Spiral waves

Contents

List of Figures

1 Introduction

Spiral waves are travelling waves that rotate outward from a center in a spiral and have been observed in various biological systems (wiki [4]). The typical examples are the Belousov-Zabotinski chemical reaction, heart electrophysiology, they are observed on the retina and in the cerebral cortex (source article???).

There are systems that their solutions produce spiral waves. The solutions for FitzHugh-Nagumo system in two dimensions where constant steady state is Turing unstable are spiral waves. The other example could be three species competition systems.

2 FitzHugh-Nagumo (FHN) model

The FHN model is an example of a relaxation oscillator (wiki [5]). It is a two-dimensional simplification of the Hodgkin-Huxely model of spike generation in squid giant axons (scholarpedia [1]).

The Hodgkin-Huxely model desribes how action potentials in neurons are initiated and propagated. It is approximating the electrical characteristics of excitable cells such as neurons and cardiac myocytes (wiki [6]).

In this project I will be referring to FHN model as a system of equations:

$$\begin{cases} \frac{\partial}{\partial t}u(\mathbf{x},t) - d\Delta u(\mathbf{x},t) &= u(\mathbf{x},t)\left(1-u(\mathbf{x},t)\right)\left(u(\mathbf{x},t)-a\right) - v(\mathbf{x},t), \quad t \geq 0, \mathbf{x} \in [0,1]^2 \\ \frac{\partial}{\partial t}v(\mathbf{x},t) &= \varepsilon(\beta u(\mathbf{x},t) - \gamma v(\mathbf{x},t) - \delta). \end{cases}$$

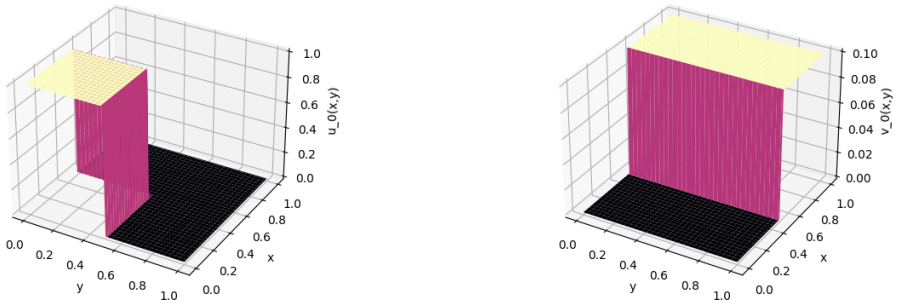
Where u is the membrane potential, v is a recovery variable and $d, a, \varepsilon, \beta, \gamma, \delta$ are constants.

2.1 Parameters

The parameters of the model were chosen (by the authors of the article [2]) to be $d = 10^{-5}, a = 0.1, \varepsilon = 0.01, \beta = 0.5, \gamma = 1, \delta = 0$, to introduce Turing instability in some of the equilibrium points (It will be discussed in later sec. 5).

2.2 Starting and boundary conditions

The starting conditions are selected to generate a spiral wave – there are different constant values in four corners of the $[0,1]^2$ square. The trivial equilibrium state $(\bar{u}, \bar{v}) = (0,0)$ was perturbed: lower-left quarter was set to $u = 1$ and upper half part was set to $v = a$. The starting conditions are presented in fig. 2.1.



(a) $u_0(\mathbf{x})$ (b) $v_0(\mathbf{x})$

Figure 2.1: Starting conditions.

The boundary conditions are not needed for variable $v(\mathbf{x},t)$, because it does not have the heat part. For the variable $u(\mathbf{x},t)$ it is assumed that there is no flow through the boundary (it is the most common assumption in biology), so:

$$\vec{\eta}(\mathbf{x}) \cdot \nabla u(\mathbf{x},t) = 0, \quad \mathbf{x} \in \partial\Omega.$$

Where $\partial\Omega$ is a boundary of the square $[0,1]^2$, $\vec{\eta}(\mathbf{x})$ is a normal vector of a boundary (perpendicular to the boundary) at $\mathbf{x}$ and $\cdot$ is a dot (scalar) product.

3 Equilibrium points

The equilibrium points are special points for system of differential equations. These are points that give constant solution for a system, so the derivatives are equal 0. In case of FHN model:

$$\begin{cases} 0 = u\,(1-u)\,(u-a) - v \\ 0 = \beta u - v. \end{cases}$$

$$\begin{cases} v = \beta u \\ 0 = u\,((1-u)\,(u-a) - \beta) = u(-u^2 + u(1+a) - (a+\beta)). \end{cases}$$

So the two non-trivial (other than $(\bar{u}, \bar{v}) = (0,0)$) solutions of this system depend on finding roots of the equation $0 = u^2 - u(1+a) + (a+\beta)$. Given the parameters we have:

$$u_{1,2} = \frac{1 + a \pm \sqrt{(a-1)^2 - 4\beta}}{2} \approx \frac{1}{2} \pm \frac{i}{2}, \quad a = 0.1, \beta = 0.5$$

And then:

$$\begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \vee \begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix}_2 = \begin{pmatrix} 0.55 - 0.545436i \\ 0.275 - 0.272718i \end{pmatrix} \vee \begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix}_3 = \begin{pmatrix} 0.55 + 0.545436i \\ 0.275 + 0.272718i \end{pmatrix}.$$

4 Linearization and stability

The FHN model is non-linear so to find if equilibrium points are stable solutions one needs to linearize the system of equations. To do that, one needs to compute a matrix of partial derivatives:

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f}{\partial u} & \frac{\partial f}{\partial v} \\ \frac{\partial g}{\partial u} & \frac{\partial g}{\partial v} \end{pmatrix} = \begin{pmatrix} -3u^2 + 2(1+a)u - a & -1 \\ \varepsilon\beta & -\varepsilon \end{pmatrix}. \tag{4.1}$$

Where f, g are non-linear functions equal to respective derivates, so in the FHN case:

$$\begin{cases} \frac{\partial}{\partial t} u = f(u,v) = u\,(1-u)\,(u-a) - v \\ \frac{\partial}{\partial t} v = g(u,v) = \varepsilon(\beta u - v). \end{cases}$$

Here the part that is responsible for heat diffusion is missing, because adding it does not change the stability and calculations are easier. To decide if equilibrium point is stable one needs to plug in this point to the matrix $\mathbf{J}$ obtained in 4.1 and find if its eigenvalues (real parts) are negative. It is sufficient to use determinant and trace of $\mathbf{J}$ to get signs of eigenvalues:

$$\det(\mathbf{J}) = \varepsilon(3u^2 - 2(1+a)u + a + \beta), \quad \mathrm{Tr}(\mathbf{J}) = -3u^2 + 2(1+a)u - a - \varepsilon.$$

The equilibrium point is stable if:

$$\det(\mathbf{J}) > 0 \wedge \mathrm{Tr}(\mathbf{J}) < 0.$$

One can use those conditions if the equilibrium points are real, so the trivial equilibrium point is stable because:

$$\det(\mathbf{J}) = a + \beta > 0, \quad \mathrm{Tr}(\mathbf{J}) = -(a+\varepsilon) > 0 \text{ if } a, \beta, \varepsilon > 0.$$

For other points one needs to compute its eigenvalues and compare the signs of real parts.

5 Turing instability

The Turing instability happens when in a problem **without** diffusion equilibrium points are stable but not in a problem **with** diffusion. In a reaction-diffusion (R-D) problem:

$$\begin{cases} u_t = d_u \Delta u + f(u,v) \\ v_t = D_v \Delta v + g(u,v) \\ \vec{\eta}(\mathbf{x}) \cdot \nabla u(\mathbf{x},t) = 0, & \mathbf{x} \in \partial\Omega \\ \vec{\eta}(\mathbf{x}) \cdot \nabla v(\mathbf{x},t) = 0, & \mathbf{x} \in \partial\Omega \\ u(\mathbf{x},0) = u_0(\mathbf{x}) \\ v(\mathbf{x},0) = v_0(\mathbf{x}). \end{cases}$$

Where $d_u, D_v > 0$ are diffusion coefficients for respective functions, $\partial\Omega$ is a boundary of the square $[0,1]^2$, $\vec{\eta}(\mathbf{x})$ is a normal vector of a boundary. In this case the necessary condition for Turing instability is:

Theorem 5.1 (Necessary condition) *The necessary condition for Turing instability is:*

$$d_u < -\frac{f_u}{g_v} D_v.$$

Where partial derivatives of f, g are computed at equilibrium point.

Theorem 5.2 (Sufficient condition) *Let γ_c be a positive root of the equation:*

$$f_u^2\gamma^2 + 2(2f_v g_u - f_u g_v)\gamma + g_v^2 = 0.$$

And let $0 < \xi_1 < \xi_2$ be roots of the equation:

$$\gamma\xi^2 - (\gamma f_u - g_v)\xi + (f_u g_v - f_v g_u) = 0, \quad \gamma = \frac{D_v}{d_u}.$$

If there exist γ_c and k such that:

$$\gamma_c < \frac{D_v}{d_u} \quad \wedge \quad \frac{\xi_1}{(k\pi)^2} < d_u < \frac{\xi_2}{(k\pi)^2}.$$

Then Turing instability happens. The partial derivatives of f, g are computed at equilibrium point.

The FHN model (with switched variables to match the theorems):

$$\begin{cases} u_t = \varepsilon(\beta v - u) = 0 \cdot \Delta u + f(u,v) \\ v_t = d\Delta v + v\,(1-v)\,(v-a) - u = d\Delta v + g(u,v). \end{cases}$$

$$d_u = 0,\ D_v = d, \quad \mathbf{J} = \begin{pmatrix} \frac{\partial f}{\partial u} & \frac{\partial f}{\partial v} \\ \frac{\partial g}{\partial u} & \frac{\partial g}{\partial v} \end{pmatrix} = \begin{pmatrix} -\varepsilon & \varepsilon\beta \\ -1 & -3v^2 + 2(1+a)v - a \end{pmatrix}. \tag{5.1}$$

Then necessary condition holds if:

$$0 < -\frac{-d\varepsilon}{-3v^2 + 2(1+a)v - a} \overset{(0,0)}{=} -\frac{d\varepsilon}{a}.$$

All of the parameters are positive, so this inequality does not hold, which means, that equilibrium point $(0,0)$ cannot be Turing unstable.

6 Numerical methods

Since there is no way to obtain analytical solutions for FHN model, the only option left is to use numerical methods. Using finite differences method one would formulate one step of iteration:

$$\begin{aligned} u(x,y,t+dt) = u(x,y,t) + \frac{d}{\tau}\frac{dt}{dx^2}(u(x+dx,y,t) + u(x-dx,y,t) + u(x,y+dy,t) + \\ + u(x,y-dy,t) - 4u(x,y,t)) + dt \cdot f(u(x,y,t), v(x,y,t)), \\ v(x,y,t+dt) = v(x,y,t) + dt \cdot g(u(x,y,t), v(x,y,t)). \end{aligned}$$

Start of this iterative method is given by starting condition $u(\mathbf{x},0) = u_0(\mathbf{x})$ and $v(\mathbf{x},0) = v_0(\mathbf{x})$. The boundaries are calculated using Neumann boundary conditions:

$$u(\mathbf{x},t) = u(x_1, x_2, \ldots, x_i \pm dx, x_{i+1}, \ldots, x_n, t) + dx \cdot f(\mathbf{x},t), \quad \mathbf{x} \in \partial\Omega.$$

Where x_i is on the boundary and the $\pm$ sign depends on which boundary (dx must be substracted or added in such a way that $x_i \pm dx$ is in Ω).

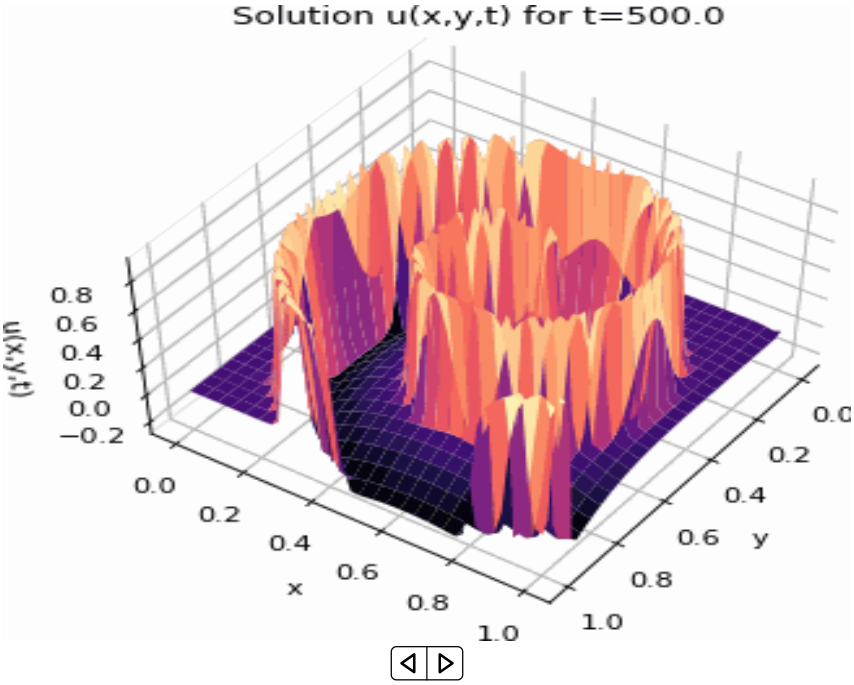


Figure 7.1: Solution for $u(x, y, t)$

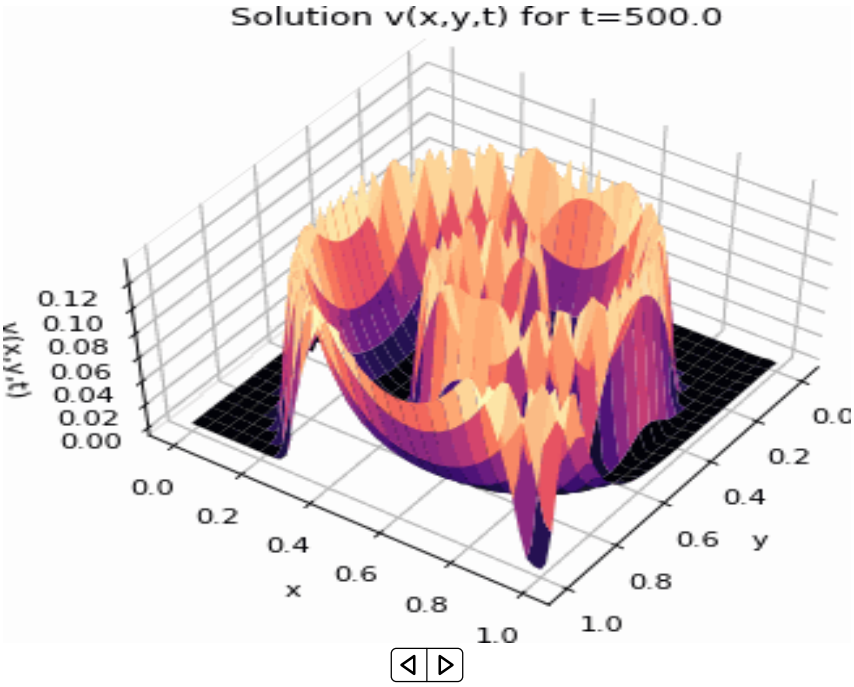


Figure 7.2: Solution for $v(x, y, t)$

7 Numerical solutions

Algorithm described above (sec. 6) was implemented in Python3, using NumPy package and calculations were done using double precision. The initial conditions were chosen so that they will perturb the stable equilibrium and the solutions will produce spiral waves. The animation of solutions are presented in fig. 7.1 and 7.2.

8 Conclusions

Indeed the stable equilibrium point losses its stability when adding the diffusion part and spiral waves are produced. It is a special kind of instability, that can make intriguing patterns. To find them, one needs to find suitable parameters, which give the Turing instability. Besides the parameters it is important to find the right initial conditions – in the case of aforementioned model, it gives its clockwise motion.

The Turing instability (and other instabilities like the one mentioned in this project) provides new insights to understanding of the nature. Now we know why and when some patterns appear in biological and chemical processes instead of just constant states.

References

[1] Fitzhugh-nagumo model - scholarpedia. `http://www.scholarpedia.org/article/FitzHugh-Nagumo_model`. (Accessed on 01/17/2022).

[2] A. Bueno-Orovio, D. Kay, and K. Burrage. Fourier spectral methods for fractional-in-space reaction-diffusion equations. *BIT Numerical mathematics*, 54(4):937–954, 2014.

[3] B. Perthame. Parabolic equations in biology. In *Parabolic Equations in Biology*, pages 101–102. Springer, 2015.

[4] Wikipedia contributors. Spiral wave — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Spiral_wave&oldid=927103832`, 2019. [Online; accessed 17-January-2022].

[5] Wikipedia contributors. Fitzhugh–nagumo model — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=FitzHugh%E2%80%93Nagumo_model&oldid=1037464531`, 2021. [Online; accessed 17-January-2022].

[6] Wikipedia contributors. Hodgkin–huxley model — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Hodgkin%E2%80%93Huxley_model&oldid=1063643443`, 2022. [Online; accessed 17-January-2022].

Appendices

Appendix 1 Notes about another model

After having some problems with original model given to me as an assignment, together with Szymon Cygan we've decided to use another model, one that gave more information about its conditions to produce the spiral waves. The original model from the paper ([3]) has form:

$$\begin{cases} \tau \frac{\partial}{\partial t} u(\mathbf{x},t) - d\Delta u(\mathbf{x},t) = u(\mathbf{x},t) - \frac{u^3(\mathbf{x},t)}{3} - v(\mathbf{x},t), \quad t \geq 0, \mathbf{x} \in [0,1]^2 \\ \frac{\partial}{\partial t} v(\mathbf{x},t) = -u(\mathbf{x},t) - c - \gamma v(\mathbf{x},t). \end{cases}$$

The parameters of the model were chosen to be $c = 0.2, d = 10^{-5}, \gamma = 0.5$. The equilibrium points are obtained:

$$\begin{cases} 0 = u(\mathbf{x},t) - \frac{u^3(\mathbf{x},t)}{3} - v(\mathbf{x},t) \\ 0 = -u(\mathbf{x},t) - c - \gamma v(\mathbf{x},t). \end{cases}$$

$$\begin{cases} v = u - \frac{u^3}{3} \\ 0 = -u - c - \gamma \left(u - \frac{u^3}{3} \right). \end{cases}$$

So the solutions of this system depend on finding roots of (numerically) the equation $\frac{\gamma}{3}u^3 - (1+\gamma)u - c$. Given the parameters: $c = 0.2, d = 10^{-5}, \gamma = 0.5$ we have:

$$\begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix}_1 = \begin{pmatrix} -2.931 \\ 5.4619 \end{pmatrix} \vee \begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix}_2 = \begin{pmatrix} -0.1336 \\ -0.1328 \end{pmatrix} \vee \begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix}_3 = \begin{pmatrix} 3.0646 \\ -6.5292 \end{pmatrix}.$$

Linearization matrix:

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f}{\partial u} & \frac{\partial f}{\partial v} \\ \frac{\partial g}{\partial u} & \frac{\partial g}{\partial v} \end{pmatrix} = \begin{pmatrix} (1-u^2)/\tau & -\tau^{-1} \\ -1 & -\gamma \end{pmatrix}. \tag{1.1}$$

Where f, g are non-linear functions equal to respective derivates, so in the FHN case:

$$\begin{cases} \frac{\partial}{\partial t}u = f(u,v) = u\frac{1}{\tau} - \frac{u^3}{3\tau} - v\frac{1}{\tau} \\ \frac{\partial}{\partial t}v = g(u,v) = -u - c - \gamma v. \end{cases}$$

$$\det(\mathbf{J}) = \left((u^2-1)\gamma + 1\right)/\tau, \quad \mathrm{Tr}(\mathbf{J}) = \left(1 - u^2 - \gamma\tau\right)/\tau.$$

Given equilibrium points computed before one can conclude the that points 1 and 3 are stable.

The FHN model (with switched variables to match the theorems) and its linearization matrix:

$$\begin{cases} u_t = -v - c - \gamma u = 0 \cdot \Delta u + f(u,v) \\ v_t = \frac{d}{\tau}\Delta v + v\frac{1}{\tau} - \frac{v^3}{3\tau} - u\frac{1}{\tau} = \frac{d}{\tau}\Delta v + g(u,v). \end{cases}$$

$$d_u = 0,\ D_v = \frac{d}{\tau}, \quad \mathbf{J} = \begin{pmatrix} \frac{\partial f}{\partial u} & \frac{\partial f}{\partial v} \\ \frac{\partial g}{\partial u} & \frac{\partial g}{\partial v} \end{pmatrix} = \begin{pmatrix} -\gamma & -1 \\ -\tau^{-1} & (1-v^2)/\tau \end{pmatrix}. \tag{1.2}$$

Then necessary condition holds if:

$$0 < -\frac{-\gamma\tau}{(1-v^2)}\frac{d}{\tau} = \frac{d\gamma}{(1-v^2)} \Leftrightarrow v \in (-1,1).$$

So the only equilibrium point that could be Turing unstable is the second one, which is unstable, so Turing instability cannot happen.

Appendix 2 Python code

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

#
    ##########################################################################################

##################################EQUILIBIRIUM
    ##############################################
def FitzHughNagumo_equilibrium(a=0.1, beta=0.5):
    u = np.roots([-1, 1+a, -(a+beta), 0])
    v = lambda x: beta*x
    return np.array([u, v(u)])

def FitzHughNagumo_part_der_matrix(uv, a=0.1, beta=0.5):
    return np.array([[-3*uv[0]**2 + 2*(1+a)*uv[0] - a, -1], [beta, -1]])

def FitzHughNagumo_point_stability(uv, a=0.1, beta=0.5):
    A = FitzHughNagumo_part_der_matrix(uv, a=a, beta=beta)
    return (np.linalg.det(A) > 0 ) and (np.trace(A) < 0)

def FitzHughNagumo_stability(tau=0.01, c=0.2, gamma=0.5):
    eq_points = FitzHughNagumo_equilibrium(c=c, gamma=gamma)
    stab = np.apply_along_axis(FitzHughNagumo_point_stability, axis=0, arr=
        eq_points, tau=tau, gamma=gamma)
    return np.append(eq_points, stab.reshape(1, 3), axis=0)

eq_points = FitzHughNagumo_equilibrium()
print(FitzHughNagumo_point_stability(eq_points[:,1]))
```

```
#
    ##########################################################################################

##########################################NUMERICAL BOUNDARIES
    ################################################
def dirichlet_boundary(X1, X2, Y, boundaries = [None, None, None, None]):
    if (boundaries[0] is not None):
        Y[:, -1, :] = boundaries[0](X1[-1, :], X2[-1, :])
    if (boundaries[1] is not None):
        Y[:, :, -1] = boundaries[1](X1[:, -1], X2[:, -1])
    if (boundaries[2] is not None):
        Y[:, 0, :] = boundaries[2](X1[0, :], X2[0, :])
    if (boundaries[3] is not None):
        Y[:, :, 0] = boundaries[3](X1[:, 0], X2[:, 0])
    return Y

def neumann_boundary(i, t, X1, X2, Y, boundaries = [None, None, None, None
    ], dx=0.1):
    if (boundaries[0] is not None):
        Y[i + 1, -1, 1:-1] = Y[i + 1, -2, 1:-1] + dx * boundaries[0](t[i +
            1], X1[-1, :], X2[-1, :])
    if (boundaries[1] is not None):
        Y[i + 1, :, -1] = Y[i + 1, :, -2] + dx * boundaries[1](t[i + 1], X1
            [:, -1], X2[:, -1])
    if (boundaries[2] is not None):
        Y[i + 1, 0, 1:-1] = Y[i + 1, 1, 1:-1] + dx * boundaries[2](t[i +
            1], X1[0, :], X2[0, :])
    if (boundaries[3] is not None):
        Y[i + 1, :, 0] = Y[i + 1, :, 1] + dx * boundaries[3](t[i + 1], X1
            [:, 0], X2[:, 0])
    return Y

#
    ##########################################################################################

##########################################NUMERICAL SOLUTIONS
    ################################################
def FitzHughNagumo_2dim_solution(u_0, v_0, tau=0.01, c=0.2, d=10**(-5),
    gamma=0.5, T_max=1, x1 = [0,1], x2 = [0,1], dt=0.01, dx=0.1,
                                     u_dir=[None, None, None, None], u_neu=[
                                         None, None, None, None], v_dir=[None,
                                         None, None, None], v_neu=[None, None,
                                         None, None]):
    f = lambda u,v: u*(1-u)*(u-0.1) -v #(u - u**3 / 3 - v) / tau #
    g = lambda u,v: 0.01*(0.5*u-v) #-u -c -gamma*v #
    D = d/tau
    x1 = np.arange(x1[0], x1[1] + dx, step=dx)
    x2 = np.arange(x2[0], x2[1] + dx, step=dx)
    t = np.arange(0, T_max + dt, step=dt)
    U = np.zeros((t.size, x1.size, x2.size))
    V = np.zeros((t.size, x1.size, x2.size))
    X1, X2 = np.meshgrid(x1, x2)

    #Starting conditions
    U[0, :, :] = u_0(X1, X2)
    V[0, :, :] = v_0(X1, X2)
```

```
    #Dirichlet boundaries
    U = dirichlet_boundary(X1=X1, X2=X2, Y=U, boundaries=u_dir)
    V = dirichlet_boundary(X1=X1, X2=X2, Y=V, boundaries=v_dir)

    for i in range(0, t.size - 1):
        #Heat part
        U[i + 1, 1:-1, 1:-1] = U[i, 1:-1, 1:-1] + D * (dt / dx ** 2) * (U[i
            , :-2, 1:-1] + U[i, 2:, 1:-1] + U[i, 1:-1, :-2] + U[i, 1:-1, 2:]
             - 4 * U[i, 1:-1, 1:-1]) + dt * f(U[i, 1:-1, 1:-1], V[i, 1:-1,
            1:-1])

        #Neumann boundaries
        U = neumann_boundary(i=i, t=t, X1=X1, X2=X2, Y=U, boundaries=u_neu)
        #V = neumann_boundary(i=i, t=t, X1=X1, X2=X2, Y=V, boundaries=v_neu
            )

        #Non-linear part
        V[i + 1, :, :] = V[i, :, :] + dt * g(U[i, :, :], V[i, :, :])

    return t, X1, X2, U, V

def u_0(X1, X2):
    n = X1.shape[0]
    k = X1.shape[1]
    U = np.zeros((n,k))
    U[:int(n/2), :int(k/2)] = 1#(1/np.sqrt(2)) # III quarter
    U[:int(n / 2), int(k / 2):] = 0#-(1/np.sqrt(2)) # II quarter
    U[int(k / 2):, :int(k / 2)] = 0#-(1/np.sqrt(2)) # IV quarter
    U[int(k / 2):, int(k / 2):] = 0#-(1/np.sqrt(2)) # I quarter
    return U

def v_0(X1, X2):
    n = X1.shape[0]
    k = X1.shape[1]
    V = np.zeros((n,k))
    V[:int(n/2), :int(k/2)] = 0#-(1/np.sqrt(2)) # III quarter
    V[:int(n / 2), int(k / 2):] = 0#-(1/np.sqrt(2)) # II quarter
    V[int(k / 2):, :int(k / 2)] = 0.1#0 # IV quarter
    V[int(k / 2):, int(k / 2):] = 0.1#0 # I quarter
    return V

barrier = lambda *args: 0

#
    ##########################################################################################

##########################################ANIMATIONS
    ###########################################################
def animation(t, X1, X2, U, dt=0.01, dx=0.1, dx_skip=2, dt_skip=10, name='U
    '):
    fig = plt.figure()
    ax = plt.axes(projection='3d')

    ax.view_init(50, 35)
    ind = np.arange(X1.shape[1], step=dx_skip)
    def animate(t):
        ax.collections.clear()
```

```
        ax.plot_surface(X1[ind, :][:, ind], X2[ind, :][:, ind], U[:, ind,
            :][t, :, ind], cmap='magma', edgecolor='none')
        ax.set_title('Solution '+name+'(x,y,t) for t='+str(round(t*dt, 2)))
        ax.set_xlabel('y')
        ax.set_ylabel('x')
        ax.set_zlabel(name+'(x,y,t) ')
        return ax

    anim = FuncAnimation(fig, animate, frames=np.arange(t.size, step=
        dt_skip), interval=10)
    anim.save('MD_proj3_sol_FHN_'+name+'.gif', dpi=80, fps=60)
    #plt.show()

#t, X1, X2, U, V = FitzHughNagumo_2dim_solution(u_0=u_0, v_0=v_0, dx=0.01,
    dt=0.01, T_max=10, tau=1,u_neu=[barrier, barrier, barrier, barrier])
#animation(t, X1, X2, U, dx=0.01, dt=0.01, dt_skip=500, dx_skip=2, name='u
    ')
#animation(t, X1, X2, V, dx=0.01, dt=0.01, dt_skip=500, dx_skip=2, name='v
    ')

#
    ##########################################################################################

##########################################BIN
    ############################################################
'''
def FitzHughNagumo_equilibrium(c=0.2, gamma=0.5):
    u = np.roots([gamma/3, 0, -(1+gamma), -c])
    v = lambda x: x - x**3/3
    return np.array([u, v(u)])

def FitzHughNagumo_part_dev_matrix(uv, tau=0.01, gamma=0.5):
    return np.array([[(1-uv[0]**2)/tau, -1/tau], [-1, -gamma]])

def FitzHughNagumo_point_stability(uv, tau=0.01, gamma=0.5):
    A = FitzHughNagumo_part_dev_matrix(uv, tau=tau, gamma=gamma)
    return (np.linalg.det(A) > 0 ) and (np.trace(A) < 0)

def FitzHughNagumo_stability(tau=0.01, c=0.2, gamma=0.5):
    eq_points = FitzHughNagumo_equilibrium(c=c, gamma=gamma)
    stab = np.apply_along_axis(FitzHughNagumo_point_stability, axis=0, arr=
        eq_points, tau=tau, gamma=gamma)
    return np.append(eq_points, stab.reshape(1, 3), axis=0)
'''
```