1 Differential equation (ODE)

1.1 Intro

Differential equation are equations that link a function and its derivatives. The simplest differential equation is :

$$y'(t) = y(t)$$

To solve this equation, we are looking for a function y, that derivative is equals to y. The only known functions that have this property are the exponential functions $y(t) = C\exp(t)$, with C a constant. Here the solution is a family of function (you can change the value of C to have different solutions). In order to have a unique solution you need to add an information, for example, the value of the function at $t = 0$:

$$y'(t) = y(t) \text{ and } y(0) = 2$$

Here the solution is $y(t) = 2\exp(t)$ (and it is the only solution).

1.2 Solving ODE with Python

Differential equations can be solved in Python with the `scipy.integrate` package using the function `odeint`. To solve differential equation with Python you need to follow the correct syntax.

```
y = odeint(model, y0, t)
```

- model : Function name that returns derivative values at requested y and t values as `dydt = model(y,t)` (in the previous example `model(y,t)= y`)
- y0 : Initial conditions of the differential states (to have a unique solution, in the previous example `y0 = 2`)
- t : Time points at which the solution should be reported. (Additional internal points are often calculated to maintain accuracy of the solution but are not reported.)
- y : return of `odeint`, the solution of the differential equation with condition y0 computed at time points t

An example of using `odeint` is with the following differential equation with parameter $k = 0.3$, the initial condition y0=5 and the following differential equation.

$$\frac{dy}{dt} = -ky(t)$$

1. The Python code first imports the needed Numpy, Scipy, and Matplotlib packages. The model, initial conditions, and time points are defined as inputs to `odeint` to numerically calculate the solution y at the asked time points.

   Copy the following code, and run it.

   ```
   import numpy as np
   from scipy.integrate import odeint
   ```

```
import matplotlib.pyplot as plt

# function that returns dy/dt
def model(y,t):
    k = 0.3
    dydt = -k * y
    return dydt

# initial condition
y0 = 5

# time points
t = np.linspace(0,20) #by default size of linspace is 50

# solve ODE
y = odeint(model,y0,t)

# plot results : the solution over time
plt.plot(t,y)
plt.xlabel('time')
plt.ylabel('y(t)')
plt.show()
```

2. An optional fourth input is `args` that allows additional information to be passed into the model function. The `args` input is a tuple sequence of values. The argument k is now an input to the model function by including an addition argument. Copy the following codes, and run them to understand how `odeint` works.

```
# function that returns dy/dt
def model(y,t,k):
    dydt = -k * y
    return dydt

# initial condition
y0 = 5

# time points
t = np.linspace(0,20)

# solve ODEs for different values of parameter k (same initial condition)
k = 0.1
y1 = odeint(model,y0,t,args=(k,))
k = 0.2
y2 = odeint(model,y0,t,args=(k,))
k = 0.5
y3 = odeint(model,y0,t,args=(k,))

# plot results
plt.plot(t,y1,'r-',linewidth=2,label='k=0.1')
plt.plot(t,y2,'b--',linewidth=2,label='k=0.2')
plt.plot(t,y3,'g:',linewidth=2,label='k=0.5')
```

```
plt.xlabel('time')
plt.ylabel('y(t)')
plt.legend()
plt.show()
```

2 Simple models

1. Play with the previous model by changing the initial condition.
2. Solve the differential equation $\frac{dy}{dt} = -ky^2(t)$ with any initial condition.
3. We want to do 2D models by using this function (now y is a vector of size 2) :

```
def model(y,t):
  return [-y[0],-y[1]]
```

   How do we change the code to make it work ? What about the initial condition ?

4. Solve the following ODE system with any initial condition :

$$\begin{cases} \frac{dA}{dt} &= A - B \\ \\ \frac{dB}{dt} &= B - A \end{cases}$$

3 Real models

3.1 Predatory prey - Lotka-Volterra model

We consider the following system :

$$\begin{cases} \frac{dA}{dt} &= rA - \mu AB \\ \\ \frac{dB}{dt} &= \rho AB - \lambda B \end{cases}$$

All the parameters (r, μ, ρ and λ are positive) :

— r is the reproduction rate of A - my prey - unit : $individual.time^{-1}$
— μ is the probability of capture of the prey by the predator (unit : $individual^{-1}.time^{-1}$)
— ρ is the reproduction of prey after capture (unit : $individual^{-1}.time^{-1}$)
— λ is the death rate of predators ($time^{-1}$).

1. Code the system / view and try different parameters to see how the system is evolving. Do not hesitate to increase the time interval (and to include several time points).
2. Print out the maximum/minimum of each species during time courses.
3. You should obtain periodic solutions. Can you compute the period ? How will you do ?

3.2 Epidemic model - SIR

We consider the following system (it's in dimension 3 !) :

$$\begin{cases} \frac{dS}{dt} &= -\beta SI \\ \frac{dI}{dt} &= \beta SI - \alpha I \\ \frac{dR}{dt} &= \alpha I \end{cases}$$

Parameters (α, β) are positive. In this model, S represents the proportion of persons who are Susceptible to the disease, I the proportion of person who are Infected and R the proportion of persons who have recovered (and cannot be infected anymore).

1. Code and solve the system (initial condition $S(0) = 0.9$, $I(0) = 0.1$, $R(0) = 0$).
2. View and try different parameter values to see how the system is evolving over time.

3.3 Chaotic Attractor Lorentz

We consider the following system (it's in dimension 3 !) :

$$\begin{cases} \frac{dx}{dt} &= \sigma(y - x) \\ \frac{dy}{dt} &= x(\rho - z) - y \\ \frac{dz}{dt} &= xy - \beta z \end{cases}$$

1. Take $\sigma = 10$, $\beta = \frac{8}{3}$ and $\rho = 28$ (Lorentz's values) and display a trajectory over a large time interval.
2. Try to visualize in 3D.
3. Take $(1, 0, 0)$ as initial conditions and $(1 + \epsilon, 0, 0)$ as another. Simulate the two trajectories over a period of time and plot the distance between the two. What happens ?

4 Data fitting

Subject : use Python to fit a model with experimental data
Targets : write and simulate a model to describe a phenomena, and find model parameters to reproduce experimental data

In the previous section, we solved a model $\frac{dA}{dt} = k\,A(t)$ for a given value of k. The question here is how to choose the value of k so that my solution is close to the experimental observations. We can estimate the value of k from the data. Here we describe a general method to estimate parameter values from data with Python.

We assume the data are (t_i, y_i) with $t = (t_i)$ a time vector (for example) and $y = (y_i)$ the observed variable (or vector of variables) at each time t_i (with $i \in \{0, \dots N-1\}$). We assume that the system can be describe by a function $f(t, \theta)$ with θ a parameter (or a vector of parameters) of this function. **Fit the model to data is finding the 'best' θ to explain the data with the model.**

1. Simple case
   Data : we assume we have the following measurements :

```
 0 1.0018
 1.00 0.6115
 2.00 0.3281
 3.00 0.2040
 4.00 0.1370
 5.00 0.0833
 6.00 0.0411
 7.00 -0.0365
 8.00 -0.0402
 9.00 0.0327
10.00 0.0281
11.00 -0.0381
12.00 0.0890
13.00 0.0027
14.00 0.0741
15.00 -0.0343
16.00 0.0805
17.00 0.0131
18.00 0.0328
19.00 -0.0637
20.00 0.0043
```

   (a) Copy and paste the data in a file that you can read and format with Numpy

```
import numpy as np
data = np.fromfile('fichier.txt', sep='\n', dtype=float)
data = data.reshape((-1, 2))
```

   other option :

```
import numpy as np
data = np.loadtxt('fichier.txt')
```

   (b) Plot the data - try to also plot $\log(y)$ with respect to t (hints : `semilogy` or function log) - does it work ?

   (c) Linear and polynomial fits are available in `Numpy`. Can you find a slope for the log of the data ? (use the help of `Numpy` functions).

```
logy = np.log(y)
p = np.polyfit(t,logy,1) # slope is p[0]
```

(d) It seems you can obtain a linear fit of the log of the data : it suggest that the data follows an exponential function $y(t) = \exp(kt)$. Can you find k (hints : previous slope) ?

Here we will choose to have a positive parameter k so we will now consider the function $y(t) = \exp(-kt)$.

(e) Compare the data and the exponential function that you found. The two curves are close, but in what sens ? Write a function *sse* that computes the square distance between your function and the data

$$sse(k) = \sum_{i=1}^{N} (y_i - \exp(-kt_i))^2$$

(f) With `scipy.optimize.fmin` - check the help - find the 'best' k, *i.e* the value of k that minimize the square distance between the data and the curve $t \mapsto \exp(-kt)$. Test different initial value for k (`fmin` implement an optimization algorithm that is local, so it will be sensitive to the initial value of k).

(g) Plot your model for the different estimation of k and conclude.

2. In order to train yourself you are going to estimate parameter with different type of function. To do so, we are going to create data (also called synthetic data - different from measured data). Synthetic data are created with the model function adding noise.

(a) Implement a function $f(x, k) = x^r \exp(-k * x)$

(b) Choose a value for r and k, and create data for a vector $x = [0, 10]$ of size N (N is a parameter that can be changed).

(c) To add noise, use the function `np.random.randn` which creates a vector of random values following a standard normal distribution ($\mathcal{N}(0, 1)$). Choose the level of noise with the standard deviation value σ (parameter that can be changed) and keep the mean at value zero. Create your vector of noisy data.

(d) With the previous methods estimate the values of k and r. Are you close to the values that you choose to generate the synthetic data ?

(e) Change the initial guess for k and r, the level of noise σ or the size of the data vector N, what do you observe ?

(f) *To continue.* You can compute an error of your estimate (since with synthetic data you know the true value of the parameters) : $\mathcal{E} = \sum_i \frac{ko_i - k_i}{k_i}$ with ko the vector of estimated parameters and k the true parameter vector.

— Fixe N and compute the error for different values of σ. Plot the error with respect to σ.

— Fixe σ and compute the error for different values of N. Plot the error with respect to N.

— What do you observe ?

Hints : like `odeint`, `fmin` accept arguments `args=()` to send extra argument to `fmin` (for example the data you want to fit).