Instituto Politécnico Nacional
Escuela Superior de Cómputo
Bioinformatics
Introducción a la Dinámica Molecular
Tinoco Videgaray Sergio Ernesto
Rosas Trigueros Jorge Luis
17/03/2024
21/03/2024

Marco teórico.

La mecánica molecular es un conjunto de técnicas de simulación que aplica la mecánica clásica para emular sistemas moleculares. Puede ser utilizada para estudiar sistemas moleculares que varían en tamaño y complejidad, desde moléculas orgánicas pequeñas, hasta sistemas biológicos complejos (proteínas, polisacáridos, ácidos nucleicos), o materiales que contienen miles de millones de átomos.

El modelado de moléculas de la Mecánica Molecular se basa en cuatro principios [1]:

- Los núcleos y los electrones se agrupan en partículas puntuales.
- Las partículas son esféricas (con un radio obtenido experimentalmente) y tienen una carga (obtenida a partir de la teoría).
- Las interacciones ente partículas están basadas en potenciales de la mecánica clásica.
- Las interacciones de la distribución espacial de las partículas y sus energías.

La Mecánica Molecular representa a las moléculas como un conjunto de esferas de diferente masa (m) y conectadas por muelles. Se distinguen distintos tipos de interacciones en este sistema de Esferas conectadas por muelles:

- Interacciones de enlace (tensión)
- Interacciones angulares (Flexión)
- Interacciones de Torsión
- Interacciones de van der Waals
- Interacciones electrostáticas
- Interacciones fuera del plano

La suma de todas las interacciones constituye la energía total de la molécula (figura 1).

$$E_{Total} = \sum E_{Enlaces} + E_{Flexiones} + E_{Torsiones} + E_{VdW} + E_{Elec} + E_{OOP}$$

Figura 1. Ecuación que describe la energía total de una molécula.

Ley de Hooke.

Durante el siglo XVII, al estudiar los resortes y la elasticidad, el físico Robert Hooke observó que para muchos materiales la curva de esfuerzo vs. deformación tiene una región lineal. Dentro de ciertos límites, la fuerza requerida para estirar un objeto elástico, como un resorte de metal, es directamente proporcional a la extensión del resorte (figura 2).

$$E_{EnlaceAB} = \frac{1}{2} K_{AB}(x_{AB} - r_{AB_{eq}})^2$$

Figura 2. Ecuación de la ley de Hooke.

Donde la KAB es la constante de fuerza para el enlace entre los átomos A y B, y $rABeq$ es la distancia de equilibrio para ese mismo enlace.

La forma de la función de la Ley de Hooke es similar a la del Potencial de Morse en las cercanías del equilibrio, pero cuando la molécula se aparta excesivamente de esta distancia de equilibrio el valor de la energía difiere mucho del real.

Ley de Coulomb.

Debe su nombre al físico francés Charles-Augustin de Coulomb, quien en 1785 enunció esta ley, a partir de la cual se puede predecir cuál será la fuerza electrostática de atracción o repulsión existente entre dos partículas según su carga eléctrica y la distancia que existe entre ambas [1].

Las interacciones electrostáticas se describen utilizando el potencial de Coulomb con la formula clásica (figura 3).

$$E_{elec} = \frac{1}{4\pi\epsilon_0} \frac{Q_A Q_B}{r_{AB}}$$

Figura 3. Ecuación que describe la Ley de Coulomb.

Donde:

- 1/4πε0: es la constante de Coulomb o constante eléctrica de proporcionalidad. La fuerza varía según la permitividad eléctrica (ε) del medio, bien sea agua, aire, aceite, vacío, entre otros.
- QA, QB: valor de las cargas eléctricas medidas en Coulomb (C).
- r_{AB}: distancia que separa a las cargas.

Material y equipo.

Para esta práctica se utilizaron los siguientes materiales y recursos:

- Un equipo con acceso a internet.
- Un entorno de desarrollo para Python como Colab o Spyder.

Desarrollo.

1. Definir las clases Átomo y Molécula.

En este paso se codificaron las clases Átomo y “MoleculaAgua” las cuales van a almacenar los datos correspondientes a cada molécula de agua y los datos de sus átomos como la carga, la posición, la velocidad y la aceleración (Figura 4 y 5).

```
class Atomo():  #Clase atomo
    def __init__(self,q,r,v,a):
        self.q=q    #Carga
        self.r=r    #Posicion
        self.v=v    #Velocidad
        self.a=a    #Aceleracion
```

Figura 4. Código de la clase Átomo.

```
class MoleculaAgua():   #Clase Molecula de agua
    def __init__(self,q,r,v,f,angle):
        self.atomos=[]  #Arreglo atomos
        self.atomos.append(Atomo(q[0],r,v,f))
        self.atomos.append(Atomo(q[1],(r[0]+50,r[1]-50),v,f))
        self.atomos.append(Atomo(q[2],(r[0]-50,r[1]-50),v,f))
        self.oxigeno=self.atomos[0] if q[0]<0 else self.atomos[1]
        self.angle=angle    #Angulo de flexion


    def getAtomos(self):    #Devolver lista de atomos
        return self.atomos
```

Figura 5. Código de la Clase MoleculaAgua

2. Definir las leyes de movimiento de la mecánica clásica.

En esta sección se codificaron las funciones que modelan las ecuaciones de movimiento en la mecánica clásica, las cuales describen el comportamiento de un cuerpo a lo largo del tiempo dada un conjunto de parámetros como lo es la posición, la velocidad inicial, la aceleración, así como otras leyes de la mecánica como la ley de Hooke que describe el comportamiento de un cuerpo sujeto a un resorte y la ley de Coulomb que modela la fuerza de atracción o repulsión entre dos cargas puntuales.

- Función de posición

Primeramente, se define la función que actualiza la posición de los átomos en función de su posición inicial, su velocidad y su aceleración (figura 7) siguiendo la ecuación de la figura 6.

$$\Delta x = v_0 \Delta t + \frac{1}{2} a \Delta t^2$$

Figura 6. Ecuación de la posición en función de la velocidad y la aceleración.

```python
def new_r(r,v,a,dt):  #Actualizar posicion
  rx,ry=r
  vx,vy=v
  ax,ay=a

  new_rx = rx + vx*dt + 0.5*ax*(dt**2)
  new_ry = ry + vy*dt + 0.5*ay*(dt**2)
  return new_rx, new_ry
```

Figura 7. Código de la función que modela la ecuación de la figura 5.

De igual forma se define la función que actualiza la velocidad de la partícula en función de su velocidad inicial y su aceleración (figura 8 y 9).

$$\vec{v}(\Delta t) = \vec{v}_0 + \vec{a}\Delta t$$

Figura 8. Ecuación de la velocidad en función de la velocidad inicial, la aceleración y el instante de tiempo.

```python
def new_v(v,a,dt):  #Actualizar velocidad
  vx,vy=v
  ax,ay=a

  new_vx = vx + ax*dt
  new_vy = vy + ay*dt
  return new_vx,new_vy
```

Figura 9. Código de la función que modela la ecuación de la figura 8.

Se define la función que actualiza la aceleración del átomo en función de su fuerza y su masa siguiendo la ecuación de la figura 10.

$$a = \frac{F}{m}$$

Figura 10. Ecuación para calcular la aceleración en función de la masa y la fuerza.

En este caso se estará trabajando con una partícula de masa unitaria, por lo que se retornará el valor de la fuerza (figura 11).

```python
def new_a(f):   #Actualizar aceleracion
  fx,fy=f
  #Masa unitaria
  new_ax =  fx/1
  new_ay =  fy/1
  return [new_ax,new_ay]
```

Figura 11. Código de la función que modela la ecuación de la figura 10.

Se define la función para calcular la fuerza de atracción o repulsión entre los átomos de cada molécula siguiendo la ecuación de la ley de Coulomb (figura 12).

```
def Fc(a,b,qA,qB):  #Actualizar fuerza entre cargas
    ax,ay=a
    bx,by=b
    ABx,ABy=bx-ax,by-ay
    magAB=np.sqrt(ABx**2+ABy**2)
    nABx,nABy=ABx/magAB,ABy/magAB
    k=-90
    magF=k*(qA*qB)/(magAB**2)
    return nABx*magF,nABy*magF
```

Figura 12. Código de la función que modela la ecuación de la figura 3.

Se define la función que calcula la fuerza de rebote del enlace entre átomos basando se en la Ley de Hooke (figura 13).

```
def Fs(a,b,d_eq,k): #Actualizar fuerza elastica
    ax,ay=a
    bx,by=b
    ABx,ABy=bx-ax,by-ay
    magAB=np.sqrt(ABx**2+ABy**2)
    nABx,nABy=ABx/magAB,ABy/magAB
    magF=-k*(magAB-d_eq)
    return nABx*magF,nABy*magF
```

Figura 13. Código de la función que modela la ecuación de la figura 2.

Se define la función para calcular el ángulo de flexión por medio del producto dividido por la longitud de los enlaces (figura 14).

```
def getAngle(A,B,C):  #Obtener angulo de flexion
    BA=np.subtract(A,B)
    BC=np.subtract(C,B)
    magBA=np.linalg.norm(BA)
    magBC=np.linalg.norm(BC)
    return np.degrees(np.arccos(np.dot(BA,BC)/(magBA*magBC)))
```

Figura 14. Código de la función que calcula el ángulo de flexión.

Se define la función para calcular la energía de potencial de flexión basándose en la ley de Hooke, pero ahora sobre los ángulos de flexión (figura 15).

```
def U(A,B,C,k,a_eq,molecula): #Energia de potencial de flexion
    global img
    molecula.angle=getAngle(A,B,C)
    return k*((molecula.angle-a_eq)**2)
```

Figura 15. Código de la función que calcula el potencial de flexión.

Se define la función que calcula el gradiente del potencial de flexión de cada átomo con respecto al ángulo formado por los 3 átomos de la molécula de agua (figura 16).

```
def gradU(A,B,C,k,a_eq,atomo_mover,molecula): #Gradiente de flexion
  D=0.01
  u=U(A,B,C,k,a_eq,molecula)
  u_dx=u_dy=u
  if atomo_mover==A:      #Mover A
    u_dx=U((A[0]+D,A[1]),B,C,k,a_eq,molecula)
    u_dy=U((A[0],A[1]+D),B,C,k,a_eq,molecula)
  elif atomo_mover == B:  #Mover B
    u_dx=U(A,(B[0]+D,B[1]),C,k,a_eq,molecula)
    u_dy=U(A,(B[0],B[1]+D),C,k,a_eq,molecula)
  elif atomo_mover==C:    #Mover C
    u_dx=U(A,B,(C[0]+D,C[1]),k,a_eq,molecula)
    u_dy=U(A,B,(C[0],C[1]+D),k,a_eq,molecula)

  return (u_dx-u)/D,(u_dy-u)/D
```

Figura 16. Código de la función que calcula el gradiente del ángulo de flexión.

3. Funciones de la simulación.

Se define la función que grafica las moléculas en un archivo de imagen generado automáticamente. Se genera un circulo de color rojo en la posición del átomo de oxígeno y una línea grisácea que vincula cada átomo de hidrogeno con su respectivo oxígeno (figura 17).

```
def mostrarMoleculas(moleculas,wIm): #Graficar moleculas
    global tam,img
    x=y=tam//2
    for mol in moleculas:
      #Atomo oxigeno
      cv2.circle(img,(int(x+mol.oxigeno.r[0]),int(y-mol.oxigeno.r[1])),10,(255,0,0),-1)
      for atomo in mol.getAtomos():
        #Atomos hidrogeno
        if(atomo.q>0):  #Si tiene carga positiva
          #Generar linea del enlace
          cv2.line(img,(int(x+atomo.r[0]),int(y-atomo.r[1])),
                   (int(x+mol.oxigeno.r[0]),int(y-mol.oxigeno.r[1])) ,
                    (180,180,180), 3)
          #Atomo hidrogeno
          cv2.circle(img,(int(x+atomo.r[0]),int(y-atomo.r[1])),10,(255,255,255),-1)
        pilIm = Image.fromarray(img, mode="RGB")
    with BytesIO() as fOut: #Actualizar imagen
        pilIm.save(fOut, format="png")
        byPng = fOut.getvalue()
    wIm.value=byPng
```

Figura 17. Código de la función que grafica los átomos con sus respectivos enlaces.

Se define una función que obtiene la fuerza electrostática de cada átomo con respecto a los demás átomos utilizando la función de la ley de Coulomb (figura 18).

```
def update_Coulomb(moleculas):  #Actualizar fuerzas entre atomos
    global dt
    atomos=[]
    for molecula in moleculas:  #Por cada molecula
        for atomo in molecula.getAtomos():  #Por cada atomo
            atomos.append(atomo)    #Agregar a la lista de atomos
    for i in range(len(atomos)):
        fx=fy=0
        for j in range(len(atomos)):
            if atomos[i].r!=atomos[j].r:    #Atomos distintos
                #Aplicar ley de Coulomb
                fcx,fcy=Fc(atomos[i].r,atomos[j].r,atomos[i].q,atomos[j].q)
                fx+=fcx;fy+=fcy #Acumular fuerza total
        atomos[i].f=new_a((fx,fy))  #Actualizar fuerza/aceleracion
        atomos[i].v=new_v(atomos[i].v,atomos[i].f,dt)   #Actualizar velocidad
        atomos[i].r=new_r(atomos[i].r,atomos[i].v,atomos[i].f,dt)   #Actualizar posicion
```

Figura 18. Código de la función que actualiza la fuerza electrostática de cada átomo con respecto a los demás átomos en el espacio definido.

Se define la función que actualiza los parámetros de las moléculas definiendo los parámetros iniciales (figura 19).

```
def update(moleculas,ka,a_eq):  #Actualizar moleculas
  global dt
  #Valores iniciales
  fsx=fsy=fx=fy=fuax=fuay=0
  angle_eq=a_eq
  d_eq=70
```

Figura 19. Código de la función update donde se definen los parámetros iniciales.

Se invoca la función para calcular la fuerza electrostática de cada átomo y actualizar sus parámetros. Posteriormente se recorre cada molécula en la lista de moléculas y se obtienen los átomos de cada molécula para definir el orden de los átomos a iterar (figura 20).

```
update_Coulomb(moleculas) #Acualizar fuerzas entre cargas
for molecula in moleculas:  #Por cada moleucla
  atomos=molecula.getAtomos()
  #Definir orden de atomos
  if atomos[0].q<0:
    A,B,C=atomos[2],atomos[0],atomos[1]
  elif atomos[1].q<0:
    A,B,C=atomos[0],atomos[1],atomos[2]
  else:
    A,B,C=atomos[1],atomos[2],atomos[0]
```

Figura 20. Código de la función update donde se itera la lista de moléculas y se obtienen los átomos a iterar.

Se itera cada átomo en la molécula y se calcula la fuerza de rebote en cada enlace. De igual forma se calcula la fuerza del gradiente de flexión del ángulo de enlaces. Posteriormente se suman las fuerzas para obtener la fuerza total que se aplica a cada átomo (figura 21).

```
for atomo in atomos:  #Por cada atomo
  #Spring
  if atomo!=B:  #Fuerza de rebote del enlace respecto al oxigeno
    fsx,fsy=Fs(B.r,atomo.r,d_eq,ka)
  else: #Fuerza de rebote respecto a hidrogenos
    fsAx,fsAy=Fs(A.r,B.r,d_eq,ka)
    fsBx,fsBy=Fs(C.r,B.r,d_eq,ka)
    fsx=fsAx+fsBx;fsy=fsAy+fsBy
  #Fuerza gradiente de flexion
  fuax,fuay=gradU(A.r,B.r,C.r,ka,angle_eq,atomo.r,molecula)
  #Fuerza total
  fx=(fsx-fuax);fy=(fsy-fuay)
```

Figura 21. Código de la función update donde se itera cada átomo y se calculan sus fuerzas.

Finalmente se actualizan los valores de su aceleración (fuerza), su velocidad y su posición utilizando las funciones definidas previamente (figura 22).

```
#Actualizar valores
atomo.f=new_a((fx,fy))  #Actualizar fuerza
atomo.v=new_v(atomo.v,atomo.f,dt) #Actualizar velocidad
atomo.r=new_r(atomo.r,atomo.v,atomo.f,dt) #Actualizar posicion
```

Figura 22. Código de la función update donde se actualizan los parámetros de cada átomo.

Se define la función principal que recibe los hiperparámetros para la simulación. Dentro de la función se crean los objetos de 2 molecular de agua que se van a simular simultáneamente (figura 23).

```
def main(qa,a_eq,ka):
    #Recibe nivel de carga, angulo de equilibrio y la constante k
    q=qa    #Cargas uniformes
    #Objetos molecula de agua
    moleculas=[]
    moleculas.append(MoleculaAgua((q,-q,q),(-70,100),(0,0),[0,0],0))
    moleculas.append(MoleculaAgua((q,q,-q),(60,-70),(0,0),[0,0],0))
```

Figura 23. Código de la función principal que recibe los hiperparámetros para la simulación.

Se definen más hiperparámetros como el número de iteraciones, el tamaño del espacio de la simulación, el tiempo y la matriz base para graficar las moléculas (figura 24).

```
MaxIterations = 500
global tam,dt,img
tam=400
dt=0.02
img = np.zeros((tam, tam, 3), dtype="uint8")
wIm = ipw.Image()
display.display(wIm)
```

Figura 24. Hiperparámetros de la simulación.

Se define el bucle principal que va a iterar 500 veces para visualizar los cambios en la simulación. Dentro del bucle se va a mostrar en la imagen el número de iteración y los ángulos de las moléculas, así como también se va a invocar a la función para actualizar los parámetros de las moléculas y finalmente visualizar las moléculas (figura 25).

```
for count in range(MaxIterations):  #Iterar
    img[:]=(0,0,0)
    #Numero de iteracion
    cv2.putText( img,str(count),(tam//3,25),
                cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,255))
    #Angulo de la molecula 1
    cv2.putText(img,str(int(moleculas[0].angle)),(30,55),
                cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,255))
    #Angulo de la molecula 2
    cv2.putText(img,str(int(moleculas[1].angle)),(3*tam//4,55),
                cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,255))
    update(moleculas,ka,a_eq)
    mostrarMoleculas(moleculas,wIm)
```

Figura 25. Bucle principal donde se invocan las funciones de actualización y visualización de las moléculas.

Pruebas y resultados.

- Primera simulación.

Inicialmente se ejecutó el programa con un valor de 3 Coulombs para cada carga, un ángulo de equilibrio de 30 grados y un valor de 7 para la constante k (figura 26 a 29).

```
#carga,angulo,cte
main(3,30,7)
```

Figura 26. Parámetros de la ejecución de la primera simulación.

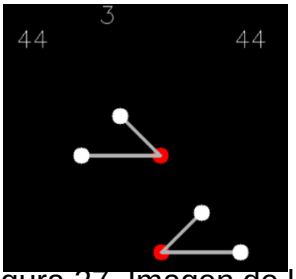

Figura 27. Imagen de la primera simulación en la iteración número 3.

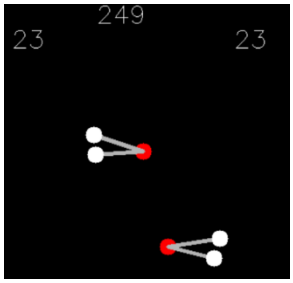

Figura 28. Imagen de la primera simulación en la iteración número 249.

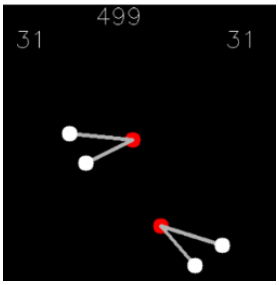

Figura 29. Imagen de la primera simulación en la iteración número 499.

- Segunda simulación.

Posteriormente se ejecutó el programa con un valor de 7 Coulombs para cada carga, un ángulo de equilibrio de 45 grados y un valor de 7 para la constante k (figura 30 a 33).

```
#carga,angulo,cte
main(7,45,7)
```

Figura 30. Parámetros de la ejecución de la segunda simulación.

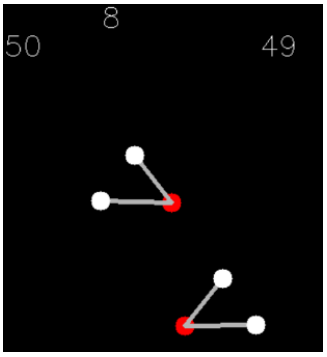

Figura 31. Imagen de la segunda simulación en la iteración número 8.

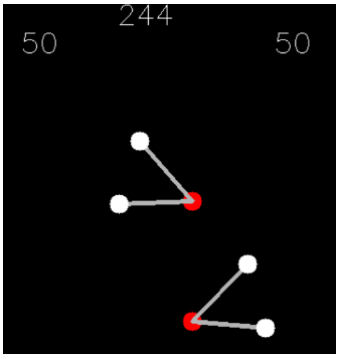

Figura 32. Imagen de la segunda simulación en la iteración número 244.

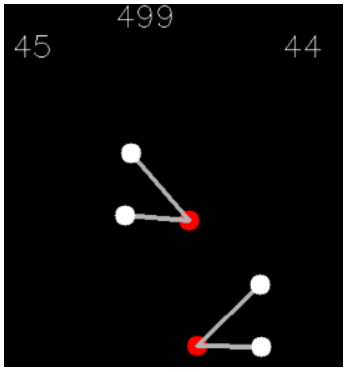

Figura 33. Imagen de la segunda simulación en la iteración número 499.

- Tercera iteración.

Se ejecutó el programa ahora con un valor de 7 Coulombs para cada carga, un ángulo de equilibrio de 60 grados y un valor de 7 para la constante k (figura 34 a 37).

```
#carga,angulo,cte
main(7,60,7)
```

Figura 34. Parámetros de la ejecución de la tercera simulación.

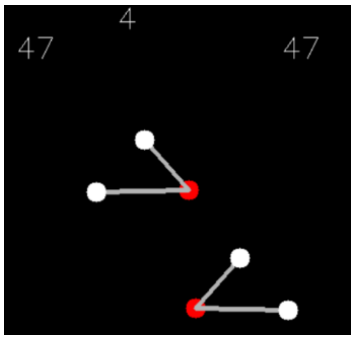

Figura 35. Imagen de la tercera simulación en la iteración número 4.

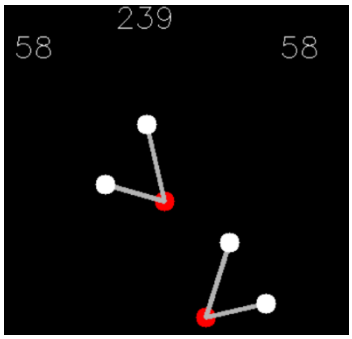

Figura 36. Imagen de la tercera simulación en la iteración número 239.

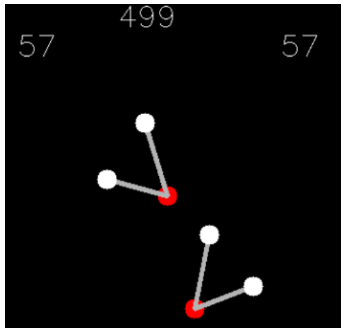

Figura 37. Imagen de la tercera simulación en la iteración número 499.

- Cuarta simulación.

Finalmente se ejecutó con una carga uniforme de 7 Coulombs y un ángulo de 104.5 grados que corresponde con el ángulo de enlace real de una molécula de agua (figura 38 a 41).

```
#carga,angulo,cte
main(5,104.5,7)
```

Figura 38. Parámetros de la ejecución de la cuarta simulación.

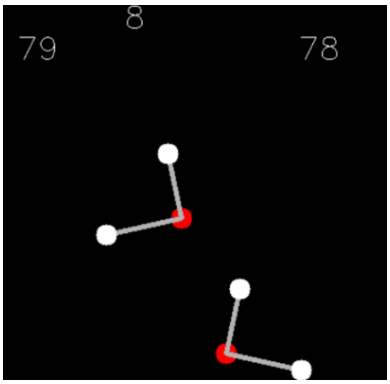

Figura 39. Imagen de la cuarta simulación en la iteración número 8.

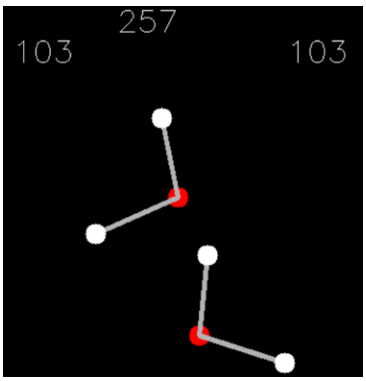

Figura 40. Imagen de la cuarta simulación en la iteración número 257.

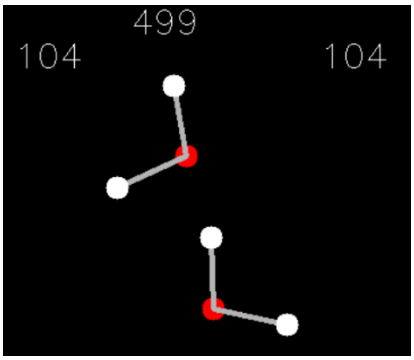

Figura 41. Imagen de la cuarta simulación en la iteración número 499.

Como se pudo observar, todas las simulaciones convergieron el resultado esperado de acuerdo con los parámetros introducidos.

Conclusiones y recomendaciones.

En esta práctica se realizó la simulación dos partículas de agua utilizando los principios de la mecánica molecular aplicando leyes de la mecánica clásica con un enfoque orientado a objetos.

La dinámica molecular consiste en un conjunto de técnicas que modelan el comportamiento de las moléculas utilizando ecuaciones de la mecánica clásica como las leyes de movimiento con movimiento uniformemente acelerado, la ley de Hooke, la electrostática, etc. Lo cual simplifica la simulación de estos modelos en un entorno computacional.

Integrar estos modelos a gran escala permite visualizar el comportamiento de varios sistemas a nivel molecular como las proteínas, lo cual resulta una herramienta útil en el campo de las ciencias biológicas y medicinales.

Referencias.

[1] G. Torres, F. Nájera, Y. Vida (2009). Tema 1: Introducción a la Química Computacional. OCW-Universidad de Málaga, http://ocw.uma.es