APÉNDICE - Prueba de evaluación continua

Juan M. Martínez Sánchez¹

¹Introducción a la ciencia no lineal

1. Relación entre r_v , r_w y las cantidades $\frac{D_v \Delta \tau}{(\Delta \chi)^2}$ y $\frac{D_w \Delta \tau}{(\Delta \chi)^2}$

Partiendo de las ecuaciones del sistema FHN incluyendo difusión:

$$\frac{dv}{dt} = D_v \Delta v + v(v - a)(1 - v) - w \quad (1)$$

$$\frac{dw}{dt} = D_w \Delta w + \varepsilon(v - bw) \quad (2)$$

Y sustituyendo las expresiones discretizadas tanto para el laplaciano y el término de reacción como para la derivada temporal obtenemos para la ecuación correspondiente a la variable v :

$$\frac{v(x, \tau + 1) - v(x, \tau)}{\Delta \tau} = D_v \frac{v(x + 1, \tau) + v(x - 1, \tau) - 2v(x, \tau)}{\Delta \chi^2} + Reac[v(i, \tau + 1)] \quad (3)$$

donde

$$Reac[v(i, \tau + 1)] = v(i, \tau)[v(i, \tau) - a](1 - v(i, \tau)) - w(i, \tau) \quad (4)$$

Despejando $v(x, \tau + 1)$ de la ecuación (3) obtenemos por tanto:

$$v(x, \tau + 1) = v(x, \tau) + \frac{D_v \Delta \tau}{\Delta \chi^2} [v(x + 1, \tau) + v(x - 1, \tau) - 2v(x, \tau)] + \Delta \tau Reac[v(i, \tau + 1)] \quad (5)$$

que finalmente queda como:

$$v(i, \tau + 1) = v(i, \tau) + r_v Dif[v(i, \tau + 1)] + \eta Reac[v(i, \tau + 1)] \quad (6)$$

donde hemos reemplazado x por i y además hemos definido:

$$r_v = \frac{D_v \Delta \tau}{\Delta \chi^2} \quad y \quad \eta = \Delta \tau \quad (7)$$

2. A qué equivale tomar $\eta = 1$?

Como se ha descrito en el cuerpo del trabajo previo éste parámetro “controla” la escala temporal. Tomarlo como 1 equivale a “conservar” la escala temporal entre los términos de difusión y reacción de las ecuaciones.

3. Obtención de las ecuaciones para el caso de un autómata bidimensional cuadrado en el caso de la vecindad de Von Neumann ($r=1$)

Las componentes de reacción en las ecuaciones no cambiarán independientemente de la dimensión de nuestro autómata (obviando naturalmente que cada término vendrá definido por los valores de tantas coordenadas como dimensiones tenga nuestro autómata). Vamos a centrarnos por tanto en el término de difusión. Este término es función del laplaciano, que consiste en la suma de las segundas derivadas de la función con respecto a cada una de las variables (dimensión). Por tanto es necesario recalcular esta contribución para el caso de 2 dimensiones.

Asumiendo una vecindad de Von Neumann con $r=1$ los 2 términos del laplaciano en 2 dimensiones serán:

$$\frac{\partial^2 v}{\partial x^2} \approx \frac{v(x - \Delta x, y, \tau) + v(x + \Delta x, y, \tau) - 2v(x, y, \tau)}{\Delta x^2} \quad (8)$$

$$\frac{\partial^2 v}{\partial y^2} \approx \frac{v(x, y - \Delta y, \tau) + v(x, y + \Delta y, \tau) - 2v(x, y, \tau)}{\Delta y^2} \quad (9)$$

A continuación asumimos $\Delta x = \Delta y = \Delta \chi$ y sumando ambos términos obtenemos la siguiente ecuación:

$$\frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial x^2} \approx \frac{v(x - \Delta x, y, \tau) + v(x + \Delta x, y, \tau) + v(x, y - \Delta y, \tau) + v(x, y + \Delta y, \tau) - 4v(x, y, \tau)}{\Delta \chi^2} \quad (10)$$

Ahora usando la misma notación que en el caso unidimensional para nuestro autómata celular:

$$v(x - \Delta x, y, \tau) = v(i - 1, j, \tau) \quad (11)$$

$$v(x + \Delta x, y, \tau) = v(i + 1, j, \tau) \quad (12)$$

$$v(x, y - \Delta y, \tau) = v(i, j - 1, \tau) \quad (13)$$

$$v(x, y + \Delta y, \tau) = v(i, j + 1, \tau) \quad (14)$$

Y finalmente obtendremos para el término de difusión:

$$Diff[v(i, j, \tau + 1)] = v(i - 1, j, \tau) + v(i + 1, j, \tau) + v(i, j - 1, \tau) + v(i, j + 1, \tau) - 4v(i, j, \tau) \quad (15)$$

El término de difusión para w es idéntico:

$$Diff[w(i, j, \tau + 1)] = w(i - 1, j, \tau) + w(i + 1, j, \tau) + w(i, j - 1, \tau) + w(i, j + 1, \tau) - 4w(i, j, \tau) \quad (16)$$

Como hemos discutido antes la aproximación para los términos de reacción no cambia con respecto al número de dimensiones pues hablamos de un sistema autónomo. Es por tanto la contribución del laplaciano lo que únicamente hemos de cambiar.

4. Modelado del caso excitable y simulación

Se han simulado los diferentes casos (excitable, oscilatorio y biestable). Los resultados pueden observarse en el cuerpo del artículo.

5. Códigos

A continuación se muestran algunos de los códigos clave usados en las simulaciones. Se ha usado el lenguaje de programación Python y el entorno de Jupyter Notebook en la realización de este trabajo:

Función que realiza los cálculos.

Se le pasan como parámetros la configuración del sistema así como el radio de la vecindad perturbada inicialmente y el número de iteraciones a realizar:

```
def FN_difusion(n=200, a=0.2, b=3, Epsilon=0.01, r_v=0.4, r_w=0.01,
               eta=0.005, r=10, iter=20000 ):
    "Los parametros son"

    numero_eventos = 0
    # Cálculo del número de iteraciones que dejaremos entre una grabación del
    # estado del sistema y el siguiente. La idea es que podamos cambiar el
    # número de iteraciones y en cualquier caso nos quedemos con el estado
    # a intervalos regulares con un máximo de 500 estados grabados.
    if (iter/500)<1:
        delta_registro = iter
    else:
        delta_registro = int(iter/500)

    # Creación y planteamiento de los vectores que contienen las células
    # En lugar de crear 2 vectores diferenciados voy a utilizar un "structured
    # array" de numpy. El código resulta más limpio
    Z = np.zeros((n+2), [('V', np.double), ('W', np.double)])
    V,W = Z['V'], Z['W']
    v,w = V[1:-1], W[1:-1]

    # r es el número de celdas alrededor de la central a las aplicamos una ex-
    # citacion (finalmente serán 10)

    #r = 10 --> No es necesario puesto que viene como parámetro de la función

    v[...] = 0.0
    V[int(n/2)-r:int(n/2)+r] = -0.2
```

```

W[int(n/2)-r:int(n/2)+r] = -0.3
v += 0.01*np.random.random(n)
w += 0.01*np.random.random(n)

v_graf = np.zeros((1,n))

for i in range(iter):
    Dif_v = (V[:-2] - 2*V[1:-1] + V[2:])
    Dif_w = (W[:-2] - 2*W[1:-1] + W[2:])

    Reac_v = v * (v-a) * (1-v) - w
    Reac_w = Epsilon * (v - b * w)

    v += r_v * Dif_v + eta * Reac_v
    w += r_w * Dif_w + eta * Reac_w

    V[0] = V[-2]
    V[-1] = V[1]
    W[0] = W[-2]
    W[-1] = W[1]

    if i%delta_registro == 0:
        v_graf = np.append(v_graf, v)
        numero_eventos+= 1

## Dibujamos los resultados para v
#plt.plot(v)

matriz = v_graf.reshape((numero_eventos+1, 200))
plt.pcolor(matriz)
plt.colorbar()
plt.xlabel("# Celda")
plt.ylabel(r'$40 \cdot \Delta \tau$')

return()

```

Función que realiza los cálculos en 2 dimensiones:

Se le pasan como parámetros la configuración del sistema, el radio de la vecindad perturbada inicialmente, el número de iteraciones a realizar y los valores iniciales de las celdas perturbadas y el resto:

```

def FN_difusion_inic(n=100, a=0.2, b=3, Epsilon=0.01, r_v=0.4, r_w=0.01,
                    eta=0.005, r=10, itera=1, v_pert=-0.2, w_pert=-0.3, v_inic=0.0, w_inic=0.0 ):
    "Los parametros son"

    numero_eventos = 0
    # Cálculo del número de iteraciones que dejaremos entre una grabación del
    # estado del sistema y el siguiente. La idea es que podamos cambiar el
    # número de iteraciones y en cualquier caso nos quedemos con el estado

```

```

# a intervalos regulares con un máximo de 500 estados grabados.
if (itera/500)<1:
    delta_registro = 1
else:
    delta_registro = int(itera/500)

# Creación y planteamiento de los vectores que contienen las celulas
# En lugar de crear 2 vectores diferenciados voy a utilizar un "structured
# array" de numpy. El código resulta más limpio
Z = np.zeros((n+2, n+2), [('V', np.double), ('W', np.double)])
V,W = Z['V'], Z['W']
v,w = V[1:-1,1:-1], W[1:-1,1:-1]

# r es el número de celdas alrededor de la central a las aplicamos una ex-
#citacion (finalmente serán 10)

#r = 10 --> No es necesario puesto que viene como parámetro de la función
V[...] = v_inic
W[...] = w_inic

#v[...] = v_inic

V[int(n/2)-r:int(n/2)+r,int(n/2)-r:int(n/2)+r] = v_pert
W[int(n/2)-r:int(n/2)+r,int(n/2)-r:int(n/2)+r] = w_pert
v += 0.01*np.random.random((n,n))
w += 0.01*np.random.random((n,n))

v_graf = np.zeros((0,n))

for i in range(itera):
    Dif_v = (
        V[0:-2,1:-1] +
        V[1:-1,0:-2] - 4*V[1:-1,1:-1] + V[1:-1,2:] +
        V[2: ,1:-1] )

    Dif_w = (
        W[0:-2,1:-1] +
        W[1:-1,0:-2] - 4*W[1:-1,1:-1] + W[1:-1,2:] +
        W[2: ,1:-1] )

    Reac_v = v * (v-a) * (1-v) - w
    Reac_w = Epsilon * (v - b * w)

    v += r_v * Dif_v + eta * Reac_v
    w += r_w * Dif_w + eta * Reac_w

V[:, 0] = V[:, -2]
V[:, -1] = V[:, 1]
V[0,2:] = V[-2, 2:]
V[-1, 2:] = V[1, 2:]

```

```

W[:, 0] = W[:, -2]
W[:, -1] = W[:, 1]
W[0, 2:] = W[-2, 2:]
W[-1, 2:] = W[1, 2:]

# if i%delta_registro == 0:
#     v_graf = np.append(v_graf, v)
#     numero_eventos+= 1

## Dibujamos los resultados para v
#plt.plot(v)
#matriz = v_graf.reshape((n*(numero_eventos), n))
#plt.pcolor(matriz)
plt.pcolor(v, vmin=-0.2, vmax=1.4)
plt.colorbar()
plt.xlabel("# X-Celda")
plt.ylabel("# Y-Celda")

return()

```