Métodos Numéricos

Trabajo práctico N°4

Ecuaciones diferenciales ordinarias

Grupo 11: CHEN, Carlos Angel, Legajo 60689
MOLDOVAN LOAYZA, Alexander Stephan, Legajo 60498
MOLINA, Facundo Nicolás, Legajo 60526

Profesor: Pablo Ignacio Fierens

Fecha de entrega: 26 de mayo de 2022

Buenos Aires, Argentina

Secciones

Índice de Figuras

Índice de Códigos

1. Función ruku4: Runge-Kutta 4

En primer lugar, se programó una función capaz de resolver el sistema de ecuaciones diferenciales dado por:

$$\begin{aligned} \dot{\vec{x}} &= \vec{f}(t, \vec{x}) \qquad t > t_0, \\ \vec{x}(t_0) &= \vec{x}_0 \end{aligned}$$

El método empleado fue el de Runge-Kutta 4, cuyo algoritmo sigue al siguiente fórmula:

$$\begin{aligned} x_{k+1} &= x_k + \frac{f_1 + 2f_2 + 2f_3 + f_4}{6}\Delta t \\ f1 &= f(t_k, x_k) \\ f_2 &= f\left(t_k + \frac{\Delta t}{2}, x_k + \frac{\Delta t}{2} f_1\right) \\ f_3 &= f\left(t_k + \frac{\Delta t}{2}, x_k + \frac{\Delta t}{2} f_2\right) \\ f_4 &= f(t_k + \Delta t, x_k + \Delta t f_3) \end{aligned}$$

La implementación de esta función se detalla a continuación:

Código 1: Definición de ruku4

```python
#Función:   ruku4
#           Resuelve de forma aproximada el sistema de ecuaciones diferenciales:
#               dx/dt = f(t,x)  t>t0
#               x(t0)=x0
#           Emplea el método de Runge-Kutta 4
#Recibe:    f:     handle a la funcion f = dx/dt
#           t0,tf: tiempo inicial y final
#           h:     paso de integración
#           x0:    condición inicial
#Devuelve:  t:     arreglo con los instantes de tiempo
#           x:     aproximaciones numéricas a x (una fila por instante de tiempo)
def ruku4(f,t0,tf,h,x0):
    n = x0.shape[0] # número de componentes de x
    if h <= 0:      # si el paso de integración no es positivo, informa el error
        print("h debe ser mayor a 0")
        return np.zeros(1),np.zeros((n,1))
    N = int(ceil((tf-t0)/h)) # cantidad de pasos. Se toma la función techo para
    # que la h empleada sea a lo sumo tan grande como la pedida
    t = linspace(t0,tf,N+1) # arreglo de tiempos
    x = np.zeros((n,N+1))   # matriz con el resultado
    x[:,0] = x0 # los valores iniciales son dato

    for k in range(N):  # por cada paso del algoritmo...
        tk=t[k]         # se extraen los valores de x(k) y t(k)
        xk=x[:,k]

```

```
        # Y se aplica el algoritmo de Runge-Kutta 4
        f1 = f(tk, xk)
        f2 = f(tk+h/2, xk+f1*h/2)
        f3 = f(tk+h/2, xk+f2*h/2)
        f4 = f(tk+h, xk+f3*h)
        x[:,k+1]=xk+h*(f1+2*f2+2*f3+f4)/6

    x = x.T
    return t,x
```

2. Función higginsselkov

La función presentada en la sección anterior fue empleada para resolver el siguiente sistema de ecuaciones:

$$\begin{aligned} \dot{s} &= v_0 - 0.23 \cdot s \cdot p^2 && t > 0 \\ \dot{p} &= 0.23 \cdot s \cdot p^2 - 0,40 \cdot p && t > 0 \\ s(0) = 2 \quad p(0) &= 3 \quad v_0 \in [0.48,\ 0.60] \end{aligned}$$

Por recomendación de la cátedra, se simuló el sistema para un intervalo de 600 segundos. Se empleó un paso de $h = 0.01$, con el cual se obtiene un error global de aproximadamente $1 \cdot 10^{-6}$.

Se definieron las siguientes funciones:

- **higginsselkov():** resuelve sistema de EDOs y realiza gráficos de la solución numérica usando la función de la Sección 1 para las siguientes situaciones de v0:
  - $v_0 = v_c$
  - $v_0 = v_c - 0.01$
  - $v_0 = v_c + 0.01$
- **dx(t,x):** devuelve resultado numérico de sistema de EDOs correspondiente a los valores de entrada.
- **plot(t, s, p, title):** funcion auxiliar para graficar.
- **isOsc():** permite analizar si la solución numérica esta dentro de un régimen de oscilación permanente. Para ello, optamos por comparar que los máximos y mínimos en los dos últimos intervalos de 100 segundos de s(t) y p(t), respectivamente, no difieran en mas de 0.01.
- **calcVc():** función para hallar el valor de v_c. Realiza iteraciones analizando la solución numérica del sistema para distintos valores de v_0. Emplea isOsc() para decidir si incrementar (en caso que las funciones aún estén en regimen oscilatorio permanente) o decrementar (en caso que no lo estén) el valor de v_0.

Se obtuvo una aproximación de $v_c = 0.52357$.

En los siguientes gráficos, se puede observar que para valores inferiores a v_c el sistema se encuentra en régimen oscilatorio permanente mientras que para valores superiores deja de estarlo.

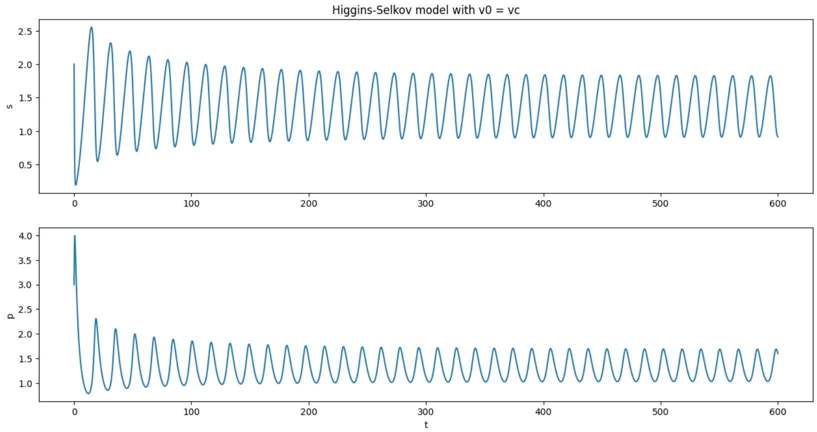

Figura 1: Sistema con $v_0 = v_c$

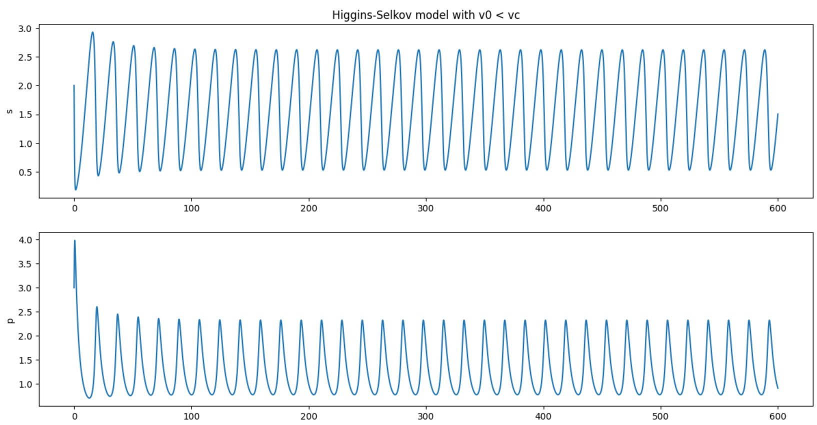

Figura 2: Sistema con $v_0 = v_c - 0.01$

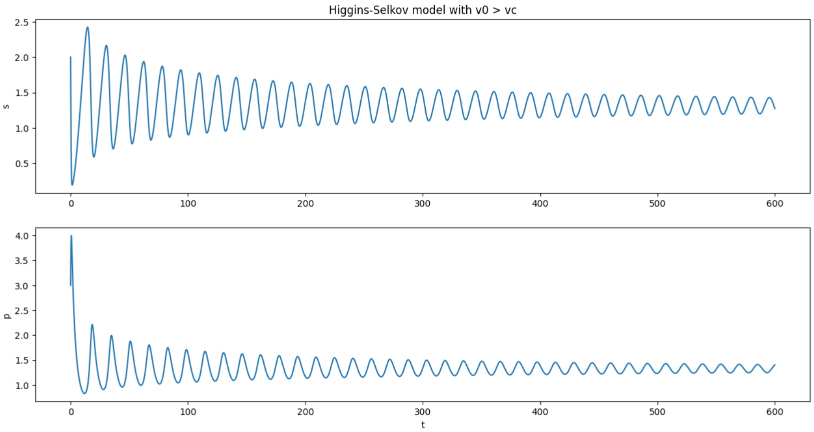

Figura 3: Sistema con $v_0 = v_c + 0.01$

Código 2: Definición de higginsselkov

```

v0 = 0.48
#Función:   higginsselkov:
#           emplea la función ruku4 para resolver el modelo de glucólisis de
#           Higgins-Selkov. Grafica el resultado para 3 valores de v0:
#           v0=vc, v0<vc y v0>vc, donde vc es el valor a partir del cual
#           s y p dejan de oscilar
#Recibe:    nada
#Devuelve:  nada
def higginsselkov():
    t0 = 0
    tf = 600
    h = 0.01
    x0 = np.array([2,3])      #s0,p0
    global v0
    v0 = vc = calcVc()

    t, result = ruku4(dx, t0, tf, h, x0)
    result = result.T
    plot(t, result[0], result[1], "Higgins-Selkov model with v0 = vc")

    v0 = vc - 0.01
    t, result = ruku4(dx, t0, tf, h, x0)
    result = result.T
    plot(t, result[0], result[1], "Higgins-Selkov model with v0 < vc")

    v0 = vc + 0.01
    t, result = ruku4(dx, t0, tf, h, x0)
    result = result.T
    plot(t, result[0], result[1], "Higgins-Selkov model with v0 > vc")

```

```python
def dx(t,x):
    return np.array([v0 - 0.23 * x[0] * (x[1]**2), 0.23 * x[0] * (x[1]**2) - 0.40 * x[1]])

def plot(t, s, p, title):
    plt.subplot(2, 1, 1)
    plt.plot(t, s)
    plt.ylabel('s')
    plt.title(title)

    plt.subplot(2, 1, 2)
    plt.plot(t, p)
    plt.xlabel('t')
    plt.ylabel('p')

    plt.show()

def isOsc(t0, tf, h, arr):
    dt = (tf-t0)/h
    eps = 0.01
    for i in range(2):
        min1 = min(arr[i][int(dt*2/3):int(dt*5/6)])
        min2 = min(arr[i][int(dt*5/6):int(dt)])
        max1 = max(arr[i][int(dt*2/3):int(dt*5/6)])
        max2 = max(arr[i][int(dt*5/6):int(dt)])
        if abs(min2 - min1) > eps or abs(max2 - max1) > eps:
            return False
    return True

def calcVc():
    t0 = 0
    tf = 600
    h = 0.01
    x0 = np.array([2, 3])
    global v0
    v0 = vc = 0.48

    delta = 0.01
    for i in range(10):
        t, res = ruku4(dx, t0, tf, h, x0)
        res = res.T
        if isOsc(t0, tf, h, res):
            v0 += delta
            if v0 == vc:
                delta /= 2
                v0 -= delta
        else:
            vc = v0
            delta /= 2
            v0 -= delta
    return vc
```

3. Función test()

Para la la función `test()` se utilizo la librería `Scipy`, en el cual tiene un método llamado `solve_ivp`, para resolver ecuaciones diferenciales.

Código 3: Definición de `test`

```
def test():
#################################
# Comparacion entre solve_ivp() y la funcion ruku4() #
#################################
    R = 1e3             #Valor de la resistencia
    C = 1e-6          #Valor de la capacidad
    w = 2.0*pi*1000     #frecuencia angular de la señal de entrada
    A = 1.0             #amplitud de la señal de entrada
    T = 5*2*pi/w     #simulo cinco ciclos
    def xsol(t):
        x = -exp(-t/(R*C))+cos(w*t)+w*R*C*sin(w*t)
        x = (A/(1+(w*R*C)**2))*x
        return x
    def dx(t,x):
        return ((A*cos(w*t)-x)/(R*C))

    x0 = np.zeros(1)

    t4,x4 = ruku4(dx,0,T,0.0001,x0)
    t = linspace(0,T,int(10e5))
    x = xsol(t)

    sol = solve_ivp(dx, (0, T), x0, method='RK23', max_step=0.01)

    plt.plot(t,x, label='Explicit')
    plt.plot(t4,x4, label='RK4')
    plt.plot(sol.t, sol.y[0], label='RK23')
    plt.legend()
    plt.grid(True)
    plt.show()

    ##############################################
    # Comparacion entre solve_ivp() y la funcion ruku4(), usando higginsselkov() #
    ##############################################
    t0 = 0
    tf = 600
    h = 0.01
    x0 = np.array([2,3])       #s0,p0
    global v0
    v0 = vc = calcVc()

    plt.figure()
    t, result = ruku4(dx, t0, tf, h, x0)
    result = result.T
```

```
    sol1 = solve_ivp(dx, (0, tf), x0, method='RK45', max_step=0.01)

    plt.subplot(2, 1, 1)
    plt.plot(sol1.t, sol1.y[0],'.', label='solve_ivp')
    plt.plot(t, result[0], label='ruku4')
    plt.ylabel('s')
    plt.title("Higgins-Selkov model with v0 = vc")
    plt.legend()

    plt.subplot(2, 1, 2)
    plt.plot(sol1.t, sol1.y[1],'.', label='solve_ivp')
    plt.plot(t, result[1], label='ruku4')
    plt.xlabel('t')
    plt.ylabel('p')
    plt.legend()
    plt.show()

    plt.figure()
    v0 = 0.48
    t, result = ruku4(dx, t0, tf, h, x0)
    result = result.T
    sol2 = solve_ivp(dx, (0, tf), x0, method='RK45', max_step=0.01)

    plt.subplot(2, 1, 1)
    plt.plot(sol2.t, sol2.y[0],'.', label='solve_ivp')
    plt.plot(t, result[0], label='ruku4')
    plt.ylabel('s')
    plt.title("Higgins-Selkov model with v0 < vc")
    plt.legend()

    plt.subplot(2, 1, 2)
    plt.plot(sol2.t, sol2.y[1],'.', label='solve_ivp')
    plt.plot(t, result[1], label='ruku4')
    plt.xlabel('t')
    plt.ylabel('p')
    plt.legend()
    plt.show()

    plt.figure()
    v0 = 0.6
    t, result = ruku4(dx, t0, tf, h, x0)
    result = result.T
    sol3 = solve_ivp(dx, (0, tf), x0, method='RK45', max_step=0.01)

    plt.subplot(2, 1, 1)
    plt.plot(sol3.t, sol3.y[0],'.', label='solve_ivp')
    plt.plot(t, result[0], label='ruku4')
    plt.ylabel('s')
    plt.title("Higgins-Selkov model with v0 > vc")
    plt.legend()

```

```
    plt.subplot(2, 1, 2)
    plt.plot(sol3.t, sol3.y[1],'.', label='solve_ivp')
    plt.plot(t, result[1], label='ruku4')
    plt.xlabel('t')
    plt.ylabel('p')
    plt.legend()
    plt.show()
```

Se obtuvieron los siguientes resultados:

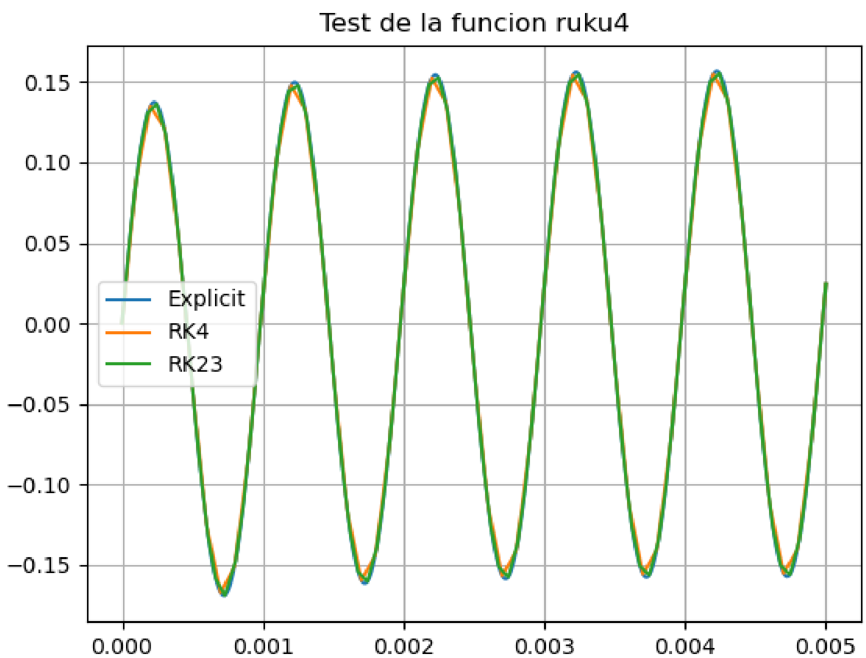

Figura 4: Comparación entre la `ruku4()` y `solve_ivp`

Es posible observar que las curvas se asemejan utilizando el método de RK23 de la función `solve_ivp`.

Luego se comparo `ruku4()` y `solve_ivp` para los distintos casos de `higginsselkov()` y es posible observar que el algoritmo escrito por nosotros resuelve correctamente la ecuación diferencial del problema.

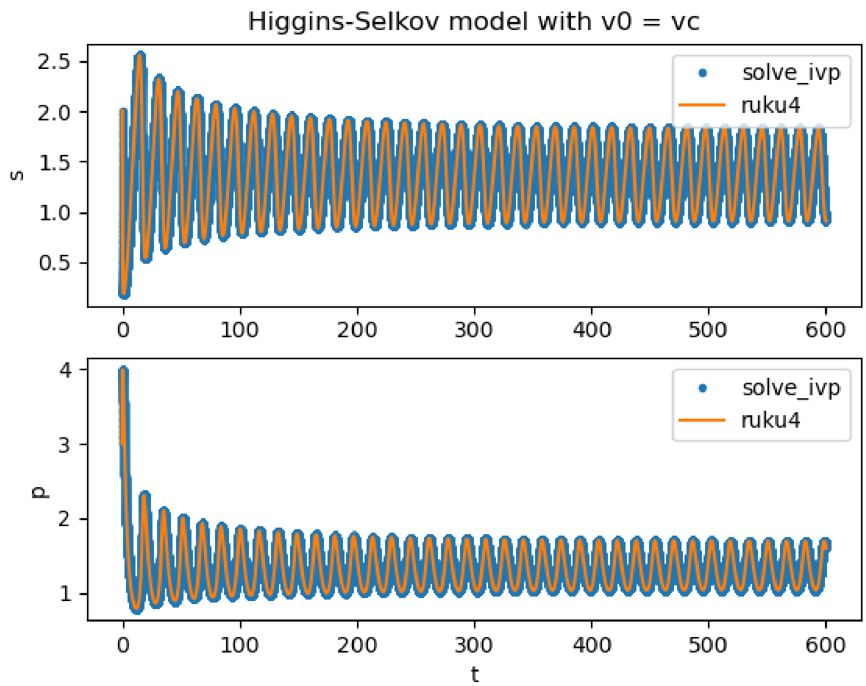

Figura 5: Comparación entre la `ruku4()` y `solve_ivp`, usando `higginsselkov()`

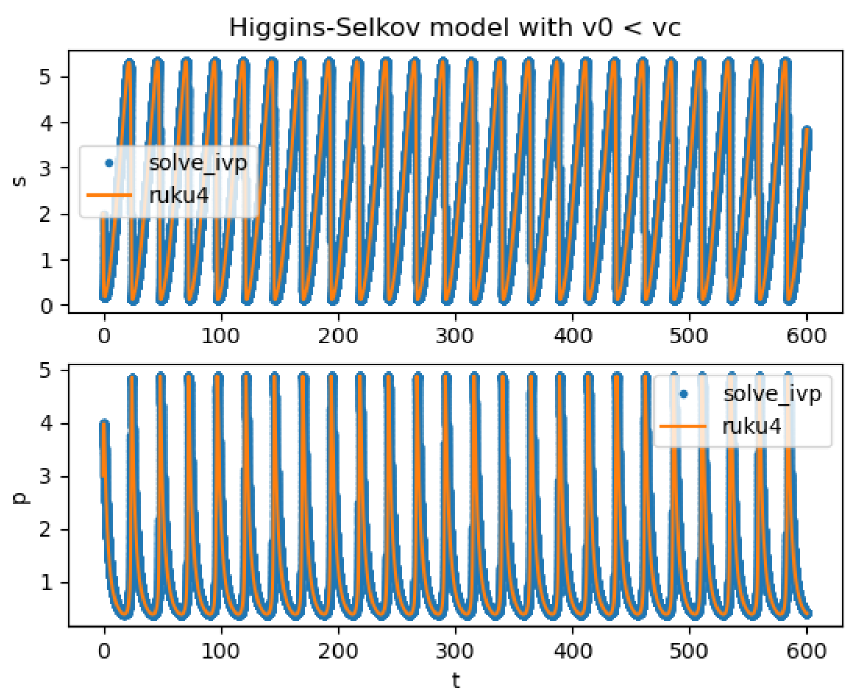

Figura 6: Comparación entre la `ruku4()` y `solve_ivp`, usando `higginsselkov()`

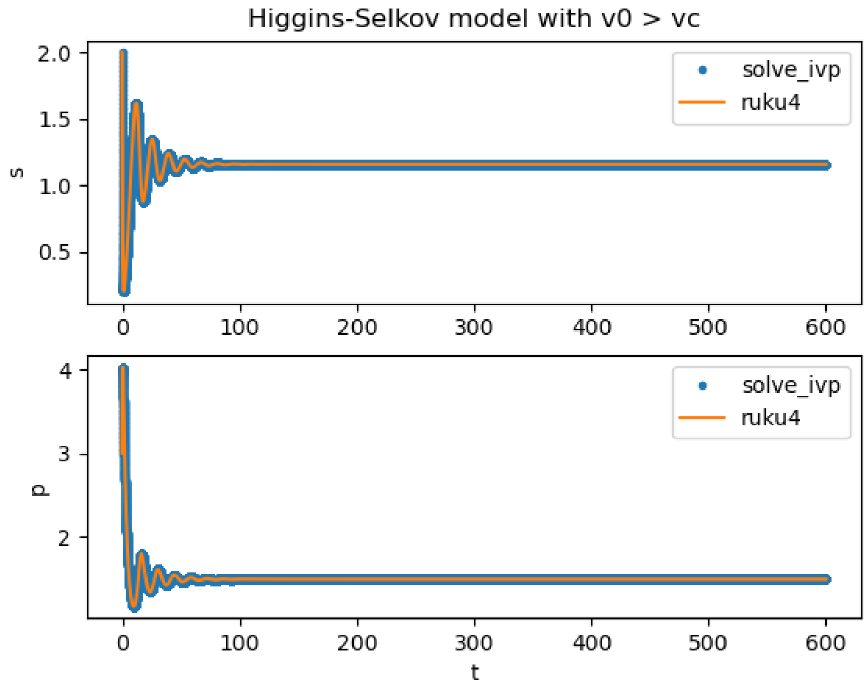

Figura 7: Comparación entre la `ruku4()` y `solve_ivp`, usando `higginsselkov()`