Task3 - Método de Euler

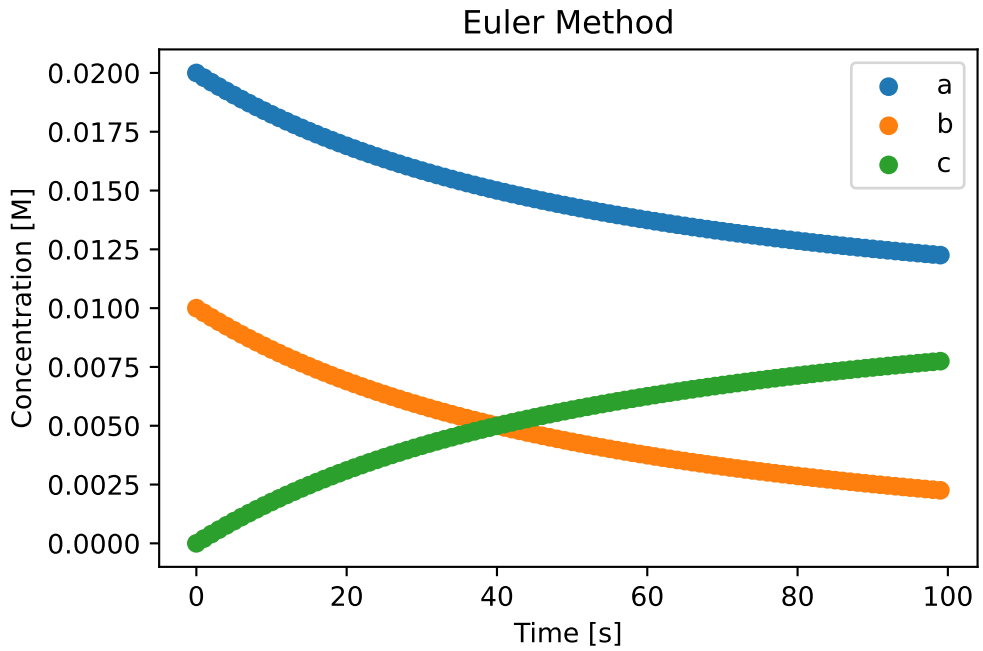
Redes Biológicas y Biología de Sistemas

Sandra Mingo Ramírez

Parte 1: Método de Euler para $a + b \xrightarrow{k} c$

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

def euler(initial_conditions, constant, timespan):

    U = np.zeros((len(initial_conditions), timespan))
    U[:, 0] = initial_conditions

    for i in range(1, timespan):
        U[0, i] = U[0, i-1] - constant * U[0, i-1] * U[1, i-1]
        U[1, i] = U[1, i-1] - constant * U[0, i-1] * U[1, i-1]
        U[2, i] = U[2, i-1] + constant * U[0, i-1] * U[1, i-1]

    return U

u_0 = [0.02, 0.01, 0.0]
k = 1
timespan = 100 #seconds

time_evolution = euler(u_0, k, timespan)


fig = plt.figure()
plt.scatter(range(timespan), time_evolution[0, :], label='a')
plt.scatter(range(timespan), time_evolution[1, :], label='b')
plt.scatter(range(timespan), time_evolution[2, :], label='c')
plt.xlabel("Time [s]")
```

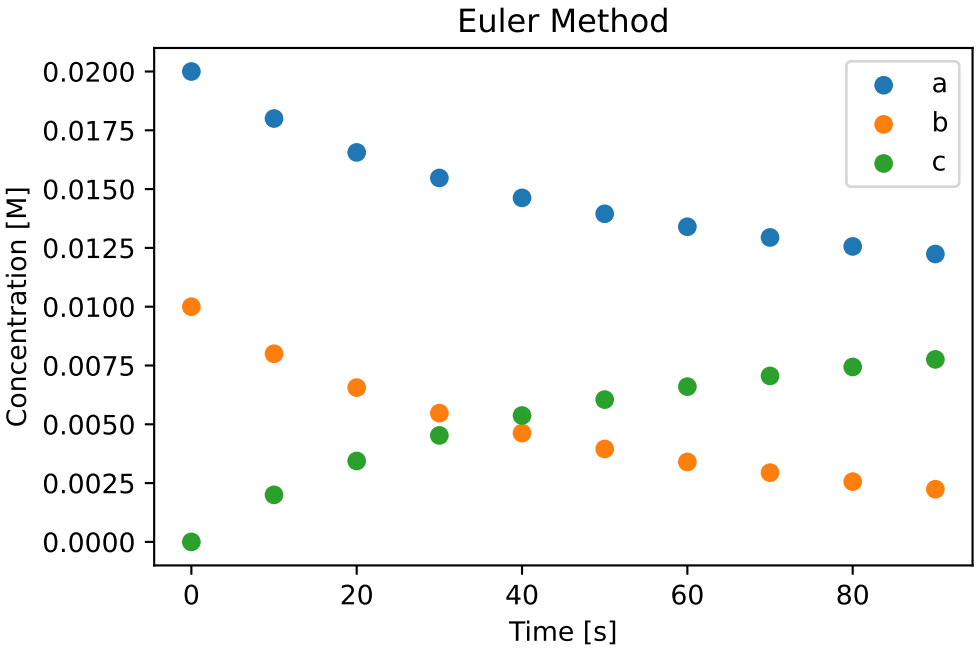
```
plt.ylabel("Concentration [M]")
plt.title("Euler Method")
plt.legend()
plt.show()
```



Incluir tiempo de integración. Medimos durante la misma cantidad de tiempo total, pero queremos medir las concentraciones con distinta frecuencia (cada segundo, cada dos segundos, etc.).

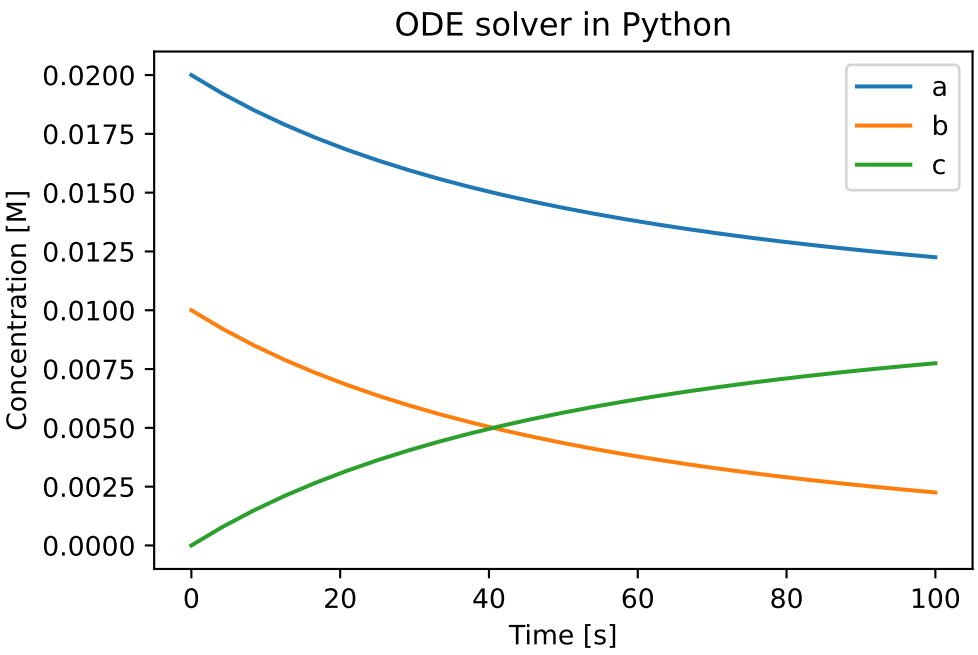
```
import numpy as np
import matplotlib.pyplot as plt

def euler(initial_conditions, constant, timespan):

    U = np.zeros((len(initial_conditions), timespan))
    U[:, 0] = initial_conditions

    for i in range(1, timespan):
        U[0, i] = U[0, i-1] - timespan * constant * U[0, i-1] * U[1, i-1]
        U[1, i] = U[1, i-1] - timespan * constant * U[0, i-1] * U[1, i-1]
        U[2, i] = U[2, i-1] + timespan * constant * U[0, i-1] * U[1, i-1]
```

```
    return U

u_0 = [0.02, 0.01, 0.0]
k = 1
timestep = 10
timespan = int(100/timestep)

time_evolution = euler(u_0, k, timespan)


fig = plt.figure()
plt.scatter(np.arange(timespan) * timestep, time_evolution[0, :], label='a')
plt.scatter(np.arange(timespan) * timestep, time_evolution[1, :], label='b')
plt.scatter(np.arange(timespan) * timestep, time_evolution[2, :], label='c')
plt.xlabel("Time [s]")
plt.ylabel("Concentration [M]")
plt.title("Euler Method")
plt.legend()
plt.show()
```



Ahora, en lugar de utilizar la función de Euler, utilizaremos ODEINT:

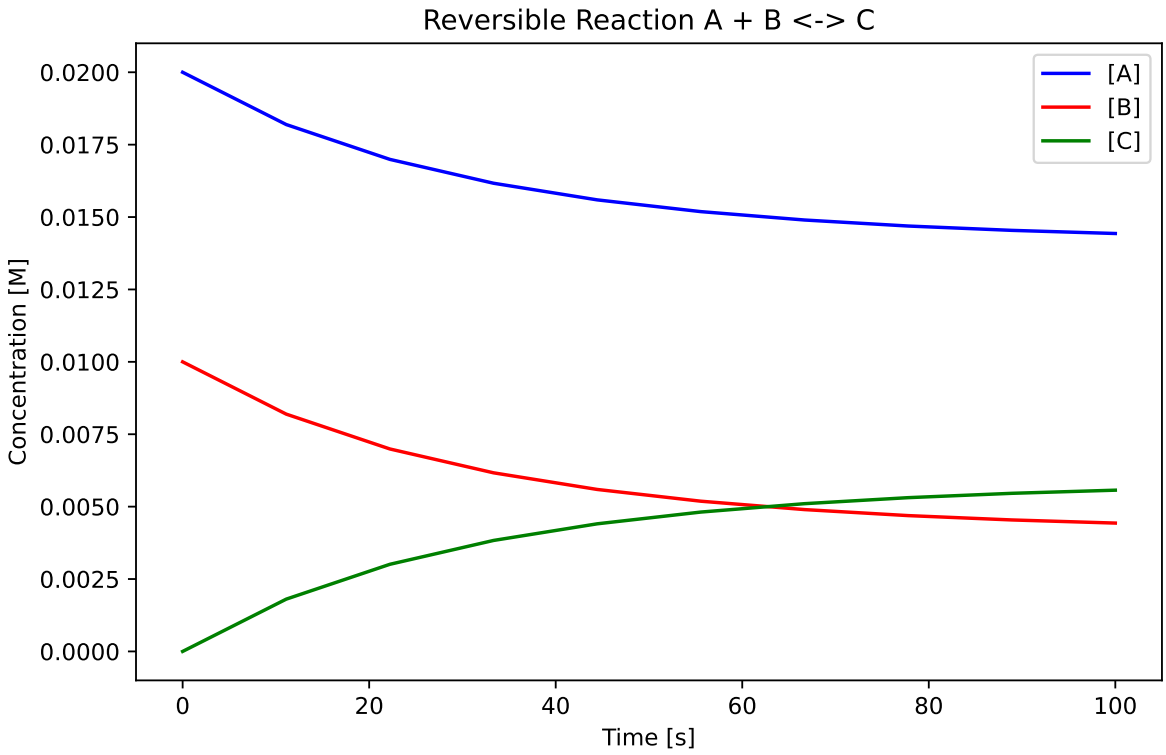
```
from scipy.integrate import odeint
import numpy as np

# Definir el sistema de ecuaciones diferenciales
def differential_equations(y, t, k):
    A, B, C = y  # Concentraciones de A, B y C
    dA_dt = -k * A * B  # Ecuación para A
    dB_dt = -k * A * B  # Ecuación para B
    dC_dt = k * A * B   # Ecuación para C
    return [dA_dt, dB_dt, dC_dt]

# Parámetros
k = 1  # Constante de reacción
A0 = 0.02  # Concentración inicial de A
B0 = 0.01 # Concentración inicial de B
C0 = 0.0  # Concentración inicial de C

# Condiciones iniciales (A0, B0, C0)
y0 = [A0, B0, C0]

# Tiempo de integración
t = np.linspace(0, 100, 25)

# Resolver las ecuaciones diferenciales
ode_solved = odeint(differential_equations, y0, t, args=(k,))

# Graficar los resultados
plt.plot(t, ode_solved[:, 0], label='a')
plt.plot(t, ode_solved[:, 1], label='b')
plt.plot(t, ode_solved[:, 2], label='c')

plt.xlabel('Time [s]')
plt.ylabel('Concentration [M]')
plt.title('ODE solver in Python')
plt.legend()
plt.show()
```



Parte 2: Método de Euler para ecuación general $a + b \underset{k_2}{\overset{k_1}{\rightleftharpoons}} c$

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

# Definir el sistema de ecuaciones diferenciales
def differential_equations(y, t, k1, k2):
    A, B, C = y  # Concentraciones de A, B y C
    dA_dt = -k1 * A * B + k2 * C  # Ecuación para A
    dB_dt = -k1 * A * B + k2 * C  # Ecuación para B
    dC_dt = k1 * A * B - k2 * C   # Ecuación para C
    return [dA_dt, dB_dt, dC_dt]

# Parámetros
k1 = 1   # Constante de reacción directa
k2 = 0.01 # Constante de reacción inversa

# Condiciones iniciales
```

```
A0 = 0.02
B0 = 0.01
C0 = 0.0
y0 = [A0, B0, C0]

# Tiempo de integración
t = np.linspace(0, 100, 10)

# Resolver las ecuaciones diferenciales
ode_solved = odeint(differential_equations, y0, t, args=(k1, k2))

# Graficar los resultados
plt.figure(figsize=(8, 5))
plt.plot(t, ode_solved[:, 0], label='[A]', color='blue')
plt.plot(t, ode_solved[:, 1], label='[B]', color='red')
plt.plot(t, ode_solved[:, 2], label='[C]', color='green')

plt.xlabel('Time [s]')
plt.ylabel('Concentration [M]')
plt.title('Reversible Reaction A + B <-> C')
plt.legend()
plt.show()
```

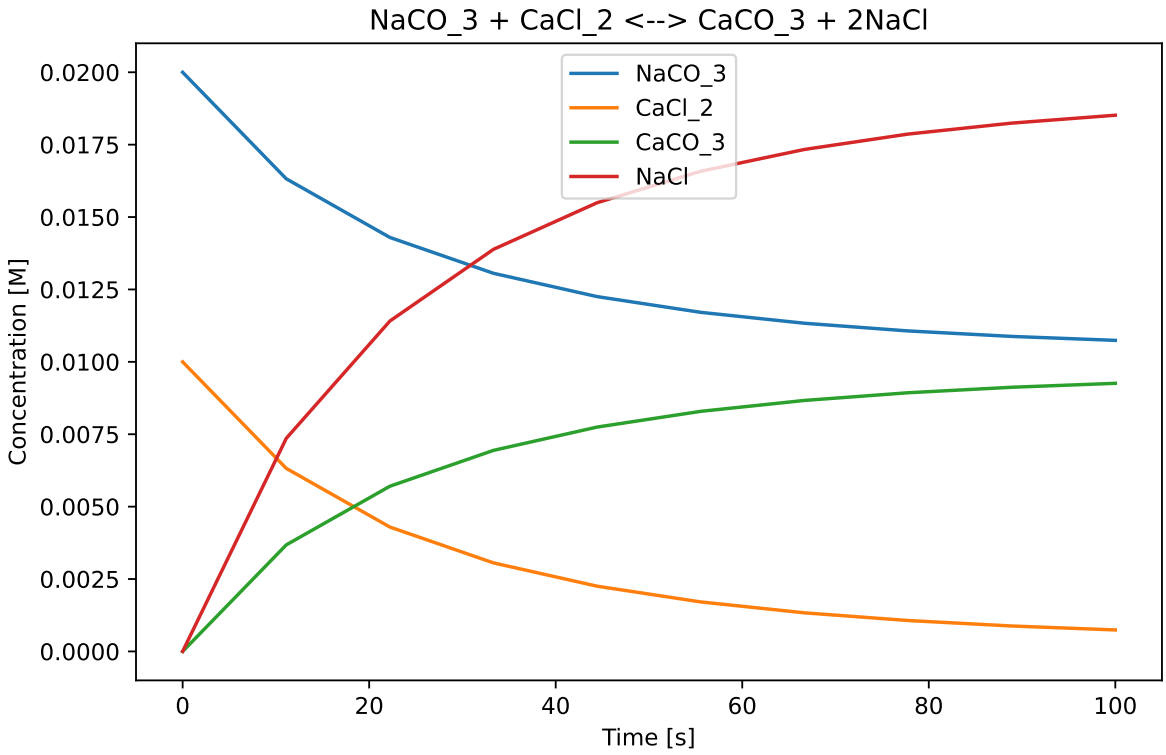


Parte 3: Método de Euler para ecuación de la sal

$$NaCO_3 + CaCl_2 \underset{k_2}{\overset{k_1}{\rightleftharpoons}} CaCO_3 + 2NaCl$$

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

# Definir el sistema de ecuaciones diferenciales
def differential_equations(y, t, k1, k2):
    A, B, C, D = y
    dA_dt = -k1 * A * B + k2 * C * D**2  # Ecuación para A (NaCO_3)
    dB_dt = -k1 * A * B + k2 * C * D**2  # Ecuación para B (CaCl_2)
    dC_dt = k1 * A * B - k2 * C * D**2   # Ecuación para C (CaCO_3)
    dD_dt = 2 * (k1 * A * B - k2 * C * D**2)# Ecuación para D (NaCl)
    return [dA_dt, dB_dt, dC_dt, dD_dt]
```

```
k_1 = 2.3
k_2 = 2.5

a_0 = 0.02
b_0 = 0.01
c_0 = 0.0
d_0 = 0.0
y0 = [a_0, b_0, c_0, d_0]

# Tiempo de integración
t = np.linspace(0, 100, 10)

# Resolver las ecuaciones diferenciales
ode_solved = odeint(differential_equations, y0, t, args=(k_1, k_2))

# Graficar los resultados
plt.figure(figsize=(8, 5))
plt.plot(t, ode_solved[:, 0], label='NaCO_3')
plt.plot(t, ode_solved[:, 1], label='CaCl_2')
plt.plot(t, ode_solved[:, 2], label='CaCO_3')
plt.plot(t, ode_solved[:, 3], label='NaCl')

plt.xlabel('Time [s]')
plt.ylabel('Concentration [M]')
plt.title('NaCO_3 + CaCl_2 <--> CaCO_3 + 2NaCl')
plt.legend()
plt.show()
```



Primero calcular valor de equilibrio de NaCl y después ajustar el tiempo “a ojo” hasta que la curva respectiva llegue a ese valor (pintar línea horizontal en ese valor para comprobar)