INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

BIOINFORMATICS

PRÁCTICA 4

INTRODUCCIÓN A DINÁMICA MOLECULAR

NOMBRE DEL ALUMNO: GARCÍA QUIROZ GUSTAVO IVAN

GRUPO: 7CV3

NOMBRE DEL PROFESOR: ROSAS TRIGUEROS JORGE LUIS

FECHA DE REALIZACIÓN DE LA PRÁCTICA: 12/02/2025

FECHA DE ENTREGA DEL REPORTE: 19/03/2025

Índice

1 Marco teórico.

1.1 Dinámica molecular

La dinámica molecular (MD) es una técnica computacional que permite simular el comportamiento físico de átomos y moléculas a lo largo del tiempo. Esta metodología se basa en la aplicación de las leyes de la física clásica, principalmente la mecánica newtoniana, para modelar las interacciones entre partículas y predecir su evolución temporal.

La dinámica molecular se fundamenta en resolver numéricamente las ecuaciones de movimiento de Newton para un sistema de partículas que interaccionan entre sí. Para cada átomo i, se aplica la ecuación:

$$F_i = m_i \cdot a_i$$

Donde F_i es la fuerza neta que actúa sobre el átomo, m_i es su masa y a_i es su aceleración. La fuerza neta se calcula como el gradiente negativo de la energía potencial del sistema:

$$F_i = -\nabla V_i$$

Las fuerzas entre átomos se modelan mediante diferentes potenciales que representan las interacciones físicas y químicas relevantes.

Principales interacciones en sistemas moleculares

En el caso de una simulación de moléculas de agua, las principales interacciones incluyen:

- **Enlaces covalentes**: Modelados típicamente como resortes mediante la ley de Hooke, donde la fuerza es proporcional a la desviación respecto a la longitud de equilibrio del enlace. $F_{enlace} = -k(r - r_{eq})$ Donde k es la constante de fuerza del enlace, r es la distancia actual entre átomos y r_eq es la distancia de equilibrio.
- **Interacciones electrostáticas**: Debido a las cargas parciales en los átomos, se aplica la ley de Coulomb: $F_{coulomb} = \frac{k_e \cdot q_1 \cdot q_2}{r^2}$ Donde k_e es la constante electrostática, $q_1 \, y \, q_2$ son las cargas de los átomos, y r es la distancia entre ellos.
- **Interacciones de van der Waals**: Para modelar la repulsión a cortas distancias y la atracción débil a distancias mayores, generalmente representadas por potenciales como el de Lennard-Jones.

1.2 La molécula de agua

El agua (H_2O) es una molécula compuesta por un átomo de oxígeno y dos átomos de hidrógeno. Sus características principales son:

- Geometría angular con un ángulo H-O-H de aproximadamente 104.5°
- Enlaces covalentes O-H con una longitud de equilibrio de aproximadamente 0.96 Å
- Distribución asimétrica de carga: el oxígeno tiene una carga parcial negativa, mientras que los hidrógenos tienen cargas parciales positivas
- Esta distribución de carga hace que las moléculas de agua interactúen entre sí mediante puentes de hidrógeno

2 Material y equipo.

Para esta práctica se usaron las siguientes herramientas de software y hardware necesarias para realizar la práctica.

2.1.1 Hardware

- Computadora.

2.1.2 Software

- Google colaboratory.

3 Desarrollo de la práctica.

En esta práctica, la implementación de esta práctica se centró en el desarrollo de un modelo de dinámica molecular para simular el comportamiento de moléculas de agua en un entorno bidimensional. Se abordaron los siguientes aspectos clave:

3.1 Configuración del entorno de simulación

Inicialmente, se estableció un espacio de simulación de 600×600 píxeles con un sistema de coordenadas centrado en el medio de la pantalla. Se definieron constantes físicas fundamentales para el modelo:

```
# Canvas setup
maxX = 600
maxY = 600
x0 = int(maxX/2)
y0 = int(maxY/2)

# Colors
oxygen_color = (0, 0, 255)   # Red for oxygen
hydrogen_color = (255, 255, 255)  # White for hydrogen

# Physical constants
delta_t = 0.05
m_oxygen = 16.0
m_hydrogen = 1.0
k_bond = 2.0         # Spring constant for O-H bond
d_OH = 30.0          # Equilibrium O-H bond length
angle_HOH = 104.5 * math.pi / 180  # H-O-H bond angle in radians
q_oxygen = -2.0      # Partial charge on oxygen
q_hydrogen = 1.0     # Partial charge on hydrogen
coulomb_factor = 200.0  # Scaling factor for electrostatic forces
repulsion_factor = 5000.0  # Repulsion factor between molecules
```

Figura 1 Configuración del entorno de simulación

3.2 Creación de las clases básicas

Se desarrollaron dos clases principales:

La clase Atom representa cada átomo individual con propiedades como posición, velocidad, aceleración, masa y carga. Implementa métodos para dibujar el átomo y actualizar su posición según las fuerzas aplicadas.

La clase WaterMolecule representa una molécula de agua completa con un átomo de oxígeno y dos de hidrógeno. Mantiene las relaciones espaciales entre los átomos y gestiona las interacciones intramoleculares.

3.3 Modelado de interacciones físicas

Se implementaron tres tipos de interacciones fundamentales:

1. **Interacciones intramoleculares**:
    - Fuerzas de enlace entre O-H modeladas como resortes (ley de Hooke)

- Fuerzas electrostáticas entre átomos de la misma molécula
- Restricciones angulares para mantener la geometría del agua

2. **Interacciones intermoleculares**:
   - Fuerzas electrostáticas entre átomos de diferentes moléculas
   - Fuerzas de repulsión para evitar superposiciones
3. **Condiciones de contorno**:
   - Implementación de condiciones periódicas para simular un sistema infinito

3.4 Algoritmo de integración numérica

Se desarrolló un algoritmo de integración basado en el método de Verlet de velocidad, que actualiza la posición y velocidad de cada átomo en pasos pequeños de tiempo

```python
def update_position(self):
    # Update acceleration
    self.ax = self.fx / self.mass
    self.ay = self.fy / self.mass

    # Update velocity
    self.vx += self.ax * delta_t
    self.vy += self.ay * delta_t

    # Update position
    self.x += self.vx * delta_t + 0.5 * self.ax * (delta_t**2)
    self.y += self.vy * delta_t + 0.5 * self.ay * (delta_t**2)

    # Boundary conditions (wrap around)
    if self.x > maxX/2:
        self.x = -maxX/2
    elif self.x < -maxX/2:
        self.x = maxX/2

    if self.y > maxY/2:
        self.y = -maxY/2
    elif self.y < -maxY/2:
        self.y = maxY/2
```

Figura 2 Algoritmo basado en el método de Verlet de velocidad

3.5 Sistema de visualización

Se implementó un sistema de visualización usando OpenCV y IPython widgets para mostrar la simulación en tiempo real:

- Representación visual de los átomos con diferentes colores y tamaños
- Visualización de enlaces entre átomos
- Actualización dinámica de la visualización en cada paso de tiempo

3.6 Inicialización del sistema molecular

Se crearon cuatro moléculas de agua con posiciones y orientaciones iniciales aleatorias:

```
# Create water molecules at random positions
num_molecules = 4
water_molecules = []
for i in range(num_molecules):
    x = np.random.uniform(-maxX/4, maxX/4)
    y = np.random.uniform(-maxY/4, maxY/4)
    angle = np.random.uniform(0, 2*math.pi)
    water_molecules.append(WaterMolecule(x, y, angle))
```

Figura 3 Inicialización del sistema molecular

3.7 Implementación del bucle principal de simulación

Finalmente, se desarrolló el bucle principal que ejecuta la simulación durante 2000 iteraciones:

```
# Main simulation loop
MaxIterations = 2000
for count in range(MaxIterations):
    # Clear the image
    img[:] = (0, 0, 0)

    # Calculate intermolecular forces
    calculate_molecular_interactions(water_molecules)

    # Update each molecule
    for molecule in water_molecules:
        molecule.update()
        molecule.draw(img)
```

Figura 4 Implementación del bucle principal de simulación

3.8 Desafíos y soluciones

Uno de los principales desafíos fue la implementación adecuada de las interacciones físicas entre los átomos. El balance entre las fuerzas de enlace, las interacciones electrostáticas y las fuerzas de repulsión resultó ser crítico para lograr una simulación estable. Inicialmente, observamos comportamientos erráticos en las moléculas debido a valores inadecuados de los parámetros físicos. La solución implicó un proceso iterativo de ajuste de las constantes, particularmente los factores de escala para las fuerzas de Coulomb y de repulsión, hasta lograr un comportamiento molecular realista sin inestabilidades numéricas.

4 Resultados de la práctica

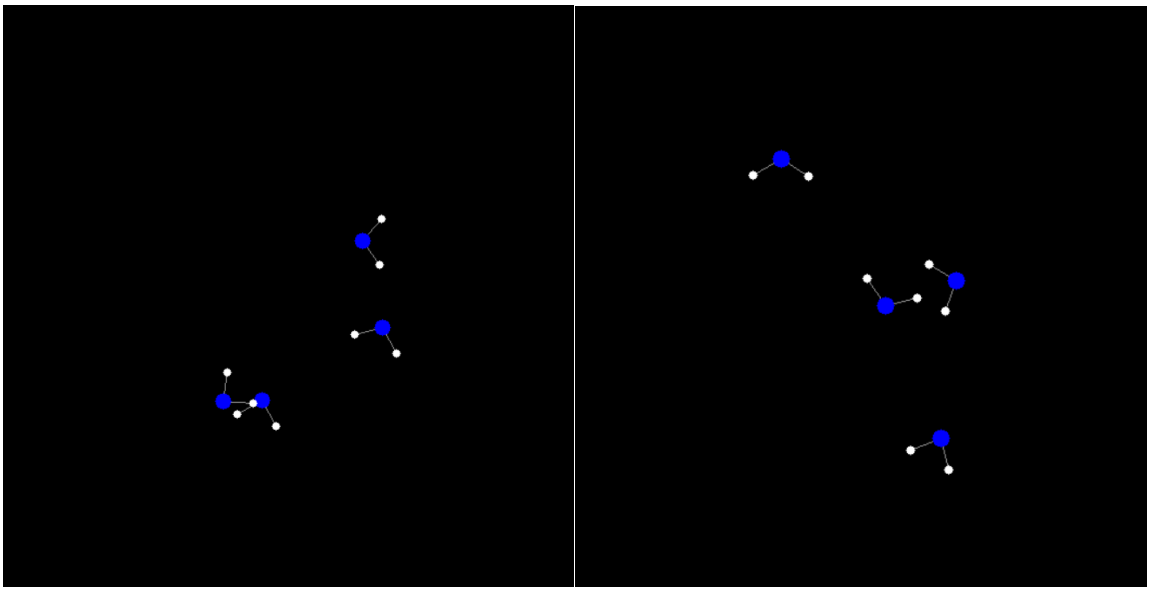

Figura 5 Resultados de la práctica

La simulación de dinámica molecular de las cuatro moléculas de agua en 2D produjo resultados que permitieron observar varios fenómenos físicos importantes:

4.1 Comportamiento de las moléculas de agua

1. **Mantenimiento de la estructura molecular**: Las moléculas de agua mantuvieron su geometría característica durante toda la simulación, con los enlaces O-H oscilando alrededor de su longitud de equilibrio y el ángulo H-O-H fluctuando cerca de los 104.5°.
2. **Movimiento browniano**: Se observó que las moléculas de agua exhibían un movimiento aleatorio característico, resultado de las interacciones entre ellas y de las condiciones iniciales aleatorias.
3. **Interacciones electrostáticas**: Las cargas parciales en los átomos de oxígeno e hidrógeno generaron fuerzas de atracción y repulsión entre diferentes moléculas, lo que influyó significativamente en su comportamiento colectivo.

5 Conclusiones y recomendaciones.

La práctica de simulación de dinámica molecular de moléculas de agua en 2D ha permitido comprender el comportamiento las moléculas. A través de esta implementación, hemos podido observar cómo las interacciones físicas básicas —enlaces covalentes, fuerzas electrostáticas y repulsiones— determinan la dinámica colectiva de las moléculas de agua.

La simulación ha demostrado ser efectiva para visualizar conceptos abstractos de la física molecular, como los enlaces, las interacciones entre moléculas y el movimiento resultante de estas interacciones. A pesar de las simplificaciones a un modelo bidimensional, los resultados obtenidos capturan aspectos esenciales del comportamiento de las moléculas de agua.

Este ejercicio ha subrayado la importancia del equilibrio entre la precisión física y la eficiencia computacional en las simulaciones de dinámica molecular. La elección de parámetros adecuados y algoritmos de integración estables resulta importante para obtener resultados físicamente significativos. En nuestra implementación, el algoritmo de Verlet modificado demostró ser suficientemente preciso para mantener la estabilidad de la simulación durante los 2000 pasos.

En conclusión, esta práctica demostró cómo incluso un modelo simplificado puede capturar aspectos esenciales del comportamiento molecular. La comprensión adquirida sobre las interacciones físicas y los algoritmos de simulación ayudo el estudio de moléculas que son fundamentales en la bioinformática.

Para futuras implementaciones, recomendamos considerar las siguientes mejoras:

- Expansión a 3D: Desarrollar una versión tridimensional que capture completamente la geometría tetraédrica del agua
- Implementación de termostatos: Incorporar algoritmos para controlar la temperatura del sistema
- Análisis cuantitativo: Añadir funcionalidades para calcular propiedades físicas como la función de distribución radial o el coeficiente de difusión

6 Bibliografía.

[1]. D. Frenkel and B. Smit, Understanding Molecular Simulation: From Algorithms to Applications, 2nd ed. San Diego, CA, USA: Academic Press, 2002.

[2]. M. P. Allen and D. J. Tildesley, Computer Simulation of Liquids, 2nd ed. Oxford, UK: Oxford University Press, 2017.

[3]. J. A. McCammon and S. C. Harvey, Dynamics of Proteins and Nucleic Acids. Cambridge, UK: Cambridge University Press, 1987.

[4]. B. R. Brooks, C. L. Brooks III, A. D. Mackerell Jr, L. Nilsson, R. J. Petrella, L. Smooth, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch, A. Caflisch, L. Caves, Q. Cui, A. R. Dinner, M. Feig, S. Fischer, J. Gohlke, S. W. Homans, T. Horikawa, A. Karplus, S. V. Kornev, A. Lazim, R. Luo, B. Ma, T. J. Muradoglu, M. K. Musalev, M. Nikolova, E. Nodelman, M. S. den Numford, J. A. P. Orts Gonzalez, C. Oostenbrink, J. C. Pan, W. Petukhov, B. Rost, C. Sagui, S. Y. Tehan, R. G. Vangunsteren, B. Voelz, J. Wang, X. Wei, Q. Wolf, X. Wu, and L. Yang, "CHARMM: The biomolecular simulation program," J. Comput. Chem., vol. 30, no. 10, pp. 1545-1614, 2009.

[5]. D. A. Case, T. E. Cheatham III, T. Darden, H. Gohlke, R. Luo, K. M. Merz Jr, A. Onufriev, C. Simmerling, B. Wang, and R. E. Woods, "AMBER 11," University of California, San Francisco, 2010.

7 Código

```
import cv2
import numpy as np
import time
from IPython import display as display
import ipywidgets as ipw
import PIL
from io import BytesIO
import math

# Canvas setup
maxX = 600
maxY = 600
x0 = int(maxX/2)
y0 = int(maxY/2)

# Colors
oxygen_color = (0, 0, 255)    # Red for oxygen
hydrogen_color = (255, 255, 255)  # White for hydrogen

# Physical constants
delta_t = 0.05
m_oxygen = 16.0
m_hydrogen = 1.0
k_bond = 2.0         # Spring constant for O-H bond
d_OH = 30.0          # Equilibrium O-H bond length
angle_HOH = 104.5 * math.pi / 180  # H-O-H bond angle in radians
q_oxygen = -2.0      # Partial charge on oxygen
q_hydrogen = 1.0     # Partial charge on hydrogen
coulomb_factor = 200.0  # Scaling factor for electrostatic forces
repulsion_factor = 5000.0  # Repulsion factor between molecules

# Class for atoms
class Atom:
    def __init__(self, x, y, mass, charge, color, radius):
        self.x = x
        self.y = y
        self.vx = 0
        self.vy = 0
        self.ax = 0
        self.ay = 0
        self.fx = 0
        self.fy = 0
        self.mass = mass
```

```
        self.charge = charge
        self.color = color
        self.radius = radius

    def draw(self, img):
        cv2.circle(img, (int(x0+self.x), int(y0-self.y)), self.radius,
self.color, -1)

    def update_force(self):
        self.fx = 0
        self.fy = 0

        # Add gravity (optional, can be set to zero for pure molecular
simulation)
        gx, gy = 0, -0.0 * self.mass
        self.fx += gx
        self.fy += gy

    def update_position(self):
        # Update acceleration
        self.ax = self.fx / self.mass
        self.ay = self.fy / self.mass

        # Update velocity
        self.vx += self.ax * delta_t
        self.vy += self.ay * delta_t

        # Update position
        self.x += self.vx * delta_t + 0.5 * self.ax * (delta_t**2)
        self.y += self.vy * delta_t + 0.5 * self.ay * (delta_t**2)

        # Boundary conditions (wrap around)
        if self.x > maxX/2:
            self.x = -maxX/2
        elif self.x < -maxX/2:
            self.x = maxX/2

        if self.y > maxY/2:
            self.y = -maxY/2
        elif self.y < -maxY/2:
            self.y = maxY/2

# Class for water molecule
class WaterMolecule:
    def __init__(self, center_x, center_y, initial_angle=0):
```

```
        # Create oxygen atom at the center
        self.oxygen = Atom(center_x, center_y, m_oxygen, q_oxygen,
oxygen_color, 8)

        # Create two hydrogen atoms
        dx = d_OH * math.cos(initial_angle)
        dy = d_OH * math.sin(initial_angle)

        dx2 = d_OH * math.cos(initial_angle + angle_HOH)
        dy2 = d_OH * math.sin(initial_angle + angle_HOH)

        self.hydrogen1 = Atom(center_x + dx, center_y + dy,
m_hydrogen, q_hydrogen, hydrogen_color, 4)
        self.hydrogen2 = Atom(center_x + dx2, center_y + dy2,
m_hydrogen, q_hydrogen, hydrogen_color, 4)

        # Initial velocities
        self.oxygen.vx = np.random.uniform(-1, 1)
        self.oxygen.vy = np.random.uniform(-1, 1)

        # Store atoms in a list for easy iteration
        self.atoms = [self.oxygen, self.hydrogen1, self.hydrogen2]

    def draw(self, img):
        # Draw bonds first (so atoms appear on top)
        cv2.line(img,
                 (int(x0+self.oxygen.x), int(y0-self.oxygen.y)),
                 (int(x0+self.hydrogen1.x), int(y0-self.hydrogen1.y)),
                 (150, 150, 150), 1)
        cv2.line(img,
                 (int(x0+self.oxygen.x), int(y0-self.oxygen.y)),
                 (int(x0+self.hydrogen2.x), int(y0-self.hydrogen2.y)),
                 (150, 150, 150), 1)

        # Draw atoms
        for atom in self.atoms:
            atom.draw(img)

    def update(self):
        # Reset forces
        for atom in self.atoms:
            atom.fx = 0
            atom.fy = 0

        # Calculate bond forces (springs between O and H)
```

```
        for hydrogen in [self.hydrogen1, self.hydrogen2]:
            # Spring force
            dx = hydrogen.x - self.oxygen.x
            dy = hydrogen.y - self.oxygen.y
            distance = math.sqrt(dx*dx + dy*dy)

            # Unit vector
            if distance > 0:
                unit_x = dx / distance
                unit_y = dy / distance
            else:
                unit_x, unit_y = 1, 0

            # Spring force magnitude (Hooke's law)
            force_magnitude = -k_bond * (distance - d_OH)

            # Apply force to both atoms (equal and opposite)
            force_x = force_magnitude * unit_x
            force_y = force_magnitude * unit_y

            hydrogen.fx += force_x
            hydrogen.fy += force_y
            self.oxygen.fx -= force_x
            self.oxygen.fy -= force_y

            # Coulomb force between O and H
            coulomb_magnitude = coulomb_factor * hydrogen.charge *
self.oxygen.charge / (distance * distance)
            coulomb_fx = coulomb_magnitude * unit_x
            coulomb_fy = coulomb_magnitude * unit_y

            hydrogen.fx += coulomb_fx
            hydrogen.fy += coulomb_fy
            self.oxygen.fx -= coulomb_fx
            self.oxygen.fy -= coulomb_fy

        # H-H interaction (repulsion between hydrogens)
        dx = self.hydrogen2.x - self.hydrogen1.x
        dy = self.hydrogen2.y - self.hydrogen1.y
        distance = math.sqrt(dx*dx + dy*dy)

        if distance > 0:
            unit_x = dx / distance
            unit_y = dy / distance
```

```
            # Coulomb force between H and H
            coulomb_magnitude = coulomb_factor * self.hydrogen1.charge
* self.hydrogen2.charge / (distance * distance)
            coulomb_fx = coulomb_magnitude * unit_x
            coulomb_fy = coulomb_magnitude * unit_y

            self.hydrogen1.fx += coulomb_fx
            self.hydrogen1.fy += coulomb_fy
            self.hydrogen2.fx -= coulomb_fx
            self.hydrogen2.fy -= coulomb_fy

        # Update positions of all atoms
        for atom in self.atoms:
            atom.update_position()

# Function to handle interactions between water molecules
def calculate_molecular_interactions(molecules):
    num_molecules = len(molecules)

    for i in range(num_molecules):
        for j in range(i+1, num_molecules):
            mol1 = molecules[i]
            mol2 = molecules[j]

            # Interaction between all atoms of different molecules
            for atom1 in mol1.atoms:
                for atom2 in mol2.atoms:
                    dx = atom2.x - atom1.x
                    dy = atom2.y - atom1.y
                    distance_squared = dx*dx + dy*dy
                    distance = math.sqrt(distance_squared)

                    if distance < 0.1:  # Prevent division by zero
                        distance = 0.1

                    # Unit vector
                    unit_x = dx / distance
                    unit_y = dy / distance

                    # Coulomb force
                    coulomb_magnitude = coulomb_factor * atom1.charge
* atom2.charge / distance_squared

                    # Repulsion force (to prevent atoms from getting
too close)
```

```
                    repulsion_magnitude = 0
                    if distance < 20:
                        repulsion_magnitude = repulsion_factor /
(distance_squared * distance)

                    # Total force
                    force_magnitude = coulomb_magnitude +
repulsion_magnitude
                    force_x = force_magnitude * unit_x
                    force_y = force_magnitude * unit_y

                    # Apply force to both atoms
                    atom1.fx += force_x
                    atom1.fy += force_y
                    atom2.fx -= force_x
                    atom2.fy -= force_y

# Create water molecules at random positions
num_molecules = 4
water_molecules = []
for i in range(num_molecules):
    x = np.random.uniform(-maxX/4, maxX/4)
    y = np.random.uniform(-maxY/4, maxY/4)
    angle = np.random.uniform(0, 2*math.pi)
    water_molecules.append(WaterMolecule(x, y, angle))

# Create display
img = np.zeros((maxX, maxY, 3), dtype="uint8")
wIm = ipw.Image()
display.display(wIm)

# Main simulation loop
MaxIterations = 2000
for count in range(MaxIterations):
    # Clear the image
    img[:] = (0, 0, 0)

    # Calculate intermolecular forces
    calculate_molecular_interactions(water_molecules)

    # Update each molecule
    for molecule in water_molecules:
        molecule.update()
        molecule.draw(img)
```

```
    # Display the current frame
    pilIm = PIL.Image.fromarray(img, mode="RGB")
    with BytesIO() as fOut:
        pilIm.save(fOut, format="png")
        byPng = fOut.getvalue()

    # Update the image in the browser
    wIm.value = byPng

    # Add simulation info
    cv2.putText(img, f"Frame: {count}", (20, 20),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)

    # Small delay to control simulation speed
    time.sleep(0.01)
```