Activity 04 : *Numerical resolution of Ordinary Differential Equations (ODE)*

Euler Method & Runge-Kutta Method

1. Numerical differentiation :

a. Within your folder, create a new notebook and name it: ***TP04.ipynb***

b. Start your notebook by the first cell of importing necessary modules:

```python
import numpy as np
import matplotlib.pyplot as plt
```

c. After that, within a new cell code, write the program allowing to compute the differentiation of a given function $\boldsymbol{f(x) = \cos(x)}$ with forward, backward, and central method

```python
n = 10
h = np.pi/n #0.1

# define grid
x = np.linspace(-h, np.pi, n+2)

# function
y = np.cos(x)

# use np.diff
forward_diff = np.diff(y)[1::]/h
backward_diff = np.diff(y)[:-1:]/h
central_diff = (y[2:] - y[:-2])/(2*h)

# grids
x_forw = x[1:-1:]
x_back = x[1:-1:]
x_central = x[1:-1:]

# compute exact solution
exact_solution = -np.sin(x_forw)
```

d. Add the following cell code to visualize the performed computing

```python
# Plot solution
plt.figure(figsize = (12, 8))
plt.plot(x, y, c='b', marker="o", ms=6, label = 'Primary function cos(x)')
plt.plot(x_forw, forward_diff, ls='', marker='o', ms=5, c='r', label = 'forward diff')
plt.plot(x_back, backward_diff, ls='', marker='o', ms=5, c='g', label = 'backward diff')
plt.plot(x_central, central_diff, ls='', marker='o', ms=5, mfc='w', mec='k', zorder=10,label = 'central diff')
plt.plot(x,-np.sin(x), c='k', ls='--', label = 'exact solution -sin(x)')
plt.legend(labelcolor=['b', 'r', 'g', 'k', 'k'])
plt.grid()
plt.show()

# max error
max_error = max(abs(exact_solution - forward_diff))
print('maximal error = ', max_error)
```

Activity 04 : *Numerical resolution of Ordinary Differential Equations (ODE)*

Euler Method & Runge-Kutta Method

2. **Euler Method :** the following code use Euler method to resolve the following ODE:

$$\frac{dy}{dt} = f(t, y(t))$$

Try it with N=10, 20, 50, and 100.

```
def euler(t, f, y0):
    n = len(t)
    y = np.zeros(n)
    h = (t[-1]-t[0])/(n-1)
    y[0]=y0
    for i in range(n-1):
        y[i+1] = y[i]+h*f(t[i], y[i])
    return y
# function f
f = lambda t, y: y
# degine grid
a = 0.0; b = 2.0;
N = 50 #try it with : 10, 20, 50, 100
t = np.linspace(a, b, N+1)
# initial condition
y0 = 1.0
# Solving dy/dt=f(t,y) by Euler method
y = euler(t,f,y0)
# Plot the results
plt.plot(t, y, '-o', ms=3, c='r', label="Euler method")
plt.plot(t, np.exp(t), '--', c='k', label="Exact solution")
plt.legend(labelcolor=['r', 'k'], fontsize=12)
plt.xlabel("t"); plt.ylabel("y"); plt.show()
```

3. **Runge-kutta method:** Use the same code above and replace the Euler method by 4th order Runge-Kutta as shown below:

```
def Runge_Kutta4(t0, y0, f, h, N):
    t_values = [t0]
    y_values = [y0]
    t = t0
    y = y0
    for _ in range(N):
        k1 = h * f(t, y)
        k2 = h * f(t + 0.5 * h, y + 0.5 * k1)
        k3 = h * f(t + 0.5 * h, y + 0.5 * k2)
        k4 = h * f(t + h, y + k3)

        y = y + (k1 + 2 * k2 + 2 * k3 + k4) / 6.0
        t = t + h

        t_values.append(t)
        y_values.append(y)
    return np.array(t_values), np.array(y_values)
```