Proyecto Física Computacional:

Dinámica molecular en dos dimensiones basado en el paper: “Traffic jams without bottlenecks–experimental evidence for the physical mechanism of the formation of a jam”

Gabriel Contreras Abellán
C12251

Resumen

Desde el punto de vista físico, un embotellamiento es un sistema de particulas que interactuan fuera del equilibrio. El efecto colectivo del sistema de múltiples partículas induce la inestabilidad de un estado de flujo libre causado por la intensificación de las fluctuaciones, y la transición a un estado de congestión ocurre de manera espontánea si la densidad promedio de vehículos supera un cierto valor crítico. Por lo tanto, un cuello de botella es solo un desencadenante y no el origen esencial de un atasco. En este artículo, presentamos la primera evidencia experimental de que la aparición de un atasco es un fenómeno colectivo similar a las transiciones de fase 'dinámicas' y a la formación de patrones.

Índice

Introducción

Simulación de la Transición de Fase en el Tráfico Vehicular.
Este proyecto de física computacional se dedica a simular y analizar la formación espontánea de atascos de tráfico, buscando demostrar que la congestión es una manifestación de una transición de fase colectiva, más que el simple resultado de un obstáculo físico.

El experimento se desarrolla en un entorno simulado de una carretera circular de 230 metros de circunferencia, con 20 vehículos (modelados como partículas). La condición inicial es un movimiento homogéneo con velocidad uniforme, ajustando la densidad promedio de vehículos a un valor crítico que prefigure el inicio de la inestabilidad.

Problema

¿Cómo y por qué un flujo vehicular inicialmente homogéneo y estable experimenta una "transición de fase" a un estado congestionado, impulsado únicamente por el movimiento colectivo y las interacciones entre vehículos cuando la densidad excede un valor umbral, imitando la formación de una onda de choque (presa fantasma) sin la intervención de una causa externa?

Objetivos

- Elaborar subrutinas que evalúan si las partículas cambian de velocidad. Calcular las nuevas velocidades y posiciones del sistema y como afecta el flujo de las partículas.

- Visualizar por medio de matplotlib la posición de cada una de las partículas dentro de los carriles y guardar el correspondiente archivo para realizar una película al final del cálculo.

- Visualizar el efecto de presa fantasma en una carretera circular.

- Crear documentación.

Metodología

Usar Paradigma de Programación Orientado a Objetos (POO)

Es un modelo de programación que organiza el diseño de software alrededor de datos, u objetos, en lugar de funciones y lógica. Se centra en crear código modular y reutilizable.

Metodología

Modelo de Seguimiento de Vehículos (IDM)

Calcula la aceleración de un vehículo basándose en su velocidad actual, la distancia de separación (gap) con el vehículo precedente y la diferencia de velocidad entre ambos. Su objetivo es mantener una distancia de seguridad deseada (s*) y alcanzar una velocidad objetivo de manera confortable, sirviendo como una herramienta clave para simular la dinámica de tráfico

La aceleración a_i está definida por:

$$a_i = A_{\max}\left[1 - \left(\frac{v_i}{V_0}\right)^4 - \left(\frac{s^*}{g_i}\right)^2\right]$$

Donde s^* es la distancia de seguridad deseada, calculada como:

$$s^* = S_0 + \text{máx}\left[0, \quad v_i T_{\text{headway}} + \frac{v_i(v_i - v_{i-1})}{2\sqrt{A_{\max} B_{\text{decel}}}}\right]$$

Implementación

```python
import numpy as np
import matplotlib.pyplot as plt
import math

class TrafficSimulation:
    def __init__(self):
        # Parámetros Globales
        self.N = 20
        self.CIRC = 230.0
        self.DT = 0.05
        self.SIM_TIME = 1200.0

        # Parámetros Físicos
        self.L_VEHICLE = 4.0
        self.S0 = 2.0

        # Modelo IDM
        self.V0 = 15.0
        self.A_MAX = 0.8
        self.B_DECEL = 4.0
        self.T_HEADWAY = 1.6

        # Eventos de parada
        self.STOP_DURATION = 6.0 # Duración del frenazo (suficiente para parar la fila)
        self.FIRST_STOP = 2.0
        self.REPEAT_INTERVAL = 15.0
```

Implementación

```
27
28          # Estado inicial
29          self.s = np.linspace(0, self.CIRC, self.N, endpoint=False)
30          self.v = np.ones(self.N) * self.V0
31
32          # Visualización
33          self.fig, self.ax = None, None
34
35      #  FUNCIONES DEL MODELO
36
37      def get_gap(self, i, s_curr):
38          """Distancia al vehículo de adelante."""
39          leader_idx = (i - 1 + self.N) % self.N
40          dist = s_curr[leader_idx] - s_curr[i]
41          if dist < 0:
42              dist += self.CIRC
43          return dist - self.L_VEHICLE
44
45      def idm_accel(self, v_curr, v_leader, gap):
46          """Aceleración IDM."""
47          delta_v = v_curr - v_leader
48
49          s_star = self.S0 + max(
50              0.0,
51              v_curr * self.T_HEADWAY + (v_curr * delta_v) / (2 * math.sqrt(self.A_MAX * self.B_DECEL))
52          )
```

Implementación

```
53
54          effective_gap = max(0.01, gap)
55
56          accel = self.A_MAX * (1 - (v_curr / self.V0)**4 - (s_star / effective_gap)**2)
57          return accel
58
59      def leader_stopped(self, t):
60          """Determina si el líder debe detenerse."""
61          time_since_start = t - self.FIRST_STOP
62          if time_since_start >= 0:
63              cycle_pos = time_since_start % self.REPEAT_INTERVAL
64              return cycle_pos < self.STOP_DURATION
65          return False
66
67      def run_step(self, t):
68          s_new = np.copy(self.s)
69          v_new = np.copy(self.v)
70
71          accel = np.zeros(self.N)
72          p0_is_stopped = self.leader_stopped(t)
```

Implementación

```
73
74          #  Cálculo de aceleraciones
75          for i in range(self.N):
76              if i == 0:
77                  # LÍDER
78                  if p0_is_stopped:
79                      if self.v[i] > 0:
80                          accel[i] = -10.0
81                      else:
82                          accel[i] = 0.0
83                          v_new[i] = 0.0
84                  else:
85                      accel[i] = self.A_MAX * (1 - (self.v[i] / self.V0)**4)
86              else:
87                  leader_idx = (i - 1 + self.N) % self.N
88                  gap = self.get_gap(i, self.s)
89                  leader_v = self.v[leader_idx]
90                  accel[i] = self.idm_accel(self.v[i], leader_v, gap)
```

Implementación

```
91
92          #  Integración (Euler)
93          for i in range(self.N):
94              if i == 0 and p0_is_stopped and v_new[i] == 0:
95                  continue
96
97              v_new[i] = max(0.0, self.v[i] + accel[i] * self.DT)
98              s_new[i] = (self.s[i] + v_new[i] * self.DT) % self.CIRC
99
100             if i != 0:
101                 gap = self.get_gap(i, self.s)
102                 if (v_new[i] * self.DT) > (gap - 0.5):
103                     v_new[i] = max(0.0, (gap - 0.5) / self.DT)
104                     s_new[i] = (self.s[i] + v_new[i] * self.DT) % self.CIRC
105
106         self.s = s_new
107         self.v = v_new
```

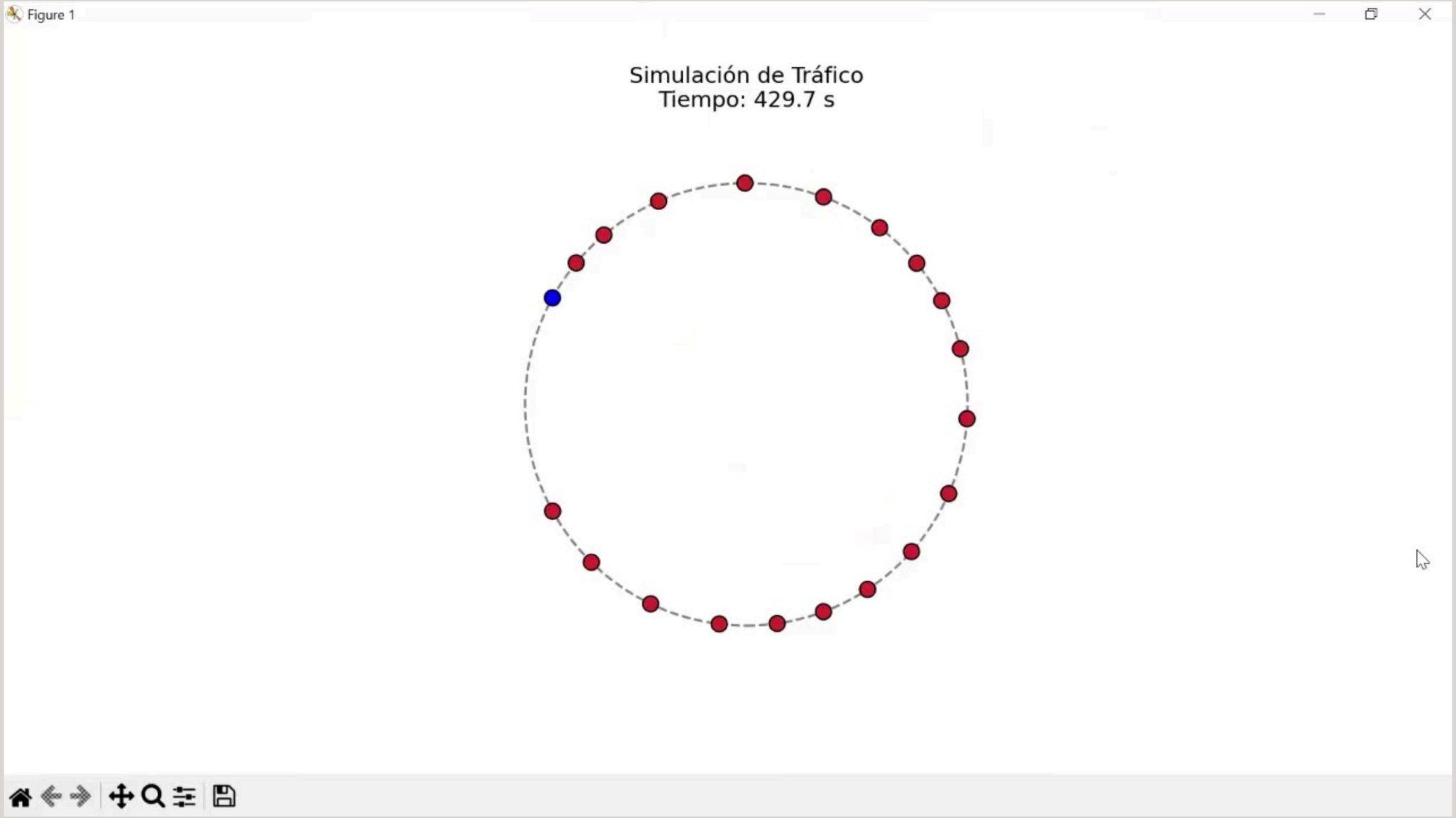
Implementación

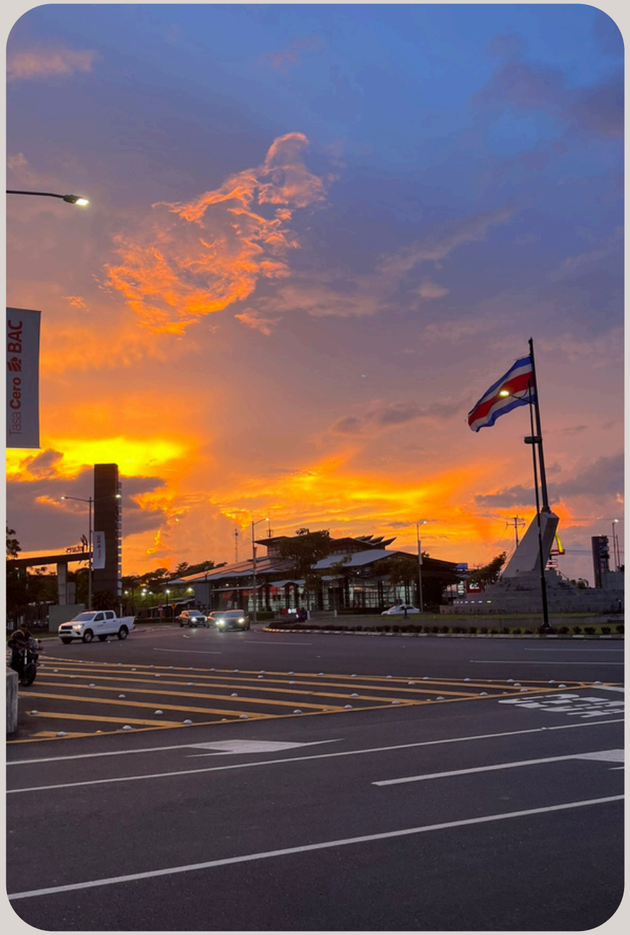
```
108
109     #  VISUALIZACIÓN
110     def setup_draw(self):
111         plt.ion()
112         self.fig, self.ax = plt.subplots(figsize=(7, 7))
113
114     def draw(self, t):
115         R = self.CIRC / (2 * np.pi)
116         self.ax.clear()
117         self.ax.set_box_aspect(1)
118         self.ax.set_xlim(-R * 1.3, R * 1.3)
119         self.ax.set_ylim(-R * 1.3, R * 1.3)
120         self.ax.axis('off')
121
122         circle = plt.Circle((0, 0), R, color='gray', fill=False, linestyle='--', linewidth=1.5)
123         self.ax.add_artist(circle)
124
125         angles = self.s / R
126         X = R * np.cos(angles)
127         Y = R * np.sin(angles)
128
129         colors = ['blue' if i == 0 else '#d62728' for i in range(self.N)]
130         self.ax.scatter(X, Y, s=100, c=colors, edgecolors='black', zorder=10)
131
132         self.ax.set_title(f"Simulación de Tráfico\nTiempo: {t:.1f} s", fontsize=14)
133         plt.pause(0.001)
```

Implementación

```
134
135     #  LOOP PRINCIPAL
136     def run(self):
137         self.setup_draw()
138
139         t = 0.0
140         print("Iniciando simulación...")
141         print(f"P0 frena a los {self.FIRST_STOP}s y cada {self.REPEAT_INTERVAL}s.")
142
143         while t < self.SIM_TIME:
144             self.run_step(t)
145
146             if int(t / self.DT) % 2 == 0:
147                 self.draw(t)
148
149             t += self.DT
150
151         plt.ioff()
152         plt.show()
153
154 #  MAIN
155 if __name__ == "__main__":
156     sim = TrafficSimulation()
157     sim.run()
```

Resultados





Conclusión

El flujo vehicular se comporta como un sistema dinámico fuera del equilibrio termodinámico. Esto implica que la formación de la congestión es un fenómeno de auto-organización que reside en la sensibilidad extrema a las condiciones iniciales.

En esencia, la congestión es el resultado de la interacción colectiva y las reglas de seguimiento, confirmando que la inestestabilidad es una propiedad inherente del sistema cuando opera en condiciones de alta densidad.

¡Gracias!