Práctica 2: The Fitzhugh-Nagumo model

Sofía Gutierrez, Mónica Higueras, Daniel Mancheño y Teresa Vargas

15 de noviembre de 2023

1. To send us

El presente trabajo está compuesto por un código en python que puedes encontrar en Aula Virtual. Existen algunos huecos en el código python que debes rellenar para que esté listo para ser ejecutado. Sigue las instrucciones y responde a las preguntas. Para aprobar este trabajo tienes que subir los siguientes documentos:

- Memoria con las respuestas y comentarios de las preguntas
- Código python rellenado

2. Introducción.

Se nos proporciona una introducción al modelo Fitzhugh-Nagumo de un sistema excitable de dos dimensiones como simplificación del modelo de Hodgkin-Huxley.

3. Instrucciones.

Tienes un cuaderno de Júpiter con un código incompleto. Su principal objetivo es programar correctamente el código para ejecutarlo y comprender así cada paso. Este trabajo se divide en varios pasos. También hay preguntas a lo largo del trabajo que deberás contestar en el documento con la respuesta correspondiente.

4. Diagrama de fase.

Implemente el flujo del modelo de Fitzhugh-Nagumo y simule algunas trayectorias. Intente utilizar una pequeña perturbación del potencial de reposo como condiciones iniciales.

¿Qué parámetros son constantes a lo largo de los diferentes escenarios? ¿Cuáles están variando? ¿Cuántos escenarios se definen? ¿Cuál es el intervalo de tiempo y cuántos puntos se representan?

Observamos tres escenarios distintos, que dependeen de los parámetros 'a', 'b', 'τ' e 'I'. Mientras que los parámetros a y b son constantes y característicos del modelo, τ es un parámetro de escalado del tiempo. El único parámetro que varía para los diferentes escenarios sería I, la corriente de estimulación.

Ante un estímulo de corriente, la membrana en reposo de una célula se despolarizará. Si se alcanza el umbral, la célula generará un potencial de acción. De esta forma, modificando el valor de corriente, podremos ver la evolución de nuestro sistema.

Estamos estudiando el sistema en un periodo de tiempo de t=0 a t=200, dibujando un total de 1500 puntos. Esto lo podemos ver en :

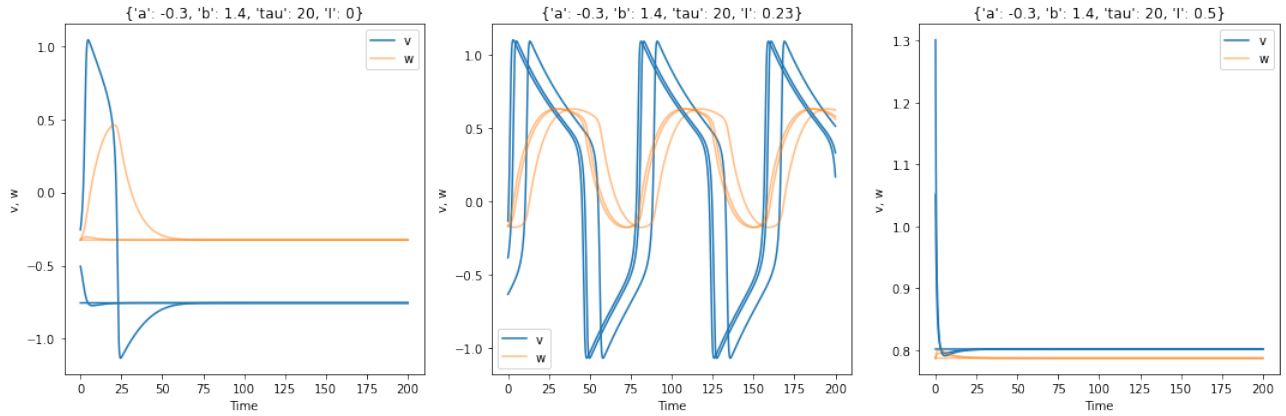
$$time\_span = np.linspace(0, 200, num = 1500) \tag{1}$$

4.1. El modelo Fitzhugh-Nagumo

¿Es un modelo lineal o no lineal?¿Cuál es el órden del modelo? Justifica tu respuesta.

```
v=x[0]
w= x[1]

v_d=v - v**3 - w + I
w_d=(v - a - b * w)/tau

X = [v_d, w_d]
```

Vemos que el grado máximo del sistema es 3, ya que el modelo presenta términos cúbicos. De esta manera, concluimos que nos encontramos ante un sistema no lineal de orden 3.



¿Cuál es el punto de descanso para todos los escenarios? ¿En qué difieren las trayectorias asociadas a un escenario?

El punto de descanso, el valor al que convergen las trayectorias, de la variable de recuperación (w) para el primer escenario es de -30 mv para el primer sistema, -20 mv para el segundo y 78 mv para el tercero. Por otra parte, para el potencial de membrana (v), el primer escenario -75 mv y para el tercero -78 mv.

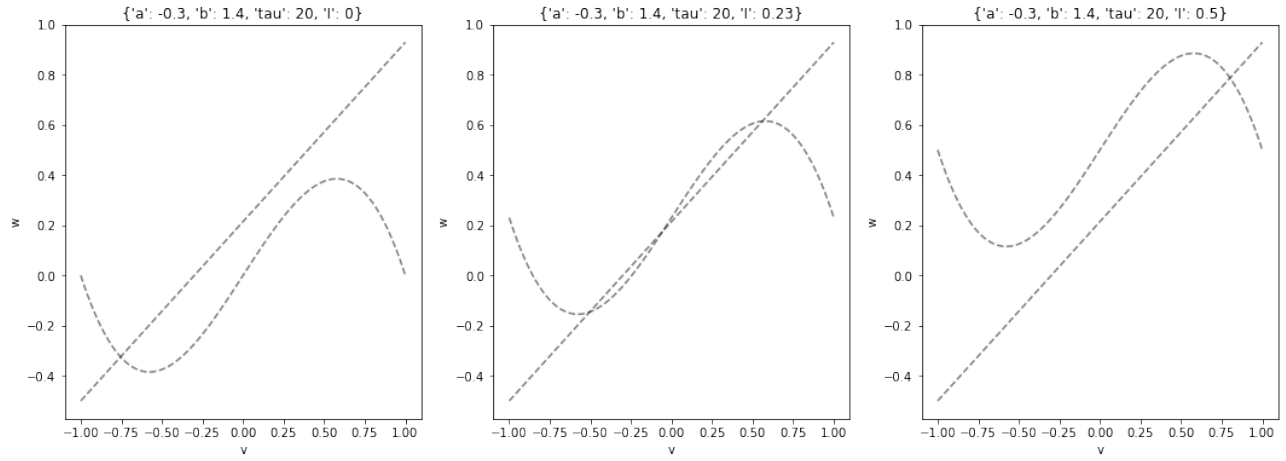
En este escenario será más dificil llegar al umbral, puesto que la corriente externa es demasiado alta, de manera que el sistema esta sobreexcitado y las células ya no son capaces de generar potenciales de acción. Por el contrario, en el segundo caso es más fácil generar potenciales de acción, como podemos comprobar por las oscilaciones.

4.2. Null-clinas

Las isoclinas o nullclinas, son las variedades en las que una de las componentes de nuestro sistema es nula.

Encuentre la ecuación de las líneas nulas para v y w.

Escribe las dos ecuaciones como w = F(v). ¿Qué tipo de funciones son las líneas nulas? Programa la función 8 para trazar las líneas nulas. Muestra las líneas nulas para los tres escenarios en tu documento.



En nuestro caso de estudio podemos observar dos nullclinas que toman una determinada forma. Este flujo puede ser horizontal o vertical según si hablamos de w-clinas o v-clinas respectivamente. La v-clina es representada por un polinomio de tercer orden, mientras que la w-clina, es un polinomio de primer orden. Además, observando las tres gráficas, podemos ver como las nullclinas intersectarán en distintos puntos, según los diferentes valores que tome la corriente de estimulación. Estos puntos de intersección corresponderían a los puntos fijos del sistema.

¿Cuántos puntos de intersección hay en cada escenario?

Tal y como hemos mencionado, las nullclinas representan las líneas donde el flujo es nulo en una de sus componentes. De esta forma, se da la aparición de un punto fijo en la intersección de estas.

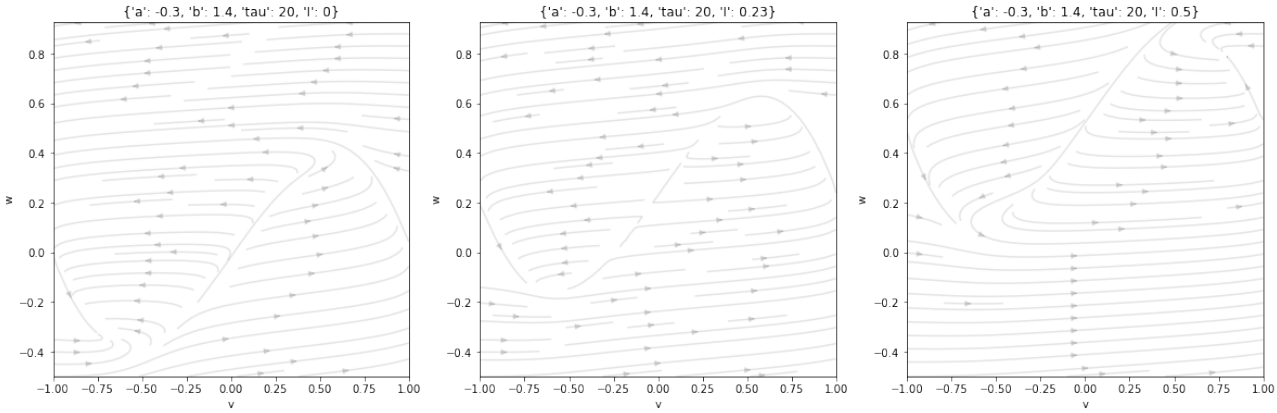
Observando el primer y último escenario, podemos contemplar como al haber una única intersección, hay un único punto fijo. Sin embargo, el segundo escenario presenta tres punto fijos al haber tres intersecciones.

4.3. Flujo

Calcula el flujo y dale la expresión en tu trabajo. Busca el trozo de código similar y añade la frase necesaria para ejecutarlo Muestre el flujo de los tres escenarios en su documento.

A continuación adjuntamos el código completado por nosotros:

```
def plot_vector_field(ax, param, xrange, yrange, steps=50):
    # Compute the vector field
    x = np.linspace(xrange[0], xrange[1], steps)     #""" vary "steps" points along x axis between xran
    y = np.linspace(yrange[0], yrange[1], steps)     #""" vary "steps" points along y axis between yran
    X,Y = np.meshgrid(x,y)   #""" compute a meshgrid"""

    dx,dy = fitzhugh_nagumo([X,Y],0, **param) #"""call fitzhugh_nagumo with t=0 and params"""

    # streamplot is an alternative to quiver
    # that looks nicer when your vector filed is
    # continuous.
    ax.streamplot(X,Y,dx, dy, color=(0,0,0,.1))

    ax.set(xlim=(xrange[0], xrange[1]), ylim=(yrange[0], yrange[1]))
```



De esta forma, logramos ver de una forma más clara el flujo que presenta nuestro sistema para los distintos valores de I (corriente de estimulación). En los siguientes apartados, sacaremos los puntos de equilibrio del sistema y los clasificaremos

4.4. Puntos de equilibrio

Los equilibrios se encuentran en el cruce entre la isoclina nula para v y la de w. Encuentre la ecuación polinómica verificada por los equilibrios del modelo. Obtén primero un polinomio en v que cruce las isoclinas nulas. A partir de sus coeficientes obtener las raíces. Completa el código. **Mostrar en una tabla las raíces de cada escenario.**

Según como hemos mencionado en apartado anteriores, el flujo en los puntos de equilibrio es nulo, por lo que la intersección de nuestras nullclinas es lo que da como resultado el conjunto de estos puntos fijos en cada escenario.

De esta forma, la función completada quedaría:

```
def find_roots(a,b,I, tau):
    coef = [1, 0, 1/b-1, -a/b-I]
    roots = [np.real(r) for r in np.roots(coef) if np.isreal(r)]

    return [[r, r-r**3+I] for r in roots] #""" the value of w for each v* """

eqnproot = {}

for i, param in enumerate(scenarios):
    eqnproot[i] = find_roots(**param)
    print (i, param,find_roots(**param))
```

Y obtenemos entonces la siguiente tabla:

```
0 {'a': -0.3, 'b': 1.4, 'tau': 20, 'I': 0}
[[-0.7547409174415924, -0.3248149410297081]]

1 {'a': -0.3, 'b': 1.4, 'tau': 20, 'I': 0.23}
[[0.5601499772018519, 0.6143928408584656],
[-0.5045483455831287, -0.14610596113080618],
[-0.05560163161872317, 0.17457026312948348]]

2 {'a': -0.3, 'b': 1.4, 'tau': 20, 'I': 0.5}
[[0.8013957389076585, 0.7867112420768972]]
```

4.5. Naturaleza de los puntos de equilibrio

La naturaleza local y la estabilidad del equilibrio vienen dadas por la linealización de la función de flujo. Para ello se utiliza la matriz jacobiana del flujo: Obtener la matriz jacobiana del flujo e implementarlo como una lista en python en el código adjunto.

Definimos la Jacobiana:

```
def jacobian_fitznagumo(v, w, a, b, tau, I):
    ====
        v (float): Membrane potential
        w (float): Recovery variable
        a,b (float): Parameters
        tau (float): Recovery timescale.
    Return: np.array 2x2"""
    return np.array([[-3*v**2+1, -1],[1/tau, -b/tau]])
```

Imprimimos la Jacobiana de nuestro sistema en particular. Ha sido obtenida por medio de las derivadas parciales y nos ayudará más adelante a clasificar nuestros puntos fijos sustituyendo estos en la matriz Jacobiana.

$$\begin{bmatrix} 1-3v^2 & -1 \\ \frac{1}{\tau} & -\frac{b}{\tau} \end{bmatrix}$$

Una vez ejecutado el código anterior, **muestra en una Tabla el valor del Jacobiano para todas las raíces en todos los escenarios. Calcula también, su determinante y su traza.** Además, complete la siguiente función para determinar la estabilidad de los puntos de equilibrio.

Ejecutamos el siguiente código sustituyendo los diferentes puntos fijos y clasificamos su estabilidad basándonos en los autovalores de nuestro Jacobiano.

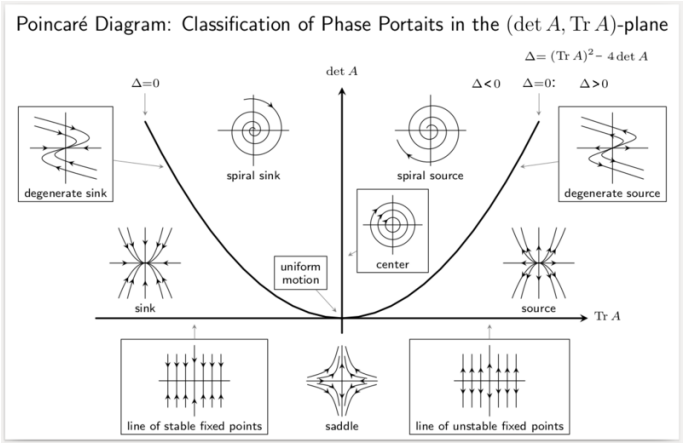
```
def stability(jacobian):
    eigv = np.linalg.eigvals(jacobian)

    if all(np.real(eigv)==0) and all(np.imag(eigv)!=0):
        nature = "Center"
    elif np.real(eigv)[0]*np.real(eigv)[1]<0:
        nature = "Saddle"
    else:
        stability = 'Unstable' if all(np.real(eigv)>0) else 'Stable'
        nature = stability + (' focus' if all(np.imag(eigv)!=0) else ' node')
    return nature
```

Para nuestros escenarios entonces sustituimos usando la función definida, y los clasificamos.

Obtenemos entonces la siguiente clasificación de puntos fijos:

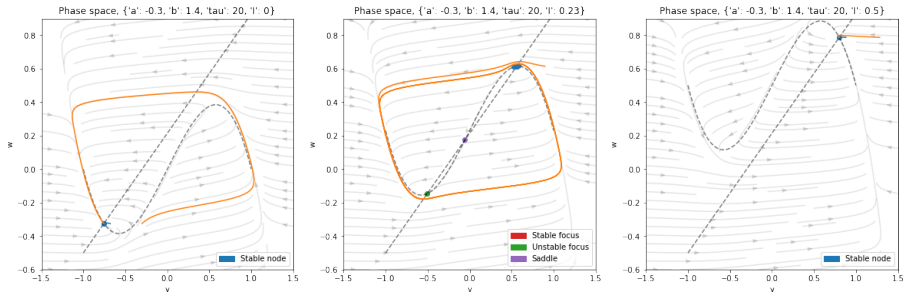
```
{0: [['Stable node',
   array([[-0.70890156, -1.        ],
          [ 0.05      , -0.07      ]])]],
 1: [['Stable focus',
   array([[ 0.05869601, -1.        ],
          [ 0.05      , -0.07      ]])],
  ['Unstable focus',
   array([[ 0.2362929, -1.       ],
          [ 0.05     , -0.07     ]])],
  ['Saddle',
   array([[ 0.99072538, -1.        ],
          [ 0.05      , -0.07      ]])]],
 2: [['Stable node',
   array([[-0.92670539, -1.        ],
          [ 0.05      , -0.07      ]])]]}
```

Podemos comprobar manualmente si son correctos. La clasificación se ha obtenido basandonos en la siguiente imagen:



4.6. Diagrama de fase completo

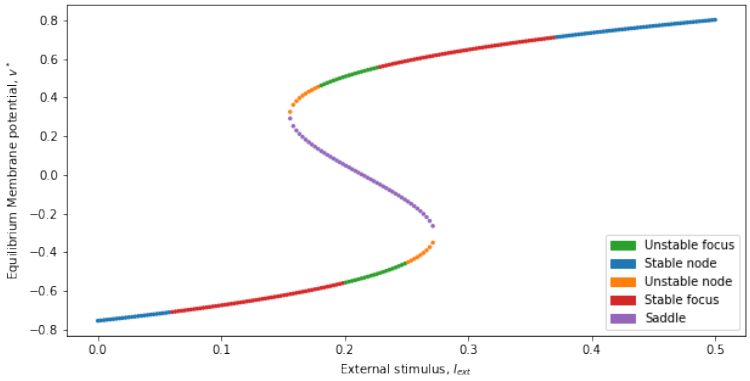
En este punto, debería poder trazar el diagrama de fase, incluidas las líneas nulas, los puntos de equilibrio y algunas trayectorias desde los puntos de equilibrio cuando sufren pequeñas perturbaciones. Muestre el diagrama de fase de todos los escenarios. Explique en detalle los resultados obtenidos para las trayectorias representadas.



Para entender las representaciones debemos centrarnos en las trayectorias de color naranjas. Estas nos ayudan a entender la evolución del sistema dependiendo de las condiciones iniciales. Las trayectorias pueden oscilacar, cayendo en un punto fijo estable o alejarse de los puntos fijos inestables.

5. Diagrama de bifurcación

Ahora, vamos a trazar el diagrama de bifurcación para v con respecto al parámetro I. Varíe el parámetro I y b entre el extremo específico del intervalo indicado.



5.1. Bifucación sobre el estímulo externo I

Muestre el diagrama de bifurcación en su documento. ¿Cuántas bifurcaciones puede detectar? ¿Que tipo de bifurcaciones son? Hay cuatro bifurcaciones de codim 1 en el diagrama: dos de tipo 'saddle-node' y dos bifurcaciones de Hopf (stable focus y unstable focus).

Muestre en su documento la trayectoria de equilibrio en el espacio determinante / traza jacobiana. ¿Qué tipo de trayectoria encontramos en este espacio en particular? Como podemos ver

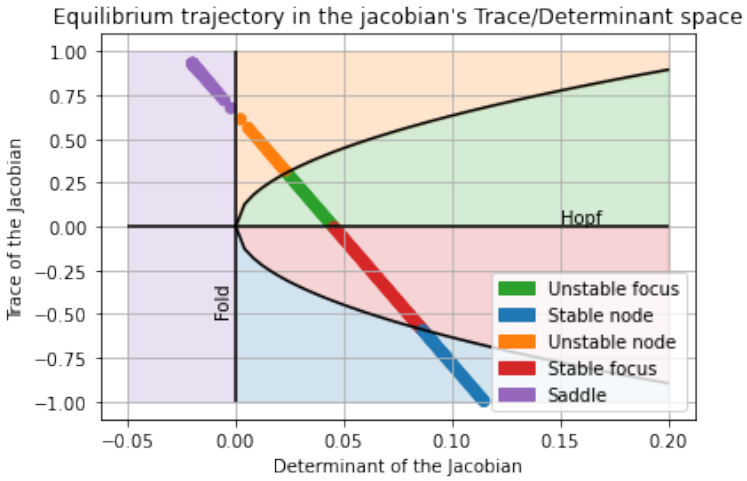
se trata de una trayectoria rectilínea por lo que tenemos una relación lineal entre traza y determinante.

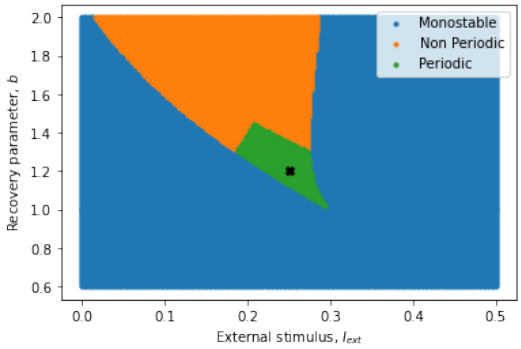
Estabilidad de los puntos de equilibrio

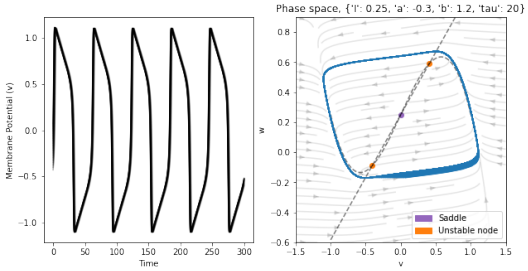
Como podemos ver este esquema es como uno de los esquemas que mostramos anteriormente en otro de los ejercicios. Aquí podemos ver claramente cinco regiones definidas donde según el determinante y la traza de nuestra Jacobiana en el punto fijo estudiado, podemos saber que tipo de estabilidad tendrá.

5.2. Bifurcación de Codim 2 en I y b

Muestre la bifurcación de Codim 2 en I y b. Analizar para qué par de valores (I, b) el sistema será periódico, no periódica y mono-estable. Trace una trayectoria en el tiempo y en el espacio de fase con las siguientes condiciones: I = 0,25, b = 1,2, $a = -0,3$ and = 20. **Comente los resultados**





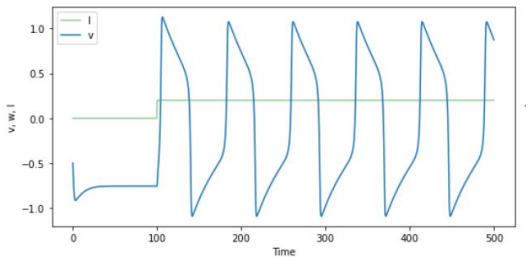


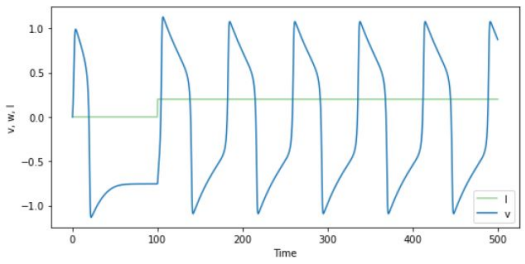
Como podemos ver obtenemos una solución periódica para aquellos valores de I y b pertenecientes a la zona verde. Al graficar la solución del dato marcado con forma de cruz vemos como efectivamente va a realizar trayectorias periódicas, por lo que tenemos un ciclo límite estable, por la dirección del flujo (flechas grises).

6. Sistema no-autónomo (opcional 1)

Hasta ahora hemos considerado el comportamiento del sistema bajo un estímulo constante I externa. Sin embargo, es posible extender este modelo a casos donde el estímulo es más complejo, haciendo que I externa sea una función del tiempo. Implementa la versión del modelo Fitzhugh Nagumo no-autónomo. Programa el código y muestra las 8 trayectorias. Hay 4 diferentes estímulos iniciados con 2 diferentes condiciones iniciales. Comenta los resultados.

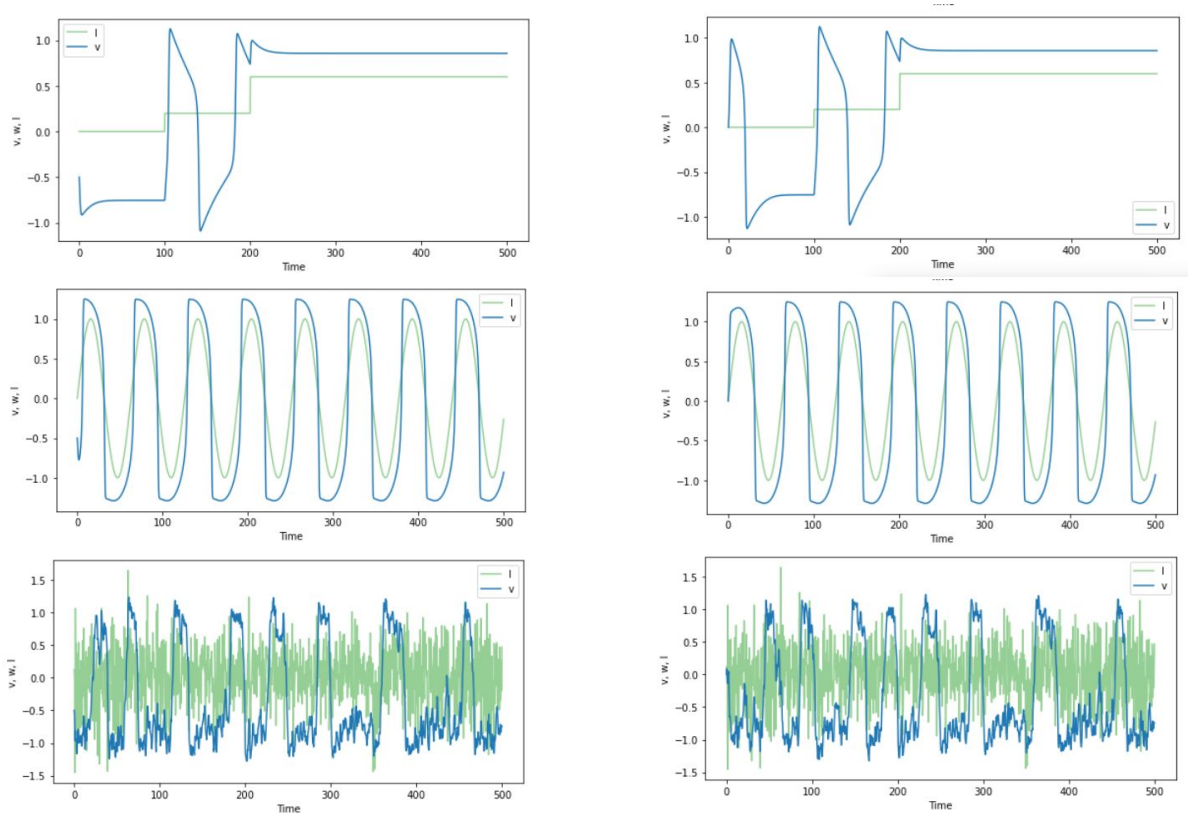
Las trayectorias obtenidas son las siguientes:





En las gráficas podemos comprobar como las diferentes condiciones iniciales afectan a las trayectorias resultantes a los estímulos de corriente externa. El potencial de membrana inicial de las gráficas de la izquierda es más negativo que las de la derecha, por lo que se necesitará una mayor corriente para que las células generen un potencial de acción.

Si comparamos los diferentes estímulos, en el primer escenario observamos una corriente de estimulación externa constante en el tiempo a partir de t=100, que genera unos potenciales de acción periódicos. Si mirasemos el diagrama de fase, observaríamos un ciclo límite estable. En el segundo escenario, vemos como inicialmente la corriente es igual que en el caso anterior, de manera que se genera un único potencial de acción. No obstante, en t=200 aumentamos la corriente continua tanto que la variable de recuperación (w) no es capaz de compensar el



estímulo recibido y el potencial de la célula se mantiene constante a un voltaje superior tanto al de umbral como al de reposo. La célula no dispara potenciales de acción, pero se mantiene en un estado tónico despolarizado, lo que se correspondería punto fijo estable en el diagrama de fase. Para el tercer escenario, observamos una generación de potenciales de acción periódicos, que se corresponde a la aparición del ciclio límite estable, debido a la variación senoidal de la corriente de estimulación. La principal diferencia que observamos entre el uso de una corriente continua y otra alterna es la forma de los potenciales. Por último, en el cuarto escenario, encontramos ruido estocástico. La corriente de estimulación basal no es suficiente como para generar un potencial. No obstante, cuando se introduce el ruido intrínseco del sistema (causado por las desincronización de las propias células cardíacas), es posible conseguir suficiente estímulo como para que el sistema entre en la órbita cerrada y se observe una activación 'periódica'.

7. Ecuación diferencial estocástica (Opcional 2)

Hasta ahora hemos visto sistemas deterministas de estado continuo en forma de ecuaciones diferenciales ordinarias (EDO). Su contrapartida estocástica son las ecuaciones diferenciales estocásticas (EDES).

Utilizaremos el método [Euler-Maruyama]. Completa el Método de Euler-Maruyama implementado. Además, completa el código para llamar a la ecuación de Euler-Maruyama.

Muestre el diagrama de fases final con las trayectorias estocásticas de los dos primeros escenarios. Compare estos diagramas con los obtenidos en la ODE no estocástica.

Este apartado nos ha generado problemas a la hora de ejecutar el código. Lo comentamos con David en clase y nos indicó que lo dejáramos así.

```
def euler_maruyama(flow, noise_flow, y0, t) :
    ''' Euler-Maruyama intergration.

    Args:
        flow (function): deterministic component of the flow (f(Yt,t))
        noise_flow (function): stochastic component of the flow (g(Yt,t))
        y0 (np.array): initial condition
        t (np.array): time points to integrate.
```

```
    Return the Euler Maruyama approximation of the SDE trajectory defined by:

    y(t) = f(Y(t),t)dt + g(Yt,t)dBt
    y(0) = y0
    '''
    y = np.zeros((len(t),len(y0)))
    y[0] = y0
    for n,dt in enumerate(np.diff(t),1):
        y[n] = y[n-1] + flow(y[n-1],dt) * dt + noise_flow(y[n-1],dt) *
        np.random.normal(0,np.sqrt(dt))
    return y



# Do the simulations.
# Remember that we define f as the partial application of fitzhugh_nagumo.
initial_conditions = [(-0.5,-0.1), [0, -0.16016209760708508]]
time_s = np.linspace(0, 1000, num=10000)
noise_flow = lambda y,t: 0.04
print('noise flow ', noise_flow)
stochastic = {}
trajectory = {}
for i,param in enumerate(scenarios[:2]):
    for j, ic in enumerate(initial_conditions):
        # i is key and j is value in the trajectory dictionary
        flow =  partial(fitzhugh_nagumo, **param)
        print("flow ", flow)
        stochastic[i, j] = euler_maruyama(flow, noise_flow, y0=ic, t=time_s)
        trajectory[i,j] = scipy.integrate.odeint(flow, y0=ic, t=time_s)



# Draw the trajectories.
fig, ax = plt.subplots(2, 2, figsize=(20,10))
initial_conditions = [(-0.5,-0.1), [0, -0.16016209760708508]]
for i,param in enumerate(scenarios[:2]):
    for j, ic in enumerate(initial_conditions):
        ax[i, j].set(xlabel='Time', ylabel='v, w', title='Trajectory of v, w, {}
        with init = {}'.format(param, ic),
                    xlim=(0, time_s[-1]), ylim=(-1.5, 1.5))
        ax[i, j].plot(time_s,stochastic[i, j][:,0], label='v (SDE)')
        ax[i, j].plot(time_s,trajectory[i, j][:,0], label='v (ODE)', color='C0', ls=":")
ax[0, 0].legend()



xlimit = (-1.3, 1.3)
ylimit = (-0.5, 0.5)
fig, ax = plt.subplots(1,2, figsize=(12,5))
for i, param in enumerate(scenarios[:2]):
    ax[i].set(xlabel='v', ylabel='w', title="Phase space, {}".format(param))
    plot_vector_field(ax[i], param, xlimit, ylimit)
    plot_vector_field(ax[i], param, xlimit, ylimit)
    plot_isocline(ax[i], **param, vmin=xlimit[0], vmax=xlimit[1])
    for j, ic in enumerate(initial_conditions):
        ax[i].plot(stochastic[i, j][:,0], stochastic[i, j][:,1],lw=1)
```