)=pi-0.1] and is
# initially at rest [omega(0)=0]
y0 = [np.pi - 0.1, 0.0]

# We will generate a solution at 101 evenly spaced samples
# in the interval 0<=t<=10
t = np.linspace(0, 10, 101)              # array of times

# The solution is an array with shape (101, 2)
# The first column is theta(t)
# and the second is omega(t)
sol = odeint(pend, y0, t, args=(b, c))

plt.plot(t, sol[:, 0], 'b', label='theta(t)')
plt.plot(t, sol[:, 1], 'g', label='omega(t)')
plt.legend(loc='best')
plt.xlabel('t')
plt.grid()
plt.title("Pendulum acted on by gravity with friction")
plt.show()
```



Sa se rezolve urmatoarea ecuatie diferentiala de ordinul doi cu coeficienti variabili omogena care modeleaza oscilatorul Van der Pol

$$x'' - a(1 - x^2)x' + x = 0. \tag{3}$$

Aceasta ecuatie reprezinta un model pentru circuite electrice. In plus, modeleaza procese biologice precum bataia inimii si ritmurile circadiene.

Indicatie de rezolvare. Se noteaza $x' = y \Rightarrow x'' = y'$. Prin urmare, ecuatia se rescrie sub forma

$$y' - a(1 - x^2)y + x = 0 \Rightarrow y' = a(1 - x^2)y - x \tag{4}$$

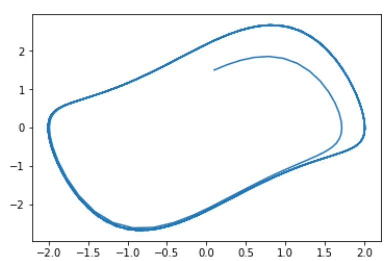
Rezulta ca ecuatia se scrie in mod echivalent sub forma urmatorului sistem de ecuatii diferentiale de ordinul intai

$$\begin{cases} x' = y \\ y' = a(1 - x^2)y - x \end{cases} \tag{5}$$

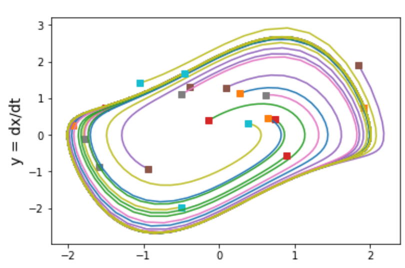
Se noteaza

$$Z = \begin{pmatrix} x \\ y \end{pmatrix} \Rightarrow x = Z[0], y = Z[1] \tag{6}$$

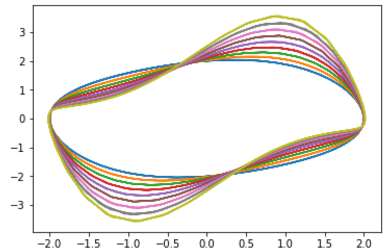
```python
# Oscilatorul Van der Pol
# Ecuatie diferentiala ordinara de ordinul doi cu parametrul a
# x''-a(1-x*x)x'+x=0
# |x|>1: pierde energie
# |x|<1: absoarbe energie
# model pentru circuite electrice
# modeleaza procese biologice precum bataia inimii si ritmurile circadiene
# x'=y
# y'=a(1-x*x)y-x

from scipy.integrate import odeint
import matplotlib.pyplot as plt

def dZ_dt(Z, t, a = 1.0):
    x, y = Z[0], Z[1]
    return [y, a*(1-x**2)*y - x]

def random_ic(scalefac=2.0): # generate initial condition
    return scalefac*(2.0*np.random.rand(2) - 1.0)

ts = np.linspace(0.0, 40.0, 400)
nlines = 20

c0=[0.1,1.5]
Ws = odeint(dZ_dt, c0, ts, args=(1.0,))
plt.plot(Ws[:,0], Ws[:,1])
plt.show()

for ic in [random_ic() for i in range(nlines)]:
    Zs = odeint(dZ_dt, ic, ts, args=(1.0,))
    plt.plot(Zs[:,0], Zs[:,1])
    plt.plot([Zs[0,0]],[Zs[0,1]], "s") # plot the first point

plt.xlabel("x", fontsize=14)
plt.ylabel("y = dx/dt", fontsize=14)

# All curves tend towards a limit cycle
```




```
Text(0,0.5,'y = dx/dt')
```

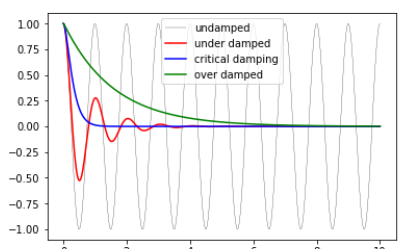


```python
# Investigate how the limit cycle varies with the parameter a

avals = np.arange(0.2, 2.0, 0.2) # parameters
minpt = int(len(ts) / 2) # look at late-time behaviour

for a in avals:
    Zs = odeint(dZ_dt, random_ic(), ts, args=(a,))
    plt.plot(Zs[minpt:,0], Zs[minpt:,1])
```



Sa se rezolve ecuatia care modeleaza oscilatorul armonic cu amortizare.

```python
from numpy import linspace
from math import pi
from scipy.integrate import odeint
from matplotlib.pyplot import subplots

def dy(y, t, zeta, w0):
    """
    The right-hand side of the damped oscillator ODE
    """
    x, p = y[0], y[1]

    dx = p
    dp = -2 * zeta * w0 * p - w0**2 * x

    return [dx, dp]
# initial state:
y0 = [1.0, 0.0]
# time coordinate to solve the ODE for
t = linspace(0, 10, 1000)
w0 = 2*pi*1.0
# solve the ODE problem for three different values of the damping ratio
y1 = odeint(dy, y0, t, args=(0.0, w0)) # undamped
y2 = odeint(dy, y0, t, args=(0.2, w0)) # under damped
y3 = odeint(dy, y0, t, args=(1.0, w0)) # critial damping
y4 = odeint(dy, y0, t, args=(5.0, w0)) # over damped
fig, ax = subplots()
ax.plot(t, y1[:,0], 'k', label="undamped", linewidth=0.25)
ax.plot(t, y2[:,0], 'r', label="under damped")
ax.plot(t, y3[:,0], 'b', label=r"critical damping")
ax.plot(t, y4[:,0], 'g', label="over damped")
ax.legend();
```



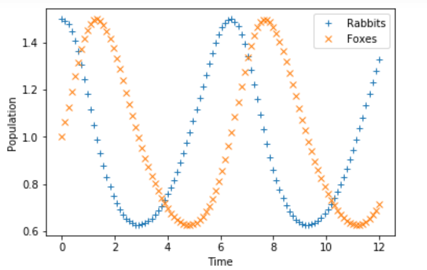
2 Sisteme dinamice

Ecuatiile Lotka-Volterra sau modelul pradator-prada reprezinta un sistem de doua ecuatii diferentiale de ordinul intai neliniare. Reprezinta un model simplificat pentru schimbarea populatiei a doua specii care interactioneaza prin pradare. Fie x - populatia de iepuri si y - populatia de vulpi. Atunci avem

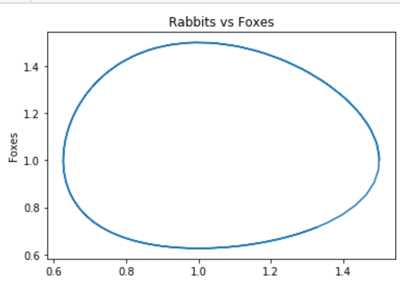
$$\begin{aligned} \frac{dx}{dt} &= x(a - by) \\ \frac{dy}{dt} &= -y(c - dx) \end{aligned} \tag{7}$$

unde a, b, c, d sunt parametri poziivi.

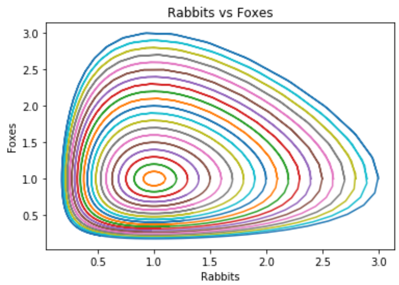
```python
# Ecuatiile Lotka-Volterra
# Modelul pradator-prada
# sistem de 2 EDO de ordinul intai neliniare
# reprezinta un model simplificat pentru schimbarea populatiei
# a doua specii care interactioneaza prin pradare
# exemple: vulpi (pradatori) si iepuri (prada)
# fie x - populatia de iepuri
# fie y - populatia de vulpi
# dx/dt=x(a-by)
# dy/dt=-y(c-dx)
# a, b, c, d, sunt parametri pozitivi

import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

a,b,c,d = 1,1,1,1

# P=[x, y], P[0]=x, P[1]=y
# a, b, c, d may be optional arguments
def dP_dt(P, t):
    return [P[0]*(a - b*P[1]), -P[1]*(c - d*P[0])]

ts = np.linspace(0, 12, 100)
P0 = [1.5, 1.0]              # initial conditions
Ps = odeint(dP_dt, P0, ts)
prey = Ps[:,0]               # first column
predators = Ps[:,1]          # second column

plt.plot(ts, prey, "+", label="Rabbits")
plt.plot(ts, predators, "x", label="Foxes")
plt.xlabel("Time")
plt.ylabel("Population")
plt.legend();
```




```python
# Phase portrait: plot x vs y (instead of x, y vs t)
# One curve for each initial condition
# Curves will not cross, in general

plt.plot(prey, predators, "-")
plt.xlabel("Rabbits")
plt.ylabel("Foxes")
plt.title("Rabbits vs Foxes");
```



```
# The plot above illustrates that the system is periodic.
# Let's plot a few more curves in the phase space.

ic = np.linspace(1.0, 3.0, 21)
for r in ic:
    P0 = [r, 1.0]
    Ps = odeint(dP_dt, P0, ts)
    plt.plot(Ps[:,0], Ps[:,1], "-")
plt.xlabel("Rabbits")
plt.ylabel("Foxes")
plt.title("Rabbits vs Foxes");

# Curves do not cross
# Closed curves <=> periodic solutions
# Equilibrium at x = y = 1 => dx/dt = dy/dt = 0
```

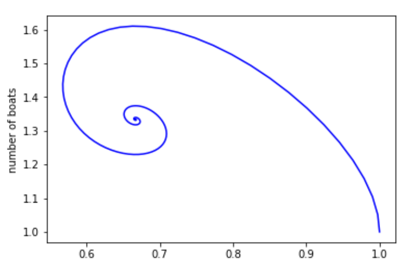


Consideram urmatorul model pradator-prada

$$\begin{aligned} \frac{dx}{dt} &= x(2 - y - x) \\ \frac{dy}{dt} &= -y(1 - 1.5x) \end{aligned} \tag{8}$$

unde x - numarul de pesti si y - numarul de barci de pescuit.

```
# Consider the predator-prey system of equations
# x - fish, y - fishing boats
# dx/dt=x(2-y-x)
# dy/dt=-y(1-1.5x)

import matplotlib.animation as animation
from scipy.integrate import odeint
from numpy import arange
from pylab import *

def BoatFishSystem(state, t):
    fish, boat = state
    d_fish = fish * (2 - boat - fish)
    d_boat = -boat * (1 - 1.5 * fish)
    return [d_fish, d_boat]

t = arange(0, 20, 0.1)
init_state = [1, 1]
state = odeint(BoatFishSystem, init_state, t)

xlabel('number of fish')
ylabel('number of boats')
plot(state[:, 0], state[:, 1], 'b-')
show()

# dx/dt=0 => y=2-x
# dy/dt=0 => x=2/3
```



Consideram modelul SEIR (Susceptible - Exposed - Infected - Removed) de transmitere a infectiei cu COVID, reprezentat de urmatorul sistem dinamic

$$\begin{aligned}
\frac{ds}{dt} &= -\gamma R_0 si \\
\frac{de}{dt} &= \gamma R_0 si - \sigma e \\
\frac{di}{dt} &= \sigma e - \gamma i \\
\frac{dr}{dt} &= \gamma i
\end{aligned} \tag{9}$$

unde σ - rata de infectare (rata la care cei expusi devin infectati), R_0 are legatura cu rata de transmitere si γ - rata de vindecare (rata la care cei infectati se vindeca sau mor).

```python
# quantitative modeling of infectious disease dynamics
# dynamics are modeled using a standard SEIR
# (Susceptible-Exposed-Infected-Removed) model of disease spread
# The states are: susceptible (S), exposed (E), infected (I) and removed (R).
# β(t) is called the transmission rate or effective contact rate
# (the rate at which individuals bump into others and expose them to the virus).
# σ is called the infection rate (the rate at which those who are exposed become infected)
# γ is called the recovery rate (the rate at which infected people recover or die)
# https://julia.quantecon.org/continuous_time/seir_model.html

import matplotlib.animation as animation
from scipy.integrate import odeint
from numpy import arange
from pylab import *

def CovidSystem(state, t):
    s, e, i, r = state
    γ = 1/18
    R_0 = 3.0                   # basic reproduction number for the SEIR model
    σ = 1/5.2

    d_s = -γ*R_0*s*i            # ds/dt = -γR0si
    d_e = γ*R_0*s*i -  σ*e      # de/dt =  γR0si -σe
    d_i = σ*e - γ*i            # di/dt =        σe -γi
    d_r = γ*i                  # dr/dt =            γi

    return [d_s, d_e, d_i, d_r]

t = arange(0.0, 350.0)        # ≈ 350 days

i_0 = 1E-7                    # 33 = 1E-7 * 330 million population = initially infected
e_0 = 4.0 * i_0               # 132 = 1E-7 *330 million = initially exposed
s_0 = 1.0 - i_0 - e_0
r_0 = 0.0

init_state = [s_0, e_0, i_0, r_0]

state = odeint(CovidSystem, init_state, t)

plot(t, state[:, 0],'r-',linewidth=2, label='susceptible')
plot(t, state[:, 1],'b--',linewidth=2, label='exposed')
plot(t, state[:, 2],'g:',linewidth=2, label='infected')
plot(t, state[:, 3],'y*',linewidth=2, label='removed')
legend()
show()
```