Trabajo Práctico N$^{\underline{o}}$4

Ecuaciones diferenciales ordinarias

93.54 Métodos Numéricos

Grupo N°4

Legajo N°	Nombre
61428	Kevin Amiel Wahle
61430	Francisco Basili
61431	Nicolás Bustelo

26/05/2022

Índice

1. Introducción

El objetivo de este informe es presentar y explicar el funcionamiento de un programa realizado en *Python*, el cual consiste en resolver una ecuación diferencial ordinaria mediante el método de Runge-Kutta 4. El código utilizado se encuentra en el Anexo, al final del informe, el cual incluye el testbench utilizado.

2. *ruku4(f, t_0, t_f, h, x_0)*

Para la resolución de la ecuaciones diferenciales ordinarias se definió una función *ruku4*, la cual recibe un handle a una función f, sus respectivas condiciones iniciales y el tiempo inicial y final. Para ello se aplicó el algoritmo de Rounge-Kutta 4, cuyas fórmulas fueron vistas en clase.

3. *Resolución del sistema de ecuaciones*

3.1. *ModeloHH(t,x)*

Se definieron las diferentes funciones brindadas en la consigna, para luego poder iterar entre los diferentes valores. Se definieron las constantes g_{Na}, g_K, g_L v_{Na}, v_K, v_L y las funciones auxiliares $\alpha_n(v)$,$\alpha_m(v)$, $\alpha_h(v)$, $\beta_n(v)$, $\beta_m(v)$, $\beta_h(v)$. Luego esta función devuelve un arreglo con los valores de $\dot{v}(t)$, $\dot{n}(t)$, $\dot{m}(t)$ y $\dot{h}(t)$ calculados, correspondientes a una iteración en específico. Es por este motivo que esta función es llamada en reiteradas oportunidades.

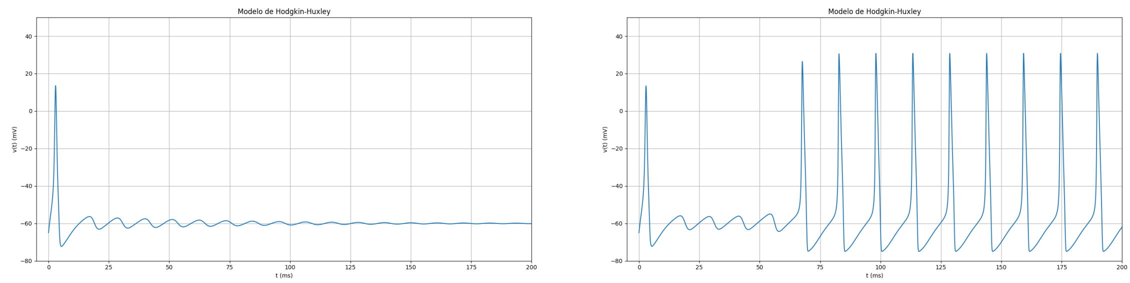
3.2. *hodgkinhuxley()*

En esta función se definieron tanto las condiciones iniciales como los valores de t para las cuales se hará el análisis. También se llama a la función *ruku4*, pasándole como parámetro la función *ModeloHH*, la cual sera utilizada para calcular las sucesiones del sistema de ecuaciones planteado.

Luego se graficó la $v(t)$ en función del tiempo para analizar las posibles oscilaciones que dependían del valor de i_0, teniendo en cuenta que $i_C(t) = i_0$, y de esta forma poder determinar el I_c óptimo que permitía que la función no oscilara.

3.3. Obtención del valor de i_c

Para la obtención de este valor, se analizaron los diferentes gráficos de $v(t)$ obtenidos para los diferentes i_0. Para la elección de los valores de i_c se realizó una búsqueda binaria, similar a la idea del método de bisección visto en clase. En particular se tomaron los valores de 10 y 8 y como en 8 no oscilaba y en 10 si, se realizó la búsqueda binaria y los valores intermedios fueron 9, 8.5, 8.75, 8.875, 8.9375. A continuación, se pueden observar los gráficos más relevantes obtenidos correspondientes a las últimas dos iteraciones.



(a) $i_O = 8,875$ (b) $i_O = 8,9375$

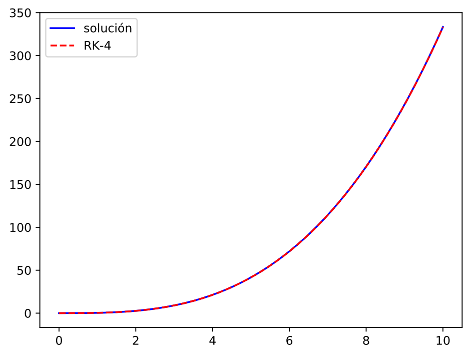
Podemos concluir que $8.875 < i_c < 8,9375$. Por lo tanto, si usamos un valor de $i_0 \leq 8.875$ la respuesta no oscila, en cambio si usamos un valor de $i_0 \geq 8.9375$ la respuesta oscila. Para obtener una cota mas precisa, se podría haber seguido iterando en la búsqueda binaria, hasta obtener la cantidad de decimales que requiera la aplicación.

4. *test()*

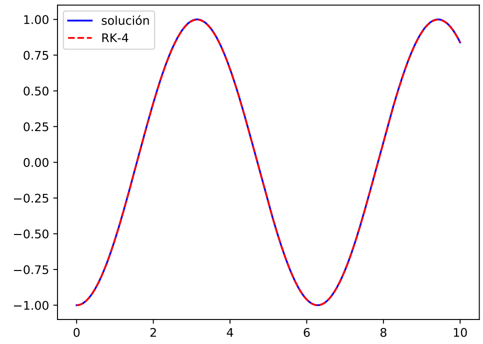
Para probar la función *ruku4()* se opto por elegir 4 funciones "famosas" de distinto tipo y averiguar sus integrales por medio de dicha función. Las funciones elegidas fueron:

$$f_1(t) = t^2 \qquad f_2(t) = \sin(t) \qquad f_3(t) = \frac{1}{t+1} \qquad f_4(t) = e^t$$

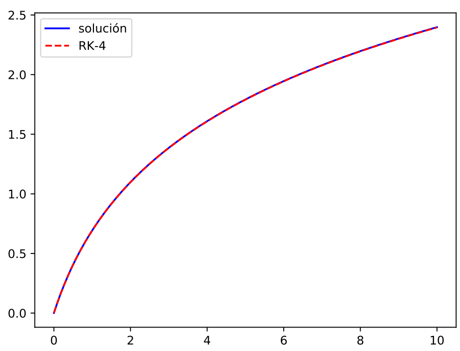
Dentro de la función test() se definió arbitrariamente el intervalo de prueba $t = [0, 10]$ con $h = 0.01$ y los valores iniciales para cada función. Luego, se calcularon sus integrales con *ruku4()* y se graficaron junto con sus verdaderos resultados que se pueden obtener integrando, con el objetivo de poder analizar sus diferencias.



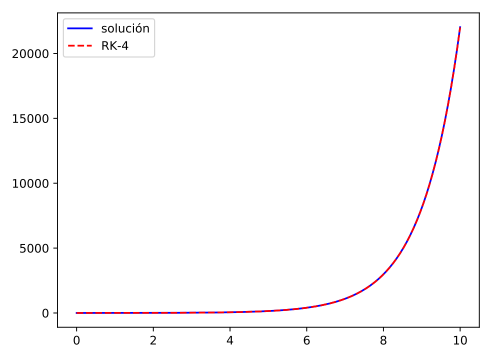
(a) Integral de f_1



(b) Integral de f_2



(c) Integral de f_3



(d) Integral de f_4

Se puede observar como en todos los casos los puntos obtenidos con las iteraciones de Runge-Kutta 4 se ajustan de manera muy precisa a las diferentes funciones, demostrando el correcto funcionamiento del algoritmo planteado. A su vez, la diversidad de funciones utilizadas, le brinda una mayor robustez al testbench.

5. Anexo

5.1. Código completo en Python

```python
# ----------------------------------------------------------------------------
#  @file      +piensa.py+
#  @brief     +Ecuaciones diferenciales ordinarias+
#  @author    +Grupo 4+
# ----------------------------------------------------------------------------

# ----------------------------------------------------------------------------
# LIBRARIES
# ----------------------------------------------------------------------------
from turtle import color
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math as mt

# ----------------------------------------------------------------------------
# FUNCTION DEF
# ----------------------------------------------------------------------------
# Resolución de ecuaciones diferenciales con RK4
def ruku4(f, t0, tf, h, x0):
    n = int ((tf-t0)/h) # Número de elementos del vector
    t=np.linspace(t0, tf, n+1) # Vector de tiempos
    x=np.zeros((n+1, len(x0))) # Matriz x: tiene una fila por cada instante de tiempo y una columna dada por el tamaño de x0
    x[0]=x0
    # Resolución de la ecuación diferencial con Runge-Kutta de orden 4
    for i in range(n):
        f1=f(t[i],x[i])
        f2=f(t[i]+h/2,x[i]+(h/2)*f1)
        f3=f(t[i]+h/2,x[i]+(h/2)*f2)
        f4=f(t[i]+h,x[i]+h*f3)
        x[i+1]=x[i]+h*(f1+2*f2+2*f3+f4)/6
    return t, x

def ModeloHH(t,x):
    # Parámetros de la ecuación diferencial
    g_Na=120; g_K=36 ; g_L=0.3
    E_Na=115; E_K=-12; E_L=10.6
    C_m=1
    V_Na=50;  V_K=-77; V_L=-54.4
    ic = 8.875

    v=x[0]; n=x[1]; m=x[2]; h=x[3]

    alfa_n = lambda v: 0.01*((v+55)/(1-np.exp(-(v+55)/10)))
    alfa_m = lambda v: 0.1*((v+40)/(1-np.exp(-(v+40)/10)))
    alfa_h = lambda v: 0.07*np.exp(-(v+65)/20)
    beta_n = lambda v: 0.125*np.exp(-(v+65)/80)
    beta_m = lambda v: 4*np.exp(-(v+65)/18)
    beta_h = lambda v: 1/(1+np.exp(-(v+35)/10))

    # Ecuaciónes diferenciales
    fv = ic-g_Na*(m**3)*h*(v-V_Na)-g_K*(n**4)*(v-V_K)-g_L*(v-V_L)
    fn = alfa_n(v)*(1-n)-beta_n(v)*n
    fm = alfa_m(v)*(1-m)-beta_m(v)*m
    fh = alfa_h(v)*(1-h)-beta_h(v)*h

    return np.array([fv,fn,fm,fh])

def hodgkinhuxley():
    #Definimos las condiciones iniciales
    x0=np.array([-65,0,0,0])
    t0=0
    tf=200
    h=0.01
    t,x=ruku4(ModeloHH,t0,tf,h,x0)

    #Graficamos las soluciones
```

```
    fig, ax = plt.subplots(figsize=(8, 4))
    ax.set_xlim(t0-5.0, tf)
    ax.set_ylim(-80, 50)
    plt.plot(t,x[:,0])
    plt.title("Modelo de Hodgkin-Huxley")
    plt.xlabel("t (ms)")
    plt.ylabel("v(t) (mV)")
    plt.grid()
    plt.show()

    return t,x

def test():
    t0=0; tf=10; h=0.01

    #Funciones de prueba
    df1 = lambda t,x: t**2
    f1 = lambda t,x: t**3/3
    x1=np.array([f1(t0,0)])

    df2 = lambda t,x: np.sin(t)
    f2 = lambda t,x: -np.cos(t)
    x2=np.array([f2(t0,0)])

    df3 = lambda t,x: 1/(t+1)
    f3 = lambda t,x: np.log(t+1)
    x3=np.array([f3(t0,0)])

    df4 = lambda t,x: np.e**t
    x4=np.array([df4(t0,0)])

    testfun = np.array([[df1,f1,x1],[df2,f2,x2],[df3,f3,x3],[df4,df4,x4]])

    for func, sol, x0 in testfun:   # Bucle de prueba
        t_rk, x_rk = ruku4(f=func, t0=t0, tf=tf, x0=x0, h=h)
        t = np.linspace(t0, tf, int((tf-t0)/h)+1)
        x = sol(t_rk,0)
        plt.plot(t,x, label="solución",color="blue")
        plt.plot(t_rk, x_rk, linestyle='dashed', label="RK-4", color="red")
        plt.legend()
        plt.show()
```