INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE COMPUTO


2024

# SESIÓN DE LABORATORIO 8: SIMULACIÓN DE LA OLA MEXICANA

## ALGORITMOS BIOINSPIRADOS

**Elaborado por:** Flores Estopier Rodrigo

**Boleta**: 2021630260

**Profesor:** Rosas Trigueros Jorge Luis

**Grupo:** 6CV2

**Fecha de realización:** 9/11/2024

**Fecha de entrega:** 12/11/2024

**Semestre:** 25-1

# Contenido

# 1. MARCO TEÓRICO

La simulación de la ola mexicana en una matriz puede modelarse como un sistema bioinspirado que imita la propagación de estados de activación a través de individuos organizados en una cuadrícula. Este fenómeno puede observarse en eventos de público masivo, donde los espectadores se levantan y sientan en sucesión para crear un efecto de "ola" que se propaga. En términos de modelado computacional, los sistemas de activación como las autómatas celulares se utilizan para replicar esta dinámica. En esta práctica, el comportamiento de activación y refracción de los individuos se gestiona mediante una serie de parámetros, tales como el umbral de activación, el peso direccional y el radio de influencia, permitiendo que las celdas en estado inactivo pasen a activas bajo determinadas condiciones y, eventualmente, a un estado refractario en el que permanecen durante un número limitado de iteraciones

# 2. DESARROLLO

## CODIFICACIÓN

Se utilizaron las siguientes librerías para la realización del algoritmo.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib.colors import ListedColormap
```

- numpy: Se utilizó para realizar una copia de la matriz sobre la cual actualizaremos los estados de cada individuo.
- Pyplot: Se utilizo para mostrar de manera grafica los estados de los individuos.
- Animation: Se utilizo para mostrar de forma animada el cambio de los estados a través del tiempo.
- Colors: se utilizo para establecer una lista de colores, los cuales nos ayudaran a a representar los estados de cada individuo y sus estados refractarios

**Parámetros**

Se consideraron los siguientes parámetros para la realización de la simulación:

- num_rows :Representa el valor de filas que tendrá la matriz.

- num_cols : Representa el valor de columnas que tendrá la matriz.
- nr: Representa el numero de pasos durante el cual un individuo se mantiene en estado refractario antes de pasar a estado inactivo.
- c: Representa el valor de umbral promedio que tiene que superar un individuo para sea correctamente influenciado y pase a estado activo.
- delta_c : Representa el valor de variación de umbral.
- R: Representa el radio máximo de individuos que pueden influenciar a uno.
- w0 : Representa un peso direccional, donde un valor mas cercano a 0 representa una mayor influencia hacia la derecha.

```
# Parámetros de la simulación
num_rows = 20  # Tamaño de la cuadrícula
num_cols = 80
nr = 3         # Número de pasos que una célula permanece refractaria
c = 0.1        # Umbral promedio (reducido)
delta_c = 0.05 # Variación del umbral
R = 2          # Radio de influencia
w0 = 0.1       # Peso direccional fuerte hacia la derecha
```

**Matriz**

Posteriormente, se inicializa una matriz que representa a todos los individuos, inicialmente en un estado 0.

Después, establecemos el valor de algunos individuos a activos, esto representa nuestro impulso inicial.

```
# Estados de las células: 0 = inactiva, 1 = activa, 2 a nr+1 = refractaria
grid = np.zeros((num_rows, num_cols), dtype=int)

# Iniciar con un punto activado
grid[5, 0] = 1
grid[6, 0] = 1
grid[7, 0] = 1
```

**Funcion de actualización de estados**

La funcion que realiza el trabajo de actualización de los estados recibe la matriz actual y crea una copia de esta.

```
# Función para actualizar el estado de la cuadrícula
def update_grid(grid):
    new_grid = np.copy(grid)
```

Se itera para cada celda (individuo) de la matriz, donde estos se pueden encontrar en 4 situaciones diferentes:

- Individuo en estado 1: Si un individuo se encuentra en estado activo (1) se cambia su valor a 2 para indicar que este pasara a un estado refractario.
- Individuo en estado entre 2 y el valor de 2 más el parámetro de nr: Si un individuo se encuentra en este estado quiere decir que está en estado refractario, y solo se aumentara su valor en 1 para que avance hasta un estado inactivo.
- Individuo con un valor mayor a 2 mas nr: Un individuo con este valor indica que ya ha llegado al ultimo paso de estado refractario, por lo tanto, se establecerá con un estado desactivado (0).

```
    for i in range(num_rows):
        for j in range(num_cols):
            if grid[i, j] == 1:
                new_grid[i, j] = 2  # Pasar a estado refractario
            elif 2 <= grid[i, j] < 2 + nr:
                new_grid[i, j] += 1  # Avanzar en el estado refractario
            elif grid[i, j] >= 2 + nr:
                new_grid[i, j] = 0
```

- Individuo con estado 0: Si un individuo esta desactivado, se realizara un cálculo probabilístico para determinar si este será activado o no.
    - activation_sum: Acumula el nivel de activación de las celdas vecinas.
    - num_neighbors: Cuenta el número de celdas vecinas consideradas.
    - Se recorren todas las celdas en un cuadrado de lado 2R+1 centrado en la celda actual (i, j).
    - ni, nj: Son los índices de la celda vecina.
    - La condición 0 <= ni < num_rows and 0 <= nj < num_cols asegura que la celda vecina está dentro de los límites de la cuadrícula.

- La condición (di != 0 or dj != 0) asegura que la celda vecina no es la celda actual.
- weight = w0 if dj > 0 else (1 - w0): Asigna un peso a la celda vecina, dando preferencia a las celdas a la derecha (dj > 0).
- activation_sum += weight * (grid[ni, nj] == 1): Incrementa activation_sum si la celda vecina está activa (grid[ni, nj] == 1), ponderada por weight.
- num_neighbors += 1: Incrementa el contador de celdas vecinas consideradas.

```
            elif grid[i, j] == 0:
                # Calcular el nivel de activación desde las células vecinas
                activation_sum = 0
                num_neighbors = 0
                for di in range(-R, R + 1):
                    for dj in range(-R, R + 1):
                        ni, nj = i + di, j + dj
                        if 0 <= ni < num_rows and 0 <= nj < num_cols and (di != 0
or dj != 0):
                            weight = w0 if dj > 0 else (1 - w0)  # Preferencia
hacia la derecha
                            activation_sum += weight * (grid[ni, nj] == 1)
                            num_neighbors += 1
```

Calcula la activación promedio de las celdas vecinas. Si no hay celdas vecinas (num_neighbors == 0), la activación promedio es 0.
Calculamos un umbral de activación aleatorio basado en c y delta_c. np.random.rand() genera un número aleatorio entre 0 y 1, por lo que (np.random.rand() - 0.5) genera un número entre -0.5 y 0.5.

```
                avg_activation = activation_sum / num_neighbors if num_neighbors
> 0 else 0
                threshold = c + delta_c * (np.random.rand() - 0.5)
```

Si la activación promedio de las celdas vecinas es mayor que el umbral calculado, la celda actual se activa en la nueva cuadrícula

```
                if avg_activation > threshold:
                    new_grid[i, j] = 1  # Activar célula
    return new_grid
```

**Muestra de la animación**

Creamos una figura (fig) y un eje (ax) utilizando matplotlib.pyplot.subplots(). Esto prepara el espacio donde se mostrará la animación.

```
# Configurar la animación
fig, ax = plt.subplots()
```

- Muestra la cuadrícula inicial (grid) en el eje (ax) utilizando imshow().
    - cmap: Especifica el mapa de colores a utilizar, definido previamente.
    - interpolation='nearest': Ajusta la interpolación para que los valores de la cuadrícula se muestren como bloques de color sin suavizado.
    - vmin=0, vmax=nr + 1: Establece los límites del rango de valores para el mapa de colores.

```
im = ax.imshow(grid, cmap=cmap, interpolation='nearest', vmin=0, vmax=nr + 1)
```

- Definimos la función animate(frame) que se llama en cada fotograma de la animación.
    - global grid: Indica que la función modificará la variable global grid.
    - grid = update_grid(grid): Actualiza la cuadrícula llamando a la función update_grid(grid), explicada anteriormente.
    - im.set_array(grid): Actualiza la imagen mostrada con la nueva cuadrícula.
    - return [im]: Devuelve una lista con el objeto de imagen actualizado, lo que es necesario para la animación con blit=True.

```
def animate(frame):
    global grid
    grid = update_grid(grid)
    im.set_array(grid)
    return [im]
```

- Crea una animación utilizando animation.FuncAnimation().
    - fig: La figura en la que se mostrará la animación.
    - animate: La función que se llama en cada fotograma para actualizar la animación.
    - frames=300: El número de fotogramas en la animación.
    - interval=50: El intervalo entre fotogramas en milisegundos (50 ms).
    - blit=True: Optimiza la animación actualizando solo las partes de la figura que han cambiado.

```
ani = animation.FuncAnimation(fig, animate, frames=300, interval=50, blit=True)

plt.show()
```

## EJECUCIÓN

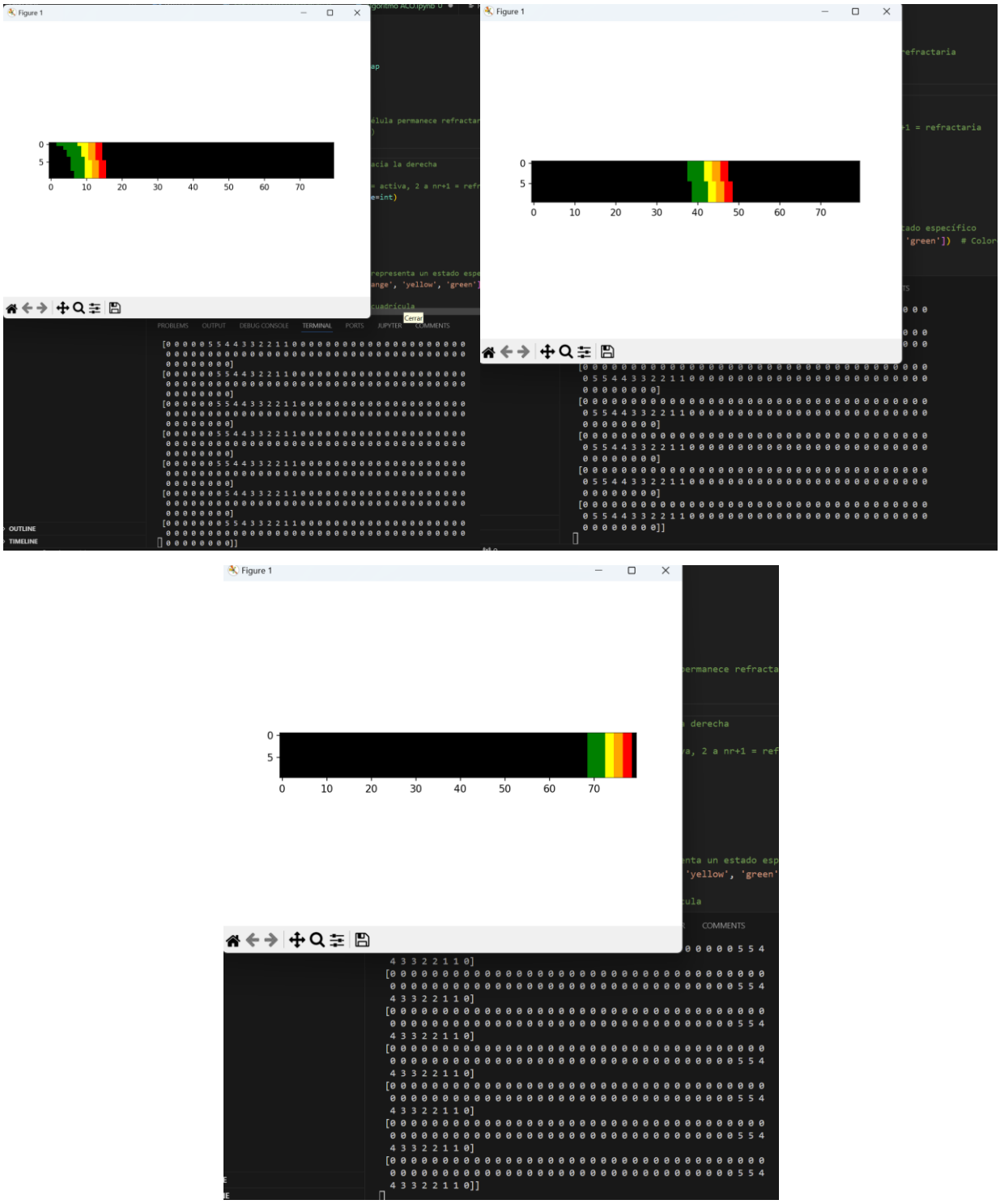


*Ilustración 1. Ejecución de la simulación.*

## 3. CONCLUSIONES

Durante la práctica, se observó que ajustar adecuadamente el peso direccional y el umbral de activación es crucial para obtener una onda consistente y continua. Se complico un poco lograr una configuración equilibrada que permitiera una propagación natural sin saturar la cuadrícula, lo cual se alcanzó con ajustes en el umbral y el radio de influencia. Un aprendizaje importante derivado de la experimentación fue la influencia del número de pasos en el estado refractario, ya que afecta la frecuencia y la duración de la ola. Para futuras implementaciones, sería útil incluir variaciones espaciales en los umbrales y experimentar con radios de influencia adaptativos para observar los cambios en la propagación de la onda.

## 4. BIBLIOGRAFÍA

[1] I. Farkas, D. Helbing, y T. Vicsek, “Mexican waves in an excitable medium”, *Nature*, vol. 419, núm. 6903, pp. 131–132, 2002.

[2] G. Improve, “Matplotlib.Animation.FuncAnimation class in python”, *GeeksforGeeks*, 05-abr-2020. [En línea]. Disponible en: https://www.geeksforgeeks.org/matplotlib-animation-funcanimation-class-in-python/. [Consultado: 10-nov-2024].