

Modeling and Simulation

Dr. Belkacem KHALDI

b.khaldi@esi-sba.dz

ESI-SBA, Algeria

Level: 2nd SC Class

Date: March 15, 2023

Modeling Dynamical Systems

General Introduction to Dynamical Systems

What is a dynamical system?

- A **dynamical system** is any system that varies through time.
- We used to describe such a system by state $x(t)$
- The state x can be multivariate (a vector)

$$x(t) = (x_1(t), \dots, x_n(t))^T$$

Some examples of dynamical systems:

- Ecological models (e.g., changes in population size)
- Chemistry (change in number of molecules during chemical reactions)
- Climate science (e.g., change in temperature through time)
- Economics models (e.g., change in stock prices)
- ...

Modeling Dynamical Systems

Types of models for Dynamical Systems

Two classes of description for dynamical systems:

- **Discrete-time dynamical system:** Discrete-time models assume that time proceeds in a step-wise manner. These dynamical systems are described by recurrence relations e.g.:

$$x_t = f(x_{t-1}, t)$$

- **Continuous-time dynamical system:** Continuous-time models assume that time is a real number (technically: “infinitesimally” small). These systems are described by differential equations e.g.:

$$\frac{dx}{dt} = f(x, t)$$

In either case, x_t or x is the state variable of the system at time t . f is a function that determines the rules by which the system changes its state over time

Modeling Dynamical Systems

Simulating Discrete-Time Systems

- A good starting point to understand discrete-time dynamical models are sequences of numbers that follow certain rules.
- **Example:**

x_0	x_1	x_2	x_3	x_4	x_5
1	-2	4	-8	16	-32

Can you guess the underlying rule for these sequences (The recursion equation f that describes the relationship between x_t and x_{t-1}) ?

$$x_i = f(x_{i-1})$$

Solution:

$$x_i = -2x_{i-1}, \text{ with } x_0 = 1$$

- The example above belongs to a specific type of Discrete-Time Dynamical systems called **First-order Linear system**.

Simulating Discrete-Time First-order Systems

- **First-order system:** A difference equation whose rules involve state variables of the immediate past time (at time $t - 1$).

Theorem 1.

The general solution of the first-order linear difference equation:

$$x_t = a.x_{t-1} + b, t = 0, 1, 2, 3, \dots,$$

is given by:

$$x_t = a^t.x_0 + \begin{cases} \frac{a^n-1}{a-1}.b, & \text{if } a \neq 1 \\ n.b, & \text{if } a = 1 \end{cases}$$

Modeling Dynamical Systems

Simulating Discrete-Time First-order Systems

Example:

- Now we want to simulate a simple exponential population growth model governed by the following difference equation model:

$$N_t = a.N_{t-1}$$

Where $a = 1.1$ is the rate of population generation and N_t is the number of individuals in generation at time t . The initial generation $N_0 = 1$.

- Our objective is to find out what kind of behavior this model will show through computer simulation.
- The general solution of the above example is:

$$N_t = a^t.N_0 + \begin{cases} \frac{a^n-1}{a-1}.b, & \text{if } a \neq 1 \\ n.b, & \text{if } a = 1 \end{cases} \xrightarrow{0} = a^t.N_0$$

Modeling Dynamical Systems

Simulating Discrete-Time First-order Systems - Mathematical Solution

We use the `sympy` python library to solve this system (<https://www.sympy.org/>)

```
1 from sympy import Function, rsolve
2 from sympy.abc import n
3
4 #rate of population generation
5 a=1.1
6 #Functions Declarations
7 N = Function('N')
8 f = N(t+1) - a * N(t)
9 #Solving the equation
10 sol = rsolve(f, N(t), {N(0):1})
11 #Printing the solution
12 print('N_t = {}'.format(sol))
13 N_5 = round(sol.subs(t, 5), 2)
14 print('N(5) = {:,}'.format(N_5))
```

$$N_t = 1.1^{**t}$$
$$N(5) = 1.61$$

Modeling Dynamical Systems

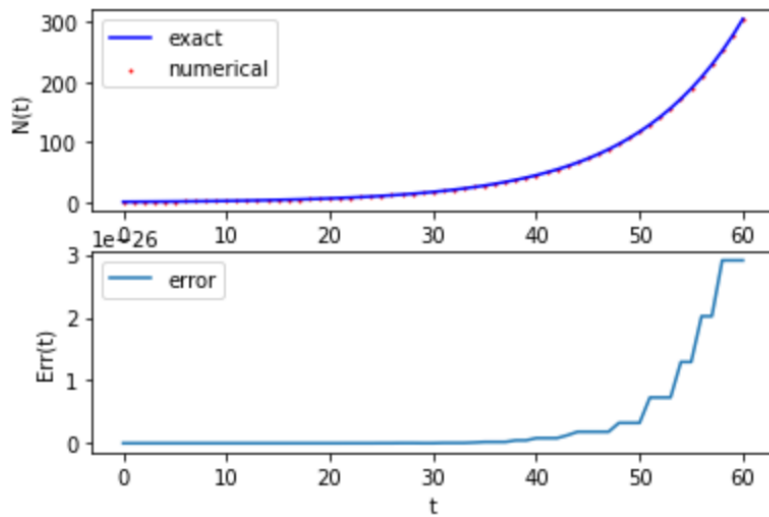
Simulating Discrete-Time First-order Systems - Numerical Solution

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 N_0 = 1.0 #initial gen.
5 a = 1.1 #rate of pop. gen.
6 N_t = 59 #Max Time steps
7 #generate time steps
8 t = np.linspace(0, (N_t+1), (
    N_t+2))
9 #Initialize N with 0
10 N = np.zeros(N_t+2)
11 #Set N[0] to N_0
12 N[0] = N_0
13 #Exact Solution
14 E=N_0*np.power(a,t)
15 #Solve Numerically
16 for n in range(N_t+1):
17     N[n+1] = N[n] * a
```

```
18 #Plotting(solution)
19 fig, axs = plt.subplots(2)
20 axs[0].scatter(x=t, y=N, c=
    'r', s=1, label='
    numerical')
21 axs[0].plot(t,E , 'b',label=
    'exact')
22 axs[0].legend(loc='upper
    left')
23 axs[0].set(xlabel='t',
    ylabel='N(t)')
24 axs[1].plot(t,(E-N)**2,label=
    ='error')
25 axs[1].legend(loc='upper
    left')
26 axs[1].set(xlabel='t',
    ylabel='Err(t)')
27 plt.show()
```

Modeling Dynamical Systems

Simulating Discrete-Time First-order Systems - Numerical Solution



Modeling Dynamical Systems

Simulating Continuous Time dynamical Systems

- Continuous Time Dynamical systems are usually described using differential equation models, specifically ordinary differential equations (ODE) and Partially Differential Equations (PDE)
- Probably more mainstream in science and engineering, and studied more extensively, than discrete-time models
- Can be used to model/simulate various natural phenomena (e.g., motion of objects, flow of electric current).
- In this course, we are interested in systems that are modeled using ODE.

Modeling Dynamical Systems

Ordinary differential equations (ODEs)

ODEs are a type of differential equations that have:

- One **independent** variable, t
- May be several **dependent** variables, x_i :

$$x_i = \{x_1(t), x_2(t), \dots\},$$

- and a set of functions f_i relating x_i and its derivatives,

$$f_i(x_i, \dot{x}_i, \ddot{x}_i, \dots; t) = 0$$

where:

$$\dot{x}_i = \frac{dx_i}{dt}, \ddot{x}_i = \frac{d^2x_i}{dt^2}, \dots, \text{etc}$$

Features

- **Order**: refers to the highest derivative: k th order $\Leftrightarrow \frac{d^k x}{dt^k}$
- **Dimension**: refers to the number of dependent variables $x = [x_1 \dots x_d]$, and the number of independent equations.
- **Autonomous**: $\Leftrightarrow f_i$ have no explicit dependence on t .
- **Linear**: if f_i has only linear dependence on x_i, \dot{x}_i, \dots and their combinations. Otherwise it is non-linear.

Modeling Dynamical Systems

Ordinary differential equations (ODEs)- Example

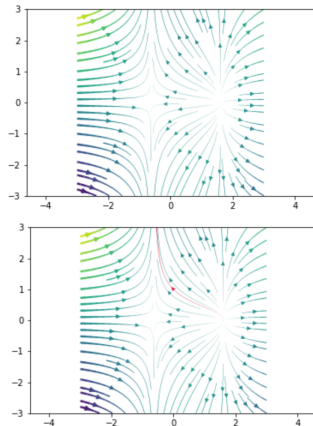
Consider a system of the following ODE:

$$\frac{dx}{dt} = x^2 - x - 1,$$

This system is:

- **First-order**, as $\frac{dx}{dt}$ is the highest derivative.
- **One-dimensional**, as x is the only dependent variable.
- **Autonomous**, as the rate of change $\frac{dx}{dt}$ does not depend on the independent variable t .
- **Non-linear**, because of the non-linear term x^2 on the right-hand side.

To find one solution among many, we specify an initial condition, e.g. $x(0) = 1$



Modeling Dynamical Systems

Ordinary differential equations (ODEs) - 1D Autonomous Equation

Consider the 1st-order autonomous case:

$$\frac{dx}{dt} = f(x),$$

- A solution is typically found by **separation of variables**.
- Divide by $f(x)$ and integrate.

$$\int \frac{dx}{f(x)} = t + c$$

- Some cases can be solved exactly, e.g,

$$f(x) = x \Rightarrow \ln(x) = t + c \Rightarrow x(t) = C_1 e^t$$

- In some cases integral can't be found analytically \Rightarrow Solve Numerically.

Modeling Dynamical Systems

ODEs - 1D Autonomous Equation - Example 1

Consider the **simple exponential population growth system** modeled by the 1st-order autonomous model:

$$\frac{dx}{dt} = x,$$

- A general mathematical solution for this system can be found using `sympy` python library:

$$x(t) = C_1 e^t$$

- To specify a specific solution, we have to specify an initial condition.

Mathematical Solution for $f(x)=x$

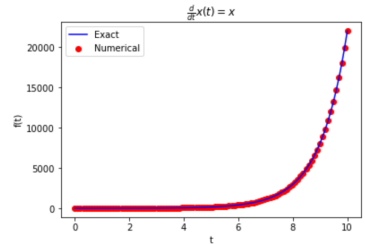
```
1 import sympy as sy
2 sy.init_printing()
3 # declare math. symbols
  used in Eq
4 t = sy.symbols('t')
5 f = sy.Function('x')
6 # Solve the equation f'(t)
  = x(t)
7 diffEq = f(t).diff(t) - f(
  t)
8 sy.dsolve(diffEq, f(t))
```

$$x(t) = C_1 e^t$$

Modeling Dynamical Systems

ODEs - 1D Autonomous Equation - Example 1 (Numerical Solution)

```
1 import matplotlib.pyplot as plt
2 from scipy.integrate import odeint
3 def PopGrowth(x, t):
4     #Returns the gradient dx/dt for
5     the exponential equation
6     return x
7 ts = np.linspace(0.0, 10.0, 100) #
8 values of independent variable
9 x0 = 1 # an initial condition, x(0) =
10 x0
11 y = odeint(PopGrowth, x0, ts)
12 # odeint returns an array of x values
13 , at the times in ts.
14 plt.xlabel('t')
15 plt.ylabel('f(t)')
16 plt.scatter(ts,y, label='Numerical',
17             color='r')
18 plt.legend()
```



Modeling Dynamical Systems

ODEs - 1D Autonomous Equation - Example 2

Consider the **logistic population growth system** modeled by the 1st-order autonomous model:

$$\frac{dx}{dt} = x(1 - x),$$

- The general mathematical solution for this system using **sympy** python library is:

$$x(t) = \frac{1}{C_1 e^{-t} + 1}$$

- To specify a specific solution, we have to specify an initial condition.

Mathematical Solution for $f(x)=x(1-x)$

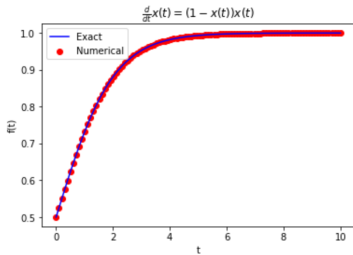
```
1 import sympy as sy
2 sy.init_printing()
3 # declare math. symbols
  used in Eq
4 t = sy.symbols('t')
5 f = sy.Function('x')
6 # Solve the equation f'(t)
  = f(t)(1-f(t)).
7 diffEq = f(t).diff(t) - f(t)
  *(1-f(t))
8 sy.solve(diffEq, f(t))
```

$$x(t) = \frac{1}{C_1 e^{-t} + 1}$$

Modeling Dynamical Systems

ODEs - 1D Autonomous Equation - Example 2 (Numerical Solution (1))

```
1 import matplotlib.pyplot as plt
2 from scipy.integrate import odeint
3 def LogGrowth(x, t):
4     #Returns the gradient dx/dt for
5     #the Logistic equation
6     return x*(1-x)
7 ts = np.linspace(0.0, 10.0, 100) #
8 # values of independent variable
9 x0 = 0.5 # an initial condition, x(0)
10 = x0
11 y = odeint(LogGrowth, x0, ts)
12 # odeint returns an array of x values
13 # , at the times in ts.
14 plt.xlabel('t')
15 plt.ylabel('f(t)')
16 plt.scatter(ts,y, label='Numerical',
17             color='r')
18 plt.legend()
```



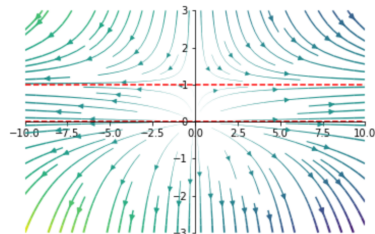
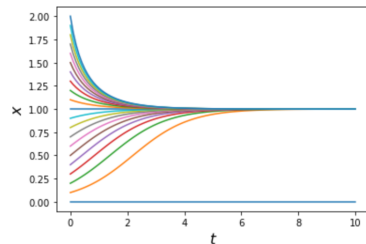
- Here $x_0 = 0.5$.

Modeling Dynamical Systems

ODEs - 1D Autonomous Equation - Example 2 (Numerical Solution (2))

- Let's plot curves for several initial conditions

```
1 # Plot curves for several initial
  conditions
2 ics = np.linspace(0.0, 2.0, 21) # a
  list of initial conditions
3 for x0 in ics:
4     xs = odeint(LogGrowth, x0, ts)
5     plt.plot(ts, xs)
6 plt.xlabel('$t$', fontsize=16);
7 plt.ylabel('$x$', fontsize=16)
```



- Two equilibrium positions: $x = 0$ and $x = 1$.
- $x = 0$ is an **unstable** equilibrium.
- $x = 1$ is a **stable** equilibrium.

Modeling Dynamical Systems

Ordinary differential equations (ODEs) - 2D autonomous equations

Now consider a first order system with two dependent variables, x and y ,

$$\begin{cases} \frac{dx}{dt} = f(x, y; t), \\ \frac{dy}{dt} = g(x, y; t), \end{cases}$$

- System is **autonomous** if f and g do not depend on t .
- **Example:** Modelling the populations of rabbits and foxes.



Modeling Dynamical Systems

ODEs - 2D Autonomous Equations - Example

Consider the **Predator-prey equations** example which is also known as **Lotka-Volterra equations**.

- the predator-prey equations are a pair of coupled first-order non-linear ordinary differential equations.
- They represent a simplified model of the change in populations of two species which interact via predation. For example, foxes (predators) and rabbits (prey).
- Let x and y represent rabbit and fox populations, respectively. Then:

$$\begin{cases} \frac{dx}{dt} = ax - bxy, \\ \frac{dy}{dt} = -cy + dxy, \end{cases}$$

Where a, b, c and d are parameters, which are assumed to be positive.

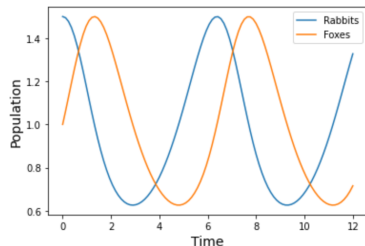
ODEs - 2D Autonomous Equations - Predator-Prey Equations (Numerical Solution)

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.integrate import odeint
4
5 def predprey(Z, t, a=1, b=1, c=1, d=1): # a,b,c,d
6     optional arguments.
7     x, y = Z[0], Z[1]
8     dxdt, dydt = x*(a - b*y), -y*(c - d*x)
9     return [dxdt, dydt]
10
11 ts = np.linspace(0, 12, 100)
12 Z0 = [1.5, 1.0] # initial conditions for x and y
13 Sol = odeint(predprey, Z0, ts, args=(1,1,1,1))
14 # use optional argument 'args' to pass parameters to
15     dZ_dt
16
17 prey = Sol[:,0] # first column
18 predators = Sol[:,1] # second column
```

Modeling Dynamical Systems

ODEs - 2D Autonomous Equations - Predator-Prey Equations (Numerical Solution)

```
1 #Let's plot 'rabbit' and 'fox' populations as a function  
  of time  
2 plt.plot(ts, prey, "+", label="Rabbits")  
3 plt.plot(ts, predators, "x", label="Foxes")  
4 plt.xlabel("Time", fontsize=14)  
5 plt.ylabel("Population", fontsize=14)  
6 plt.legend();
```



$$\begin{cases} \frac{dx}{dt} = ax - bxy, \\ \frac{dy}{dt} = -cy + dxy, \end{cases}$$

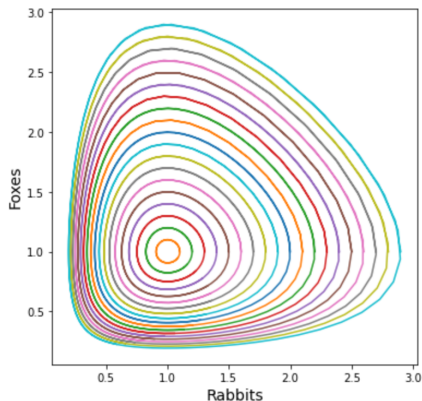
ODEs - 2D Autonomous Equations - Predator-prey equations: Phase plot

- The ODEs are autonomous: no explicit dependence on t
- Phase portrait: Plot x vs y (instead of x,y vs t).
- One curve for each initial condition
- Curves will not cross (typically) for an autonomous system.

```
1 fig = plt.figure()
2 fig.set_size_inches(6,6) # Square plot, 1:1 aspect ratio
3 ics = np.arange(1.0, 3.0, 0.1) # initial conditions
4 for r in ics:
5     Z0 = [r, 1.0]
6     Sol = odeint(predprey, Z0, ts)
7     plt.plot(Sol[:,0], Sol[:,1])
8 plt.xlabel("Rabbits", fontsize=14)
9 plt.ylabel("Foxes", fontsize=14)
```


Modeling Dynamical Systems

ODEs - 2D Autonomous Equations - Predator-prey equations: Phase plot



- Curves do not cross
- Closed curves: Periodic solutions
- Equilibrium at:
- $x = y = 1 \implies \dot{x} = \dot{y} = 0$

ODEs - Second-Order Systems - Example: The Van der Pol oscillator

- In dynamics, the Van der Pol oscillator is a non-conservative oscillator with non-linear damping. It evolves in time according to the following ODE:

$$\frac{d^2x}{dt^2} - a \left(1 - x^2(t)\right) \frac{dx}{dt} + x(t) = 0$$

or

$$\ddot{x} - a(1 - x^2)\dot{x} + x = 0$$

- This is a 2nd-order ODE with one parameter, a
 - $|x| > 1$: loses energy
 - $|x| < 1$: absorbs energy
- Originally, used as a model for an electric circuit with a vacuum tube.
- Used to model biological processes such as heart beat, circadian rhythms, biochemical oscillators, and pacemaker neuron

ODEs - Second-Order Systems - Example: The Van der Pol oscillator

$$\ddot{x} - a(1 - x^2)\dot{x} + x = 0$$

First-order reduction:

Any second-order equation can be written as two coupled first-order equations, by introducing a new variable.

- Let $y = \frac{dx}{dt}$. Then

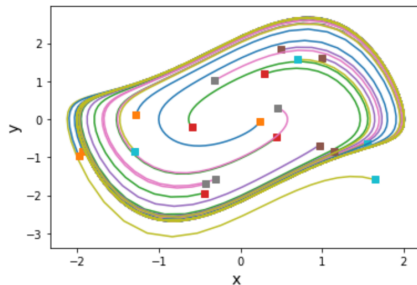
$$\dot{x} = y$$

$$\dot{y} = a(1 - x^2)y - x$$

Modeling Dynamical Systems

ODEs - Second-Order Systems - Example: The Van der Pol oscillator

```
1 def Van_der_Pol (Z, t, a = 1.0):
2     x, y = Z[0], Z[1]
3     dxdt = y
4     dydt = a*(1-x**2)*y - x
5     return [dxdt, dydt]
6 def random_ic(scalefac=2.0): #
7     stochastic initial condition
8     return scalefac*(2.0*np.random
9         .rand(2) - 1.0)
10
11 ts = np.linspace(0.0, 40.0, 400)
12 nlines = 20
13 for ic in [random_ic() for i in
14     range(nlines)]:
15     Zs = odeint(Van_der_Pol, ic,
16         ts)
17     plt.plot(Zs[:,0], Zs[:,1])
18     plt.plot([Zs[0,0]], [Zs[0,1]],
19         's') # plot the first point
```



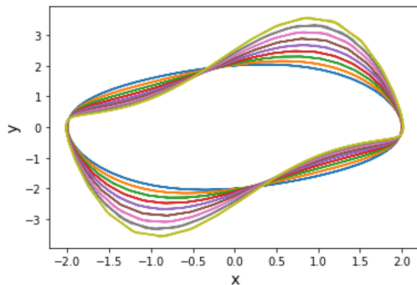
- All curves tend towards a limit cycle

Modeling Dynamical Systems

ODEs - Second-Order Systems - Example: The Van der Pol oscillator

- Investigate how the **limit cycle** varies with the parameter **a**:

```
1 avals = np.arange(0.2, 2.0, 0.2) #  
  parameters  
2 minpt = int(len(ts) / 2) # look at  
  late-time behaviour  
3  
4 for a in avals:  
5     Zs = odeint(Van_der_Pol,  
6                 random_ic(), ts, args=(a,))  
7     plt.plot(Zs[minpt:,0], Zs[  
8             minpt:,1])  
9  
10 plt.xlabel("x", fontsize =14)  
11 plt.ylabel("y",  fontsize =14)
```



End of Lecture 4:

Modeling Dynamical Systems