UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

INSTITUTO DE INVESTIGACIÓN EN CIENCIAS BÁSICAS Y APLICADAS

CENTRO DE INVESTIGACIONES QUÍMICAS

“DESARROLLO DE HERRAMIENTAS METODOLÓGICAS DE SIMULACIÓN NUMÉRICA PARA EL ESTUDIO DE INTERFASES ACUOSAS”

TESIS

QUE PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS

PRESENTA:

I.Q. ANTHONI ALCARAZ TORRES

DIRECTORA DE TESIS:

DRA. MARGARITA ISABEL BERNAL URUCHURTU

SINODALES:

DR. JORGE URUCHURTU CHAVARÍN

DR. HUMBERTO SAINT-MARTIN POSADA

DR. ANTONIO GAMBOA SUÁREZ

DR. JORGE HERNÁNDEZ COBOS

DR. MINNHUY HÔ

CUERNAVACA, MORELOS MAYO, 2018

AGRADECIMIENTOS

A CONACYT por la beca de maestría con número de registro 594599 y el apoyo económico del proyecto de investigación “Enfoques multi-escala para el estudio de halógenos en fases condensadas” con número 253716 para poder concluir mi tesis de maestría.

Al Centro de Investigaciones Químicas (CIQ) por el apoyo administrativo, equipo de cómputo e infraestructura para realizar mi proyecto de maestría. A los profesores-investigadores, del área de fisicoquímica teórica, que me guiaron en el proceso de aprender y desarrollar diferentes habilidades.

Al Laboratorio Nacional de Cómputo de Alto Desempeño (LANCAD), en especial a la Universidad Autónoma Metropolitana campus Iztapalapa por el tiempo de cálculo concedido en el clúster YOLTLA y al Centro de Investigaciones y Estudios Avanzados (CINVESTAV) por el apoyo para utilizar el clúster ABACUS: Laboratorio de Matemática Aplicada y Cómputo de Alto Rendimiento del Departamento de Matemáticas.

Al grupo de fisicoquímica teórica del CIQ, en especial a la Dra. Margarita Bernal Uruchurtu y al Dr. Antonio Gamboa Suárez por sus asesorías y enseñanzas en todo el proyecto.

Al Dr. Peter Eastman, desarrollador principal de OpenMM, quien me apoyo con la solución de diferentes dudas del uso del programa.

Al Dr. Humberto Saint-Martín y al Dr. Jorge Uruchurtu por sus comentarios y sugerencias durante el proyecto.

Al Dr. Saurabh Belsare, del grupo de Biofisica de la Universidad de California en Berkeley, por la asesoría acerca del uso de OpenMM.

DEDICATORIA

Al progreso de la ciencia

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO UNO

INTRODUCCIÓN

1. INTRODUCCIÓN

El progreso científico y tecnológico ocurrido en los últimos 70 años, ha conducido a la puesta a punto de técnicas experimentales que permiten caracterizar a nivel molecular una mayor cantidad de materiales, sustancias y especies químicas. Ejemplo de estos equipos son el microscopio de fuerza atómica o los microscopios de barrido por sondeo[1]. Gracias a estos avances podemos intentar mejorar nuestra comprensión de la naturaleza y los problemas modernos que atentan contra ella: la contaminación ambiental generada por la quema de hidrocarburos provenientes del petróleo; la contaminación de agua potable con metales pesados, por ejemplo, el envenenamiento por plomo causado por consumir agua con altas concentraciones de plomo, como es el caso de Washington en la década pasada[2] y el de Flint en el 2016[1].

La investigación científica ha contribuido, tanto a identificar el origen de los problemas señalados anteriormente como en la búsqueda de soluciones. Sin embargo, esto último no es simple ni directo. Por ejemplo, con el descubrimiento de algunos productos que se derivan de recursos renovables y causan menor impacto ambiental, como el biodiesel, su transporte en tuberías metálicas provoca corrosión y, aunque se han descubierto inhibidores verdes para este problema[3,4], el reto de aprovechar innovaciones sin generar problemas mayores es fundamental. En el caso de Flint, el agua recibida en los hogares tuvo una concentración más alta de plomo en comparación con la que se suministraba al inicio de la red desde la planta de distribución, esto sucedió al modificar las técnicas de tratamiento del agua; el aumento en la cantidad de cloro para su potabilización provocó que se acidificara. Por lo que, entre las tuberías (hechas de plomo) y la disolución electrolítica, se generaron reacciones electroquímicas, provocando la formación de residuos que corroyeron la red de distribución de agua. La corrosión, es el fenómeno electroquímico generado por las reacciones químicas producidas por la transferencia de electrones entre fases cargadas eléctricamente. De acuerdo con Michael Kappl *et al.,* es posible entender

[1] Todas las noticias del caso Flint están disponibles en: http://flintwaterstudy.org/

una interfase como la región que separa dos fases, si, consideramos algunas fases como el vapor, el líquido, el sólido, amorfo o cristalino, también podemos encontrar la combinación de fases que generan interfases como líquido-vapor, líquido-sólido, líquido-cristal, entre otras[1]. Se sabe que las reacciones electroquímicas suceden en la interfase comprendida entre el metal (puede ser una tubería o un electrodo) y la disolución acuosa. Esta interfase recibe el nombre de doble capa electroquímica y reviste vital importancia para diferentes áreas de la ciencia como electroquímica, materiales o ciencia de superficie[5–8]. La doble capa (DC) está comprendida por la capa de Nernst y la capa difusa; la DC se extiende desde la superficie del electrodo hasta terminar la capa difusa. En ella podemos encontrar moléculas de disolvente, iones solvatados, moléculas de disolvente o iones adsorbidos a la superficie del electrodo[9,10].

Experimentalmente, los procesos electroquímicos se han estudiado desde un punto de vista fenomenológico utilizando técnicas como: voltamogramas cíclicos, curvas de impedancia, curvas de ruido, curvas de electrocapilaridad, entre otras. La fisicoquímica teórica por su parte ha analizado fenómenos como el transporte de masa con modelos diferentes para su estudio que el necesario para analizar la transferencia de carga y la reactividad en celdas electroquímicas. En el primero, el modelo clásico, la naturaleza química de las substancias disueltas y/o depositadas no cambia mientras que en el segundo sí (el modelo cuántico). Por ejemplo, el transporte de masa se estudia por lo general con modelos clásicos, es decir, no se consideran las propiedades atómicas o electrónicas ya que se analiza el comportamiento estadístico de partículas como conjunto. Por otra parte, los modelos cuánticos son capaces de abordar explícitamente la transferencia de electrones, es decir, estudian detalladamente la interacción de átomos y moléculas. Por ello, la modelización de problemas relacionados con los fenómenos electroquímicos puede ser hecha con diferentes grados de aproximación.

La implementación de los modelos a los que se hace referencia en el párrafo anterior se realiza con técnicas de simulación numérica como la dinámica molecular (DM) o el Monte Carlo Metrópolis (MC). En la literatura reciente es posible identificar varios

trabajos dirigidos a avanzar en la compresión de la interfase electroquímica proporcionando información con detalle molecular de ella[11–14]. Es posible identificar que los niveles de aproximación con que lo hacen son muy distintos. La descripción de estos sistemas requiere de modelos que describan al electrodo (fase inmóvil), y de modelos que describan la disolución (modelos clásicos, flexibles y con capacidad de respuesta ante el medio). Encontrar un modelo que integre la fase sólida, la fase líquida y la interfase, requiere aún de gran cantidad de trabajo[13,15–19].

En la disolución, el disolvente como el agua es el que se encuentra en mayor cantidad, entender su comportamiento ha sido de los grandes retos que se originaron desde el siglo pasado y por ello, se han desarrollado varios modelos clásicos de agua que tratan de describir propiedades medibles experimentalmente[20]. Es posible clasificar los modelos de agua empleados, según los grados de libertad en su diseño. Por ejemplo, grados de libertad estructurales, aquellos donde la geometría de la molécula es flexible o aquellos con grados de libertad en la descripción de su respuesta electrostática.

Por un lado, en los modelos de agua con grados de libertad estructurales, se pueden reconocer dos categorías grandes: 1) los modelos rígidos, donde se mantiene fija tanto la longitud de enlace de oxígeno - hidrógeno (OH) como el valor del ángulo entre los OH y, 2) los modelos flexibles en los que tanto las longitudes de los OH como el ángulo entre ellos pueden cambiar[20,21].

Los grados de libertad asociados a la descripción clásica de la densidad electrónica han sido sujetos de representaciones diferentes[22,23]. En particular, la forma en la que se incluyen propiedades como el dipolo permanente, el dipolo inducido o la carga formal de un ion. De hecho, es posible clasificarlos por la forma en la que tratan (o no) la polarización, que es la redistribución de la densidad electrónica de una molécula ante la presencia de un campo eléctrico[24]. La mayoría de los campos de fuerza se han ajustado para considerar únicamente cargas puntuales en cada centro atómico sin incluir a la polarización o un desarrollo multipolar. Por ejemplo, el potencial *ad-hoc*

propuesto por Bernal y Fowler[25], que incluye la expresión de Coulomb para monopolos eléctricos.

Entre los modelos de electrodo que se han usado se encuentran aquellos que simulan explícitamente los átomos que conforman a la pared[26] y modelos donde el efecto del muro se estudia con una superficie sin estructura discreta[27]. Una característica importante de cualquier modelo de electrodo es contar con la posibilidad de modificar sus propiedades electrónicas: que sea polarizable y también que se pueda cargar eléctricamente[28].

Este proyecto de maestría busca generar un protocolo de simulación basado en estudios de dinámica molecular, útil para analizar el comportamiento de diferentes modelos clásicos de agua en la interfase sólido-líquido, aproximando la fase sólida a su representación más sencilla que es utilizar una pared sin estructura.

CAPÍTULO DOS

ANTECEDENTES

2. ANTECEDENTES

Entender una disolución, requiere preguntarse qué tipo de solvatación ocurre en el sistema de estudio. Cuando el disolvente es agua, se distinguen dos grandes tipos de hidratación: Por un lado, la hidratación que favorece la interacción entre el soluto y el agua se llama hidrofílica y, donde la interacción dominante es entre las moléculas de agua, recibe el nombre de hidrofóbica[29]. Desde siglo XX, ha existido gran interés en caracterizar a nivel molecular el comportamiento del agua en presencia de superficies con propiedades hidrofóbicas e hidrofílicas. Ejemplos de este tipo de sistemas son: agua en contacto con interfases no polares o interfases metálicas; una disolución electrolítica en interfases metálicas; agua dentro de una membrana biológica, la hidratación de proteínas, entre otros [30,31]. A continuación, se presenta de manera sucinta, la evolución histórica de los modelos usados en simulaciones numéricas de sistemas de dos fases que involucran superficies metálicas con agua y/o disoluciones electrolíticas.

El primer intento fue hecho por Lee *et al.,* en 1984 y consistió en estudiar mediante dinámica molecular 216 moléculas de agua, las cuales estaban confinadas entre dos superficies hidrofóbicas de tamaño infinito en una celda de simulación alargada en la dirección ***Z*** (13.16 x 13.16 x 37.24 *Å*) y en una celda cúbica (lado igual a 18.62 *Å*)[32]. Los modelos clásicos empleados en ese estudio, por ejemplo, el potencial rígido de interacción agua-agua fue el desarrollado por Rahman y Stillinger(ST2)[33] y, la interacción agua-superficie consistía en una variante de Lennard-Jones (LJ) como la ecuación 1, donde *Z* es la distancia de cada átomo de la molécula de agua al plano.

$$U(Z) = \frac{A}{Z^9} - \frac{B}{Z^3} \qquad (1)$$

La caracterización molecular consistió en obtener perfiles de densidad, perfiles de número de enlaces de hidrógeno y la distribución de orientaciones de las aguas con respecto a las superficies. Estos análisis, fueron construidos dividiendo la celda de

simulación en bloques de tamaño igual a 0.372 *Å* en dirección ***Z***; cuyo valor es mucho menor al diámetro de una molécula de agua (2.76 *Å*[34]). De acuerdo con lo reportado, obtuvieron perfiles de densidad similares en los dos tipos de celdas de simulación. Sin embargo, las oscilaciones en el perfil del sistema rectangular deberían ser más pequeñas que en el sistema cúbico, porque la distancia entre los planos es más larga y el efecto de ambas superficies no debería traslaparse. Es importante notar que, por mucho tiempo se utilizaron 256 moléculas para simular el bulto; ya que se creía que era el mínimo número de moléculas necesarias para que una molécula de agua tenga un número efectivo de vecinos en las tres direcciones. Ese número es pequeño considerando las características de simulación empleadas, por ejemplo, el radio de corte de las interacciones intermoleculares.

En esa misma década, Spohr propuso una metodología nueva para estudiar la interfase agua-metal, empleó un bloque de platino colocado en los extremos de la celda de simulación; cuya longitud fue de 10 *Å* en la dirección ***Z*** y estaba constituido por átomos explícitos[35]. Además, utilizó una celda de simulación alargada en el eje ***Z*** de 20 *Å*. El potencial de interacción agua-platino, de carácter hidrofílico, el cual consta de un potencial de LJ 12-6 más un potencial coulómbico; que considera las cargas imagen generadas por las cargas de las moléculas de agua. La restricción en el potencial de LJ es que solo interactúa el oxígeno del agua con los átomos de platino, los parámetros (σ y ε) fueron obtenidos de la regla de combinación de Kong[36] del potencial de LJ de dos átomos de oxígeno, tomados de los modelos rígidos de agua: TIP3P y TIP4P creados por Jorgensen *et al*[37]; la interacción agua-agua se simuló con el modelo de agua rígido ST2 y el potencial de interacción Pt-Pt fue el propuesto por Pound[38]. Años más tarde, se ajustó el potencial de interacción por pares a un conjunto de funciones exponenciales en las que ya se considera la interacción hidrógeno-metal [39]. Esta interacción se calculó como la suma del potencial oxígeno-metal más el potencial hidrógeno-metal y fue el primero en utilizar un modelo de agua flexible (Bopp-Jancsó-Heinzinger)[40]. El avance en ese estudio fue simular explícitamente la fase sólida, se confinaron las aguas de tal manera que, el radio de corte utilizado (9.8 *Å*) no

permitiese que aguas de la celda central interaccionaran con aguas en la celda imagen, como sucedió en lo reportado por el grupo de Rossky. Los nuevos análisis incluyeron funciones de distribución radial por regiones. En ellas se observa que la estructura de agua se ve afectada en los segundos vecinos a medida que se aproxima a la fase sólida. Este efecto disminuye, es decir, de tal forma que se recupera el segundo pico de la gO-O(r) a distancias menores a 5 *Å* del centro de la celda. Por un lado, en este tipo de sistemas discretos, el perfil de densidad se comporta como el reportado por Lee, pero se pierde la simetría a medida que se acerca al metal. Por el otro, el tiempo de simulación sigue siendo la mayor limitación, al aumentar el número de átomos solo obtiene dinámicas de apenas un par de decenas de *ps*, a diferencia del modelo continuo de Lee donde simuló más de 70 *ps*.

Hautman *et al.,*[41] propusieron simular las superficies hidrofóbicas con una modificación al modelo propuesto por Peter Rossky; el nuevo modelo considera tener cargas eléctricas. Estas cargas generan un nuevo potencial que se suma al considerado en el caso anterior. El resultado más interesante que obtienen es el cálculo de la capacitancia cerca de la superficie; con ello calculan valores de la constante dieléctrica seis veces menor que en agua líquida (el modelo de agua utilizado fue cargas puntuales simples, SPC por sus siglas en inglés[42]) y, que es lo esperado cuando los dipolos están alineados cerca de la superficie ($\epsilon \approx 12$) a diferencia del desorden que puede existir en el seno del líquido, ya que cerca del electrodo las fluctuaciones orientacionales son pequeñas a diferencia de lo que ocurre en el bulto.

A finales del siglo pasado, Spohr calculó trayectorias por dinámica molecular considerando una superficie infinita, hidrofílica y con cargas eléctricas. Además, crea una disolución electrolítica 2.2 molal de NaCl, con lo que es posible llamar a la interfase con la superficie como doble capa electroquímica (DCE)[43]. Los resultados obtenidos son interesantes porque se analiza la distribución de orientaciones de las aguas con respecto a cada superficie cargada positiva o negativamente, y es clara la tendencia de la orientación dependiente de la naturaleza eléctrica de la pared. Aunque

fue el inicio de protocolos de simulación para el estudio de la DCE, todavía requieren mejorar la descripción de la física de los diferentes componentes que la integran. Sin embargo, aún no existe un modelo clásico capaz de reproducir, por ejemplo, la correcta interacción de iones con agua[44–46]; esto se debe a que algunos modelos solo consideran la contribución de cargas puntuales, a pesar de que una descripción completa debería incluir un desarrollo multipolar, donde el inconveniente es que se vuelve costoso computacionalmente.

De acuerdo con Jedlovzky *et al.,* [27,47] es posible pensar una fase sólida constituida por moléculas hidrofóbicas como un soluto esférico de radio infinito, hacerlo de esta manera reduce el problema a tener una superficie plana infinita que está en contacto con el agua, es decir, es la representación más sencilla de un modelo de electrodo, un capacitor o la superficie de una membrana en contacto con una disolución electrolítica.

Trabajos recientes en los que se han construido mejores modelos clásicos para agua [21,48–51], iones y metales, ofrecen una nueva oportunidad de caracterizar la DCE no solo con los perfiles de orientaciones o análisis termoquímicos, sino también, con análisis dinámicos cerca de la interfase, como son los trabajos de Laage quien es pionero en análisis de dinámica orientacional en interfases. Ejemplo de las interfases que han estudiado son: líquido-aire, líquido-membranas o hidratación de proteínas[52–54].

La hidratación de superficies se ha estudiado desde hace tres décadas en las que los modelos de agua y la capacidad de cómputo han evolucionado notoriamente. Los campos de fuerza de agua han sido re-parametrizados; algunos modelos de agua tienen el objetivo de reproducir una propiedad experimental en específico o un conjunto pequeño de propiedades. Por ejemplo, el modelo clásico de agua TIP4P/Ice[55] fue ajustado para reproducir algunas líneas del diagrama de fases del hielo y líquido; o el modelo de cargas puntuales simples extendido (SPC/E), usado en el estudio de interfases sólido-líquido, que reproduce tanto la constante dieléctrica estática como el coeficiente de auto-difusión con valores cercanos al experimental[20,56]. Sin embargo,

este último potencial de agua no es polarizable y, por lo tanto, no responde a el efecto de un campo eléctrico producido por una superficie cargada.

El propósito de avanzar en la comprensión de los procesos que favorecen o inhiben los fenómenos en la doble capa electroquímica requiere de un *modelo realista* y robusto de la interfase electrodo metálico-disolución y, hasta el momento no se cuenta con alguno. Caracterizar este tipo de sistemas con detalle molecular nos ayudará a entender el rol de cada uno de los componentes, superficie, disolvente, electrolito y solutos, desempeña en un proceso de interfase.

Desarrollar progresivamente la metodología, para estudiar el efecto de superficies sin estructura con características diferentes de interacción, necesita de modelos de agua capaces de responder al efecto producido por una pared, por ejemplo, el potencial AMOEBA responde al ambiente permitiendo la inducción de dipolos en los centros atómicos (se discutirá en la sección 3.2.2) así, estudiar el cambio en propiedades moleculares colectivas, permite comparar con la respuesta de modelos no polarizables. De esta manera, se descarta la influencia de la fase sólida sobre propiedades específicas del disolvente y se atribuyen a las características intrínsecas del campo de fuerzas.

Por lo tanto, en este proyecto de maestría se plantean los siguientes objetivos para iniciar el estudio de soluciones acuosas ante superficies que no incluyen átomos explícitamente.

2.1 OBJETIVOS

2.1.1 Objetivo general

Este proyecto de investigación busca generar la metodología necesaria para el estudio de la interfase agua líquida-superficie a través de simulaciones numéricas de dinámica molecular y analizar las trayectorias dinámicas del sistema en sus aspectos estructurales y dinámicos.

2.1.2 Objetivos específicos

- Simular la hidratación de la fase sólida con una pared sin estructura y estudiar el efecto de esta sobre el disolvente cuando la superficie es hidrofóbica, hidrofílica o está cargada eléctricamente.
- Caracterizar con detalle molecular el comportamiento del agua ante la superficie sin estructura y conocer las diferencias con respecto al bulto.

2.1.3 Metas

- Seleccionar potenciales clásicos, para el disolvente y los tres tipos de superficie, hidrofóbica, hidrofílica y cargada eléctricamente, que permitan caracterizar el comportamiento del agua en la interfase líquido-superficie empleando para ello, modelos con diferentes grados de libertad y refinamiento en el tratamiento de las interacciones.
- Preparar celdas de simulación en condiciones que coincidan con las superficies sin estructura.
- Calcular trayectorias de dinámica molecular que permitan analizar el comportamiento del disolvente ante superficies de diferente naturaleza.
- Calcular propiedades estructurales y dinámicas de las trayectorias de dinámica molecular.

2.2 HIPÓTESIS

El estudio de la interfase sólido-agua líquida, por dinámica molecular, requiere de potenciales de interacción capaces de reproducir propiedades medibles experimentalmente. Las características del modelo de disolvente seleccionado son determinantes en la evaluación de la respuesta ante una superficie sólida sin estructura; la cual no responde a estímulos generados por la fase contraria. Entender el efecto que ejerce la superficie sobre modelos de agua con diferencias estructurales y aproximaciones clásicas a la densidad electrónica, ayuda a caracterizar, únicamente, la respuesta del disolvente y el alcance del efecto de la pared, sobre diferente número de capas de vecinos.

CAPÍTULO TRES

METODOLOGÍA

3. METODOLOGÍA

3.1 Dinámica Molecular

En fisicoquímica, la naturaleza puede analizarse desde dos puntos de vista: a un nivel macroscópico o a un nivel molecular. Las explicaciones científicas dadas a niveles moleculares están ligadas íntimamente con las explicaciones macroscópicas, para conectar estos dos mundos, la termodinámica estadística es el área de la fisicoquímica que lo hace [57].

Se sabe que la mecánica cuántica, que ofrece la visión molecular de un fenómeno, utiliza como objeto de estudio los electrones y núcleos que forman el sistema. Por el contrario, la termodinámica clásica trabaja con cantidades molares. La conexión entre estas dos visiones puede hacerse con simulaciones numéricas usando, por ejemplo, modelos construidos con información de mecánica cuántica y con variables de la termodinámica clásica, que bajo un esquema de simulación numérica generen conjuntos estadísticamente significantes; útiles para reproducir las propiedades ***promedio*** del sistema de acuerdo a la función de partición del sistema, como lo establece la termodinámica estadística[58].

La DM es una simulación numérica determinista que analiza el comportamiento en conjunto de un sistema molecular y se rige por las leyes deterministas de Newton, que son dependientes del tiempo[59,60]. Por el contrario, MC es una simulación numérica estocástica en donde cada configuración depende solo de la configuración anterior, ya que MC genera configuraciones aleatoriamente y en la modalidad de Metropolis, utiliza un criterio de probabilidad para aceptar o rechazar la nueva configuración[61].

3.1.1 Ensambles

El estado termodinámico de un sistema clásico, llamado *macroestado*, depende de variables naturales de tipo extensivas como número de átomos (***N***), volumen (***V***) y la energía de esa cantidad de partículas y espacio ocupado (***E***). Un *macroestado* tiene

asociado un conjunto estadístico de sistemas temporales igualmente probables llamados *microestados*. En un tiempo lo suficientemente largo, el sistema pasa de un *microestado* a otro y el comportamiento promedio del número de *microestados* visitados representa el *macroestado* del sistema de estudio. El conjunto de microestados del sistema recibe el nombre de ***ensamble***[62].

Al simular un *macroestado* con ***N*** átomos se requiere definir el *microestado* temporal del sistema, el cual se representa con coordenadas instantáneas: ***3N*** coordenadas de posición y ***3N*** coordenadas de momento de todas las partículas que forman el sistema. El conjunto de coordenadas representa un punto de un espacio de ***6N*** dimensiones, este espacio recibe el nombre de espacio de fases. El *ensamble* será un conjunto de puntos en el espacio de fases. La DM genera estos puntos y los conecta en el tiempo, entre más puntos o réplicas del sistema hay, el muestreo del espacio fase será más amplio y con mayor valor estadístico [63].

En termodinámica clásica podemos clasificar al sistema de estudio de tres formas distintas: abierto, cerrado y aislado. En termodinámica estadística se utilizan ensambles equivalentes: los sistemas abierto, cerrado y aislado corresponden al ensamble gran canónico (***VTμ***), ensamble canónico (***NVT***) y ensamble microcanónico (***NVE***), respectivamente [64].

Por lo general, la DM utiliza un ensamble *microcanónico* en el que un número ***N*** de partículas, el volumen y la energía total de las partículas permanece constante durante la trayectoria. Este ensamble es el más importante para validar que, tanto la energía como las fuerzas estén implementadas correctamente en los códigos de simulación. En DM se utilizan celdas de simulación, que son la región del espacio donde localizamos los átomos y el volumen de la celda es igual al volumen ocupado por las partículas. Un sistema termodinámico tiene un valor “preciso” de energía total; con la que se genera una trayectoria en el espacio de fases restringida por una híper-superficie. Cada *microestado* con ***N*** átomos tiene asociado un rango de energía total,

la cual permitirá generar una trayectoria restringida en una híper-capa de la híper-superficie. La energía total permanece constante, pero las contribuciones de energía cinética y potencial tendrán fluctuaciones porque se recorre la híper-capa en un rango de energía específico.

El ensamble canónico tiene dos características similares con el ensamble microcanónico, número de partículas y volumen constante; la diferencia es que se mantiene la temperatura promedio constante durante la trayectoria de simulación. La manera de poder hacer esto es igual que en un experimento de laboratorio, usando un termostato, en DM los termostatos son numéricos.

La energía cinética *promedio* de las partículas es la que ésta ligada con la temperatura instantánea (T_k) del sistema mediante la siguiente ecuación:

$$T(t) = \sum_{i=1}^{N} \frac{m_i v_i^2(t)}{k_B N} \tag{2}$$

Donde m_i es la masa de la partícula i, k_B es la constante de Boltzmann y V_i es la velocidad de cada partícula[60,65]. La forma en que el termostato actúa sobre el sistema de interés es mediante colisiones aleatorias (método propuesto por Andersen). Se considera que el sistema está inmerso en un reservorio de calor, los átomos experimentan colisiones con el reservorio, por lo tanto, se cambian las velocidades que origina un valor de T_k. Esto se traduce en mantener una distribución de velocidades tipo Boltzmann asociada a la temperatura del macroestado ($T = \langle T_k \rangle$), la T_k fluctúa durante la trayectoria con varianza relativa de $\frac{2}{3N}$[60].

3.1.2. Ecuaciones de movimiento y algoritmo de integración

La DM genera una trayectoria de réplicas del sistema conectadas en el tiempo. En cada punto de la trayectoria especifica la posición, velocidad y aceleración de cada

átomo. El desplazamiento de las partículas del sistema obedece las leyes clásicas del movimiento, las leyes de Newton. Con ellas se calculan las posiciones, velocidades y aceleraciones de cada partícula en cada instante de la trayectoria.

La DM se inicia con la solución de la segunda ley de movimiento, que es una ecuación diferencial de segundo orden[64,66]:

$$\boldsymbol{F_i} = m_i \frac{d^2 \boldsymbol{r_i}}{dt^2} = m_i \boldsymbol{a}_i \tag{3}$$

La masa m_i es constante para cada tipo de partículas; la fuerza se calcula a partir del gradiente de la energía potencial que solo depende de las posiciones atómicas[67] (ver ecuación 4).

$$\boldsymbol{F_i}(\boldsymbol{r}_{1,}\boldsymbol{r}_{2,}\cdots,\boldsymbol{r}_N) = \nabla_i U(\boldsymbol{r}_{1,}\boldsymbol{r}_{2,}\cdots,\boldsymbol{r}_N) = \frac{\partial U(\boldsymbol{r}_{1,}\boldsymbol{r}_{2,}\cdots,\boldsymbol{r}_N)}{\partial \boldsymbol{r}_i} \tag{4}$$

Una solución práctica de la ecuación diferencial se ha hecho por medio de la aproximación de diferencia finita, es decir, los elementos diferenciales $\left(\frac{d\boldsymbol{r}}{dt}\right)$ se aproximan a deltas $\left(\frac{\Delta \boldsymbol{r}}{\Delta t}\right)$. La ecuación se resuelve paso a paso, procurando obtener con suficiente grado de exactitud las posiciones, velocidades y aceleraciones en un tiempo $t + \Delta t$. Por lo tanto, Δt, debe ser lo suficientemente pequeño para reproducir el movimiento más rápido que sucede en el sistema, por ejemplo, la vibración del enlace O-H la cual es del orden de femtosegundos[63].

La solución discreta de las ecuaciones de movimiento se hace empleando algoritmos que calculan posiciones, velocidades y aceleraciones para $\boldsymbol{N}$ partículas del sistema. Comúnmente se utiliza el algoritmo de Velocidades de Verlet (AVV)[64], que se explica a continuación.

- Algoritmo de Velocidades de Verlet

Este algoritmo calcula posiciones y aceleraciones en un paso con tamaño igual a Δt y las velocidades en dos pasos, tomando para cada uno la mitad de Δt. Por ejemplo, en el tiempo $t = 0$, se dan posiciones y velocidades para todos los átomos. Con las posiciones se calculan las fuerzas asignadas a cada átomo utilizando la ecuación 4, a partir de esto, se calculan las aceleraciones de todas las partículas utilizando la expresión 3. En este punto inicial, se conocen los valores de las variables ($\boldsymbol{r, v, a}\ y\ \boldsymbol{F}$) para poder integrar las ecuaciones de movimiento.

El AVV establece que primero se calculan las posiciones en el tiempo $t + \Delta t$ con la ecuación 5 y con la expresión 6 se calculan las velocidades en la mitad del tamaño de paso y conociendo las posiciones en el paso $t + \Delta t$, se calculan las fuerzas con la ecuación 4 y, con ello se conocen las nuevas aceleraciones para el mismo tiempo, empleando la segunda ley de movimiento (ecuación 3).

$$\boldsymbol{r_i}(t + \Delta t) = \boldsymbol{r_i}(\boldsymbol{t}) + \Delta t \boldsymbol{v_i}(t) + \frac{1}{2}\Delta t^2 \boldsymbol{a_i}(t) \tag{5}$$

$$\boldsymbol{v_i}\left(t + \frac{\Delta t}{2}\right) = \boldsymbol{v_i}(\boldsymbol{t}) + \frac{1}{2}\Delta t \boldsymbol{a_i}(t) \tag{6}$$

Hecho lo anterior, solo faltaría calcular las velocidades para el tamaño de paso completo, que se logra utilizando la ecuación 7, porque es necesario conocer antes la aceleración en el tiempo $t + \Delta t$, que se calculó en el paso anterior.

$$\boldsymbol{v_i}(t + \Delta t) = \boldsymbol{v_i}\left(t + \frac{\Delta t}{2}\right) + \frac{1}{2}\Delta t \boldsymbol{a_i}(t + \Delta t) \tag{7}$$

Ahora, se conocen los valores de las posiciones, aceleraciones y velocidades a un paso completo después del tiempo t. Entonces, estas cantidades serán los valores nuevos de inicio y se repite el AVV hasta el tiempo requerido de simulación.

- Algoritmo *Rattle* y *Shake*.

Este algoritmo involucra restricciones de tipo holonómicas, es decir, únicamente aquellas que dependen de las posiciones y no de las velocidades[64,67,68]. Por ejemplo, para la restricción k-ésima, que depende de las posiciones de los átomos involucrados en dicha restricción, estará definida de la siguiente manera:

$$\sigma_k(\boldsymbol{r}_1, \cdots, \boldsymbol{r}_N) = 0 \qquad k = 1, \cdots, N_{restricciones} \tag{8}$$

Así, la segunda ley de movimiento, ecuación 3, está afectada por la suma del producto de una constante (λ_k) por el gradiente de cada restricción que involucra al átomo i, como en la ecuación 9.

$$m_i \boldsymbol{a}_i = \boldsymbol{F_i} + \sum_{k=1}^{N_r} \lambda_k \nabla_i \sigma_k \tag{9}$$

El problema del segundo factor de la ecuación anterior es que el conjunto de factores λ_k, son multiplicadores indeterminados de Lagrange y las ecuaciones del AVV están incompletas. Por ejemplo, la ecuación 5 se transforma en la ecuación 10, donde $\boldsymbol{r_i'}$ corresponde a la posición calculada como en el AVV.

$$\boldsymbol{r_i}(t + \Delta t) = \boldsymbol{r_i'} + \sum_{k=1}^{N_r} \frac{\Delta t^2}{2m_i} \lambda_k \nabla_i \sigma_k \tag{10}$$

Por ejemplo, para una molécula diatómica con una longitud de enlace constante e igual a a solo hay una restricción ($\sigma_1 = \sigma$) y debe satisfacer la igualdad $\sigma = a^2 - \left\|\boldsymbol{r}_j - \boldsymbol{r}_i\right\|^2 = 0$. Por lo tanto, solo hay un multiplicador indeterminado y de la condición $\sigma = 0$ se obtiene la siguiente ecuación cuadrática.

$$\left(\frac{\boldsymbol{r}'_{ji}\Delta t}{m_{red,ij}}\right)^2 \lambda^2 - \frac{2\left(\boldsymbol{r}'_{ji}\right)^2(\Delta t)^2}{m_{red,ij}}\lambda - \left(\boldsymbol{r}'_{ji}\right)^2 + a^2 = 0 \tag{11}$$

Con la solución de la ecuación anterior, se obtiene el multiplicador λ y, por ende, la posición de los átomos de la molécula diatómica en el tiempo $t + \Delta t$. A este algoritmo iterativo se le conoce como *Shake* (AS). Se evalúa la distancia y si la diferencia con la restricción es menor a un valor umbral, usualmente 1×10^{-8}, se acepta. Si no, se utilizan las nuevas posiciones como posiciones obtenidas con el AVV y se lleva a cabo el AS hasta su convergencia.

Después de obtener las nuevas posiciones con el AS, se calculan las fuerzas en el tiempo $t + \Delta t$ y, las velocidades en la mitad del paso se calculan utilizando los multiplicadores encontrados, ver ecuación 12.

$$\boldsymbol{v_i}(t + \frac{\Delta t}{2}) = \boldsymbol{v'_i} + \sum_{k=1}^{N_r} \frac{\Delta t}{2m_i}\lambda_k \nabla_i \sigma_k \tag{12}$$

El algoritmo *Rattle* (AR) afecta el cálculo de las velocidades en el paso completo, es iterativo porque es necesario satisfacer la condición 13 y, por lo tanto, se debe encontrar un nuevo conjunto de multiplicadores indeterminados de Lagrange (μ_k), para obtener las velocidades finales aplicando la ecuación 14.

$$\sum_{i=1}^{N} \nabla_i \sigma_k(t + \Delta t) \cdot \boldsymbol{v}_i(t + \Delta t) = 0 \tag{13}$$

$$\boldsymbol{v_i}(t + \Delta t) = \boldsymbol{v_i}\left(t + \frac{\Delta t}{2}\right) + \frac{\Delta t}{2m_i}\boldsymbol{F}_i(\Delta t) + \frac{\Delta t}{2m_i}\sum_{k=1}^{N_r} \mu_k \nabla_i \sigma_k(\Delta t) \tag{14}$$

Con la condición 13, en el caso de la molécula diatómica, se obtiene una expresión lineal con la forma de la ecuación 15, que depende de las posiciones en el paso completo y de los dos primeros términos de la ecuación 14.

$$\mu_1 = \frac{m_{red,ij}\left[\boldsymbol{r}_{ji}(t+\Delta t)\cdot \boldsymbol{v}'_{\boldsymbol{j}} - \boldsymbol{r}_{ji}(t+\Delta t)\cdot \boldsymbol{v}'_{\boldsymbol{i}}\right]}{\Delta \boldsymbol{t}\left[\boldsymbol{r}_{ji}(t+\Delta t)\right]^{\mathbf{2}}} \tag{15}$$

El AVV modificado, algoritmo *Rattle* y *Shake*, se utiliza en cada paso de integración y debe satisfacer las condiciones de cada restricción. Por lo tanto, es un algoritmo más caro, porque en cada paso se calculan los dos conjuntos de multiplicadores indeterminados en forma iterativa y auto-consistente.

En la ecuación 4, se hace referencia que la fuerza se calcula a partir de una función conocida como *función de energía potencial*, esa expresión indica la forma en que interaccionan los átomos del sistema de estudio y su valor en un instante dado depende de las posiciones como se explica en la siguiente sección.

3.1.3 Potenciales clásicos

En DM el cálculo de la energía potencial es la rutina que consume el mayor tiempo de cómputo. Es considerada como la parte más importante en una simulación, ya que toda la física del sistema de estudio se encuentra en la forma en la que esta energía es modelada. Estudiar la naturaleza con un modelo cuántico requiere de la ecuación de Shrödinger, porque su solución contiene toda la información del macroestado (ver ecuación 16).

$$H\psi(R,r) = E\psi(R,r) \tag{16}$$

En la expresión 16, R corresponde a las coordenadas nucleares y r al conjunto de coordenadas electrónicas. En sistemas con más de un electrón, la ecuación 16 solo

tiene solución utilizando la aproximación adiabática en la que se separan los movimientos acoplados entre núcleos y electrones, porque la contribución cinética del movimiento de un electrón al hamiltoniano del sistema es 1836 veces más grande que la de un protón. Entonces, se considera que los núcleos están en reposo y la energía asociada a los electrones es una función de las distancias entre núcleos, es decir, representa la energía potencial para el movimiento relativo nuclear[69].

Una de las aproximaciones adiabáticas más usada es la propuesta por Born-Oppenheimer, la cual propone que, en la función de onda del sistema, los estados electrónicos corresponden a un conjunto de estados nucleares, por lo que la energía potencial recibe el nombre de *potencial de interacción molecular* y con ello, es posible calcular la energía como función de las posiciones de los núcleos atómicos.

En mecánica molecular, la energía potencial es descrita con modelos representados por funciones analíticas cuya evaluación que requieren menor costo computacional que los modelos cuánticos y, que son capaces de reproducir la energía del grado de libertad de interés cerca de la región de equilibrio, porque están parametrizadas para ello. Entre los modelos de potencial más usados se encuentran: el potencial armónico, el de Lennard-Jones, el de Coulomb, el de torsión, entre otros. La forma analítica, para describir las interacciones, se ajusta a la suma de potenciales intra- e intermoleculares que describen la física de un sistema y la ecuación 17 es un ejemplo de ello[61,63,70]:

$$V = V_{enlace} + V_{ángulo} + V_{diedros} + V_{vdW} + V_{electrostático} \qquad (17)$$

3.1.3.1 Interacciones de enlace

Los primeros dos términos de la ecuación 17 corresponden a las interacciones intramoleculares, su evaluación da como resultado la energía necesaria para estirar o comprimir un enlace y acercar o alejar dos enlaces con un átomo en común. Únicamente se describirán estas contribuciones ya que la especie molecular, de

interés en el proyecto, no tienen más contribuciones a la interacción intramolecular, por ejemplo, la energía asociada a un ángulo diedro.

- Estiramiento de enlace

En mecánica molecular, el comportamiento de una vibración de enlace se puede describir con un potencial armónico (expresión semejante a la ley de Hooke). Al considerarse el estiramiento del enlace como un movimiento elástico se necesitan: un valor mínimo de enlace, l_o, y una constante de estiramiento, k_l, (*ver ecuación* 18) que son específicos para cada tipo de enlace y para cada especie atómica.

$$V_{enlace,ij} = \frac{1}{2}k_l\left(l_{ij} - l_o\right)^2 \tag{18}$$

En una dimensión (por ejemplo x), la ecuación 18 produce una fuerza de magnitud igual a la expresión 19; las coordenadas de los pares i y j, generan una distancia l, tal que $l_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}$. La fuerza depende del potencial y este a su vez depende de la distancia l. Obtener la componente de la fuerza para la partícula i, requiere utilizar la regla de la cadena. El primer término de la expresión 19 corresponde a la derivada del potencial con respecto a la distancia y, el segundo a la derivada de la distancia con respecto a la componente de interés, por ejemplo, x. La siguiente expresión es la misma para calcular la fuerza de la partícula j en la misma componente, pero con signo contrario. Entonces, solo se necesita calcular una vez por cada componente y par atómico, por ejemplo, $F_{x,j} = -F_{x,i}$.

$$F_{x,i} = -\frac{\partial V_{enlace,ij}}{\partial l_{ij}}\frac{\partial l_{ij}}{\partial x_i} = k_l(l_{ij} - l_o)\frac{x_j - x_i}{l_{ij}} \tag{19}$$

La expresión anterior funciona para las componentes restantes, y y z. Existen otras expresiones para describir el enlace de dos átomos que pueden reproducir las características anarmónicas del mismo, por ejemplo, el potencial de Morse[69].

- Ángulo de enlace

La expresión 20 se utiliza en el cálculo de la energía necesaria para acercar o alejar dos enlaces que comparten un átomo, θ_{ijk}. La interacción es proporcional al cuadrado de la diferencia entre el ángulo formado y el ángulo de mínima energía. Al igual que en la ecuación 18, la constante de proporcionalidad y el valor mínimo del ángulo son específicos para cada tipo de triada de átomos.

$$V_{ángulo} = \frac{1}{2} k_\theta \left(\theta_{ijk} - \theta_o\right)^2 \tag{20}$$

La fuerza calculada de este potencial es más complicada que la anterior, porque el ángulo depende de los vectores de posición de las partículas i, j y k. La ecuación 21 relaciona el coseno del ángulo con los vectores de posición, donde $\boldsymbol{r}_{ij} = \boldsymbol{r}_i - \boldsymbol{r}_j$.

$$\cos(\theta_{ijk}) = \frac{\boldsymbol{r}_{ij} \cdot \boldsymbol{r}_{kj}}{\|\boldsymbol{r}_{ij}\| \|\boldsymbol{r}_{kj}\|} \tag{21}$$

La fuerza se obtiene con la expresión 22 en que el potencial se deriva respecto al ángulo y para obtener la derivada del ángulo con respecto a las componentes, es necesario derivar el coseno con la regla de la cadena como se muestra en la ecuación 23.

$$F_{x,i} = -\frac{\partial V_{ángulo,ijk}}{\partial \theta_{ijk}} \frac{\partial \theta_{ijk}}{\partial x_i} \tag{22}$$

$$F_{x,i} = -\frac{\partial V_{ángulo,ijk}}{\partial \theta_{ijk}} \frac{\partial \cos\left(\theta_{ijk}\right)}{\partial x_i} = \frac{\partial V_{ángulo,ijk}}{\partial \theta_{ijk}} \frac{\sin(\theta_{ijk})\, \partial \theta_{ijk}}{\partial x_i} \tag{23}$$

La componente x de la fuerza para las partículas i, j y k tienen la forma como las ecuaciones 24, 26 y 25, respectivamente.

$$F_{x,i} = k_\theta(\theta_{ijk} - \theta_o)\frac{1}{\sin(\theta_{ijk})}\left(\frac{x_j - x_k}{\|\boldsymbol{r}_{ij}\|\|\boldsymbol{r}_{kj}\|} - \cos(\theta_{ijk})\frac{x_i - x_j}{\|\boldsymbol{r}_{ij}\|^2}\right) \quad (24)$$

$$F_{x,k} = k_\theta(\theta_{ijk} - \theta_o)\frac{1}{\sin(\theta_{ijk})}\left(\frac{x_j - x_k}{\|\boldsymbol{r}_{ij}\|\|\boldsymbol{r}_{kj}\|} - \cos(\theta_{ijk})\frac{x_j - x_k}{\|\boldsymbol{r}_{kj}\|^2}\right) \quad (25)$$

$$F_{x,j} = -F_{x,i} - F_{x,k} \quad (26)$$

3.1.3.2 Interacciones no enlazantes

Átomos de moléculas independientes interactúan entre sí mediante fuerzas no covalentes. Este tipo de interacciones son tratadas como función de la distancia y las más comunes son de dos tipos: 1) las interacciones de corto alcance o interacciones tipo van der Waals y 2) las interacciones electrostáticas o de largo alcance. Con ellas se intenta aproximar el comportamiento asociado a la interacción entre densidades electrónicas. Dependiendo el campo de fuerzas, existen diferentes formas para hacerlo (ver la sección 3.2.2 y la sección 3.2.3).

- Interacción electrostática

El modelo más simple de potencial electrostático requiere de cargas atómicas puntuales y es el potencial de Coulomb. Dichas cargas son obtenidas por medio de cálculos cuánticos (MP2/6-311+G**) y son las que reproducen el potencial electrostático de la molécula o del fragmento de ella[71]. Un ejemplo de metodología es la propuesta por Mulliken, llamado análisis poblacional[59]. En la ecuación 27 se observa que el potencial electrostático depende del inverso de la distancia entre el átomo i y el j.

$$V_{electrostático} = \frac{1}{4\pi\varepsilon_o}\sum_{i=1}^{N_A}\sum_{j=1}^{N_B}\frac{q_i q_j}{r_{ij}} \quad (27)$$

Donde A y B son moléculas, N_A y N_B son los números de cargas puntuales, r_{ij} es la distancia entre las dos cargas y q_i y q_j son las cargas puntuales.

El inconveniente de este potencial es que decae muy lentamente con la distancia. La solución a este problema es la metodología propuesta por Ewald[60], donde se transforma una serie lenta a la suma de dos series que convergen rápidamente. Las sumas de Ewald están compuestas por la suma del espacio real (directo), el espacio recíproco (espacio de Fourier) y la auto-energía.

$$V_{electrostático} = V_{directo} + V_{recíproco} + V_{auto-energía} \tag{28}$$

Ewald propuso escribir el potencial original de Coulomb en términos de la función de error ($V_{reciproco}$), que converge rápidamente con la transformada de Fourier y, la función de error complementaria ($V_{directo}$) más la constante de auto-energía, como en la ecuación 29.

$$V_{Ewald} = \frac{1}{2}\sum_{i\neq j}^{N}\frac{q_i q_j erfc(\sqrt{\alpha}r_{ij})}{r_{ij}} + \frac{1}{2V}\sum_{\boldsymbol{k}\neq 0}^{N}\frac{4\pi}{k^2}\|p(\boldsymbol{k})\|^2 e^{\left(-\frac{k^2}{4\alpha}\right)} - \sqrt{\frac{\alpha}{\pi}}\sum_{i=1}^{N} q_i^2 \tag{29}$$

Por un lado, la ecuación anterior es una solución al cálculo del potencial electrostático de manera más exacta, pero el costo computacional crece como N^2, donde N es el número de átomos. Una forma de disminuir el tiempo de cómputo es la elección de un valor óptimo de α y con ello reducir a $N^{3/2}$. Sin embargo, un valor grande de α evita considerar en la suma directa a todas aquellas partículas que están más allá de un radio de corte (ver sección 3.1.4)[72].

Si se restringe el número de átomos considerados en la suma del espacio real, entonces la suma recíproca no se calcula directamente, es decir, las partículas cargadas se distribuyen en una malla tridimensional y la energía se aproxima a una

interpolación utilizando β-splines de quinto orden. El potencial se calcula con una convolución y se evalúa con la transformada rápida de Fourier, por lo tanto, el tiempo requerido se reduce a $N \log N$ y este método más barato computacionalmente, recibe el nombre de *Particle Mesh Ewald*, PME[73].

- Interacción de van der Waals

Las interacciones de corto alcance pueden calcularse mediante el potencial de Lennard-Jones 12-6 que tiene la siguiente forma:

$$V_{vdW} = 4\varepsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right] \tag{30}$$

En la ecuación anterior, σ es el diámetro de colisión donde la energía es igual a cero; ε es la profundidad de la curva de energía potencial y el 4 es la constante de proporcionalidad que se deriva de las potencias 12-6. La componente repulsiva está representada con el término de exponente 12 y la componente atractiva con el término de exponente 6. En un campo de fuerzas, se especifican los valores de sigma y épsilon para cada especie atómica, pero en especies diferentes, se utilizan reglas de combinación de parámetros. La regla más usada es la de Lorentz-Berthelot, así el valor de σ_{ij} está definido como la expresión 31 y el valor de ε_{ij} como la ecuación 32.

$$\sigma_{ij} = \frac{\sigma_i + \sigma_j}{2} \tag{31}$$

$$\varepsilon_{ij} = \sqrt{\varepsilon_i \varepsilon_j} \tag{32}$$

3.1.4 Condiciones periódicas a la frontera

En la simulación numérica un modelo realista necesita varios cientos de moléculas de disolvente. Eso lo convierte en un problema, porque la DM tendría un sistema de estudio infinito y la cantidad de memoria RAM necesaria para el cálculo supera la capacidad disponible. Los cálculos de la simulación se limitan a una región del espacio con dimensiones finitas, este espacio recibe el nombre de celda unitaria, funciona como una caja de simulación y generalmente tiene forma cúbica. La celda unitaria es muy parecida a la forma del sistema, así podemos encontrar celdas unitarias cúbicas, prismas hexagonales, primas cuadrangulares, octaedros, entre otras. La celda que contiene al sistema de interés se replica en todas las direcciones por sus propias imágenes (la primera vez son 26 réplicas en el espacio), entonces si alguna molécula abandona la celda será reemplazada por la molécula que abandona la celda en su imagen contraria, esto recibe el nombre de condiciones periódicas en la frontera, CPF (ver figura 1). Para evitar interacciones no permitidas entre moléculas durante la DM, es necesario considerar el concepto de *radio de corte*, esto es la distancia máxima en la cual un átomo en particular interactúa con átomos vecinos sin que lo haga con su propia imagen[60,61,64].

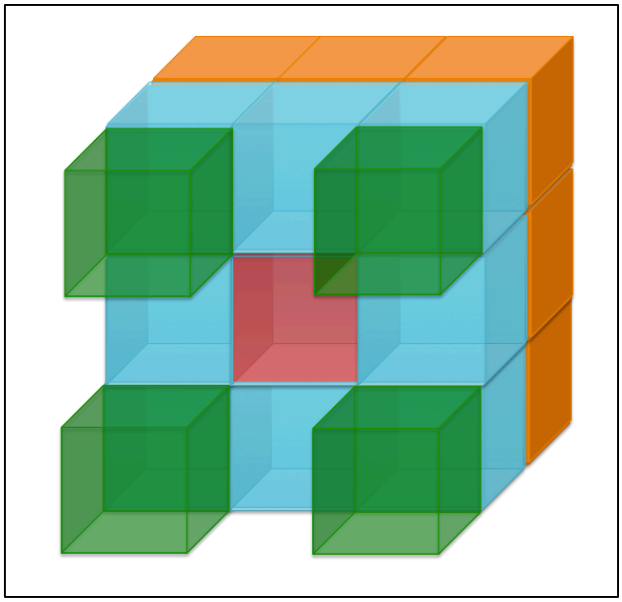

Figura 1 Condiciones periódicas a la frontera en 3 dimensiones, la celda unitaria está representada en color rojo y sus 26 imágenes en color verde, azul y naranja.

3.1.5 Etapas de la dinámica molecular

Los experimentos que se hacen en un laboratorio generalmente siguen los siguientes pasos: preparación de la muestra y equipo, llevar la muestra a las condiciones fisicoquímicas necesarias para hacer un buen análisis, desarrollar la técnica de experimentación, recolectar datos y obtener información del sistema. La DM necesita pasos análogos a un experimento: minimización, calentamiento, producción, equilibrio y análisis de la dinámica molecular[57,59,64,70].

- Minimización

La DM siempre requiere una geometría inicial del sistema de estudio y el arreglo de las moléculas, no siempre es el más favorable para el tipo de interacciones que dominan la física del macroestado. Por lo tanto, es necesario hacer una minimización de energía del sistema, donde se reacomoden las especies presentes para obtener un arreglo molecular de menor energía, es decir, que la geometría propuesta tenga propiedades instantáneas dentro del rango de las propiedades promedio.

- Calentamiento

Con la minimización del sistema, es tiempo de suministrar energía para que las moléculas comiencen a vibrar, rotar, trasladarse e interactuar a través del potencial utilizado. Una forma de proporcionar energía es asignando velocidades iniciales a cada átomo de la geometría inicial minimizada. Las velocidades pueden ser asignadas en dos formas: 1) todos los átomos inician con velocidad cero o 2), a cada átomo se le asigna una velocidad aleatoria que sigue una distribución tipo Maxwell-Boltzmann[67], como en la ecuación 33. Con la energía proporcionada al sistema, la propiedad temporal como la T_k, se acerca a las condiciones termodinámicas del ensamble.

$$f(\boldsymbol{v}) = \sqrt{\frac{m}{2\pi k_B T}}\, e^{-\frac{m\boldsymbol{v}^2}{2k_B T}} \tag{33}$$

- Producción

En esta etapa de la DM las propiedades temporales, descritas por una función de densidad $\varrho\ (q, p;\ t)$, presentan fluctuaciones que, conforme aumenta la longitud de la trayectoria, tienden a ser parte del ensamble estacionario, es decir, que el sistema pasará a una fase de equilibrio.

- Equilibrio

Es la etapa más importante de la simulación numérica, porque el comportamiento estadístico del conjunto de átomos refleja propiedades medibles experimentalmente. La fase de equilibrio depende del tiempo de relajación del sistema, es decir, la convergencia para cada propiedad tiene tiempos específicos. Algunos indicadores de una simulación en equilibrio son: que la $\langle T_k \rangle$ del sistema sea la del macroestado y que las propiedades estructurales proporcionen la correcta distribución de moléculas.

- Análisis de la DM

La etapa de análisis implica que la trayectoria de la DM ha finalizado y el requisito principal, pero no el más importante, es contar con información suficiente de que el sistema ha alcanzado el equilibrio en la simulación. Los datos guardados de la simulación (posiciones, velocidades, momentos dipolares, energía total, energía cinética, energía potencial, temperatura, etc.) deberán ser procesados para obtener información de tipo estructural o dinámica[64].

3.2 Modelos de Agua

Estudiar el comportamiento de agua en la interfase líquido-sólido requiere de plantear modelos que permitan entender la estructura de solvatación de las moléculas de agua cerca de la interfase. En el bulto una molécula de agua es doblemente aceptora y donadora de enlaces de hidrógeno, pero en la interfase esta característica se pierde porque el número de vecinos es menor en comparación con el bulto, es decir, la estructura centro-simétrica se rompe.

Dentro de las múltiples opciones de modelos de agua disponibles en la literatura, elegimos uno parametrizado para reproducir propiedades experimentales y con una descripción de molécula rígida. La otra opción corresponde a un modelo de molécula flexible en el que las propiedades electrostáticas se reproducen a través de un desarrollo multipolar y en el que los parámetros estructurales y electrostáticos fueron ajustados para reproducir datos cuánticos y propiedades de las tres fases.

La elección del potencial de agua a utilizar se hace entre dos modelos con características diferentes: por un lado, tenemos el modelo TIP4P-FB[50] que es rígido y no polarizable, el cual se evaluó contra otros potenciales que tienen características similares y se ha demostrado que está parametrizado para reproducir propiedades experimentales[74]. Por el otro, se escogió un modelo flexible y polarizable: el modelo iAMOEBA[48]; este potencial ha demostrado que reproduce propiedades de la fase líquida, sólida y gas, además del diagrama de fases del agua. Además, se utilizó el modelo AMOEBA[23,75] que, a diferencia de su versión económica (iAMOEBA), depende de una polarización mutua entre todas las moléculas (ver la sección 3.2.2). A continuación, son descritos los modelos de agua utilizados en este proyecto.

3.2.1 TIP4P-FB

El modelo de agua rígido de cuatro sitios, TIP4P desarrollado por Jorgensen *et al.*, considera cuatro partículas que interactúan: tres sitios representan a los átomos; dos átomos de hidrógeno, donde se colocan las cargas positivas, y el átomo de oxígeno, que es el único sitio de Lennard-Jones; el cuarto sitio (M) está en la bisectriz ($H-O-H$), cerca del centro de masas de la molécula, en éste se coloca la carga negativa[37]. Desde su publicación en 1983, se han generado muchas versiones de este modelo. Cada versión de este potencial responde a un conjunto de propiedades a reproducir. En el caso del campo de fuerzas TIP4P/Ew[68], se utiliza la técnica de sumas de Ewald para el cálculo de la energía potencial electrostática (ver sección 3.1.3.2), por lo tanto, los parámetros originales ya no funcionan para reproducir propiedades, porque con la modificación, la energía electrostática tiene contribuciones más allá del radio de corte.

En el potencial TIP4P-2005[76], las propiedades objetivo fueron la temperatura de máxima densidad y la estabilidad del polimorfismo del hielo.

En 2011 Vega *et al.* evaluaron diferentes versiones en su capacidad de reproducir 17 propiedades, encontrando que TIP4P-2005 obtenía una calificación de 7.2/10[77]. Ninguno de los modelos anteriores (TIP4P, TIP4P/Ew y TIP4P-2005) reproduce la constante dieléctrica estática, propiedad importante para simulaciones que incluyen biomoléculas o sistemas con iones[21].

En 2014, empleando el método de parametrización sistemática (*Force Balance*, FB[78]), se obtuvo un nuevo conjunto de parámetros que se conoce como TIP4P-FB[50]. En la parametrización se escogieron diferentes propiedades objetivo de datos experimentales[79] y teóricos. Seis propiedades de la fase líquida: densidad, entalpía de vaporización, coeficiente de expansión térmica, compresibilidad isotérmica, capacidad calorífica isobárica y la constante dieléctrica estática; la densidad del hielo y, energías y fuerzas provenientes de cálculos cuánticos. En el proceso de optimización con FB se utilizaron, como datos iniciales, los tres juegos de parámetros de los potenciales TIP4P, TIP4P/Ew y TIP4P-2005, y se hizo una optimización para cada modelo. Las tres optimizaciones convergieron al mismo conjunto de parámetros con los que, a diferencia de los potenciales iniciales, se obtiene un valor de la constante dieléctrica estática cercano al valor experimental.

Por otra parte, también en ese año, Fuentes *et al.*[80] emplearon un método de parametrización no sistemático en el que la constante dieléctrica fue una de las propiedades objetivo. Se obtuvieron parámetros muy semejantes a los obtenidos con FB, con diferencias hasta el tercer decimal. La decisión de ocupar TIP4P-FB en lugar de TIP4P/ε fue de orden práctico: ya estaba implementado en el programa para calcular trayectorias de DM (ver anexo 7.13).

En la tabla 1 y 2, para el modelo TIP4P-FB, se reportan los parámetros intra- e intermoleculares, respectivamente.

Distancia O-H [nm]	$\measuredangle\theta_{HOH}$ [Grados]	Distancia O-M $[nm]$	Cte. de enlace $k_l\ \left[\frac{KJ}{nm^2 mol}\right]$	Cte. de ángulo $k_\theta\ \left[\frac{KJ}{nm^2 mol}\right]$
0.9572	104.52	0.01052	462750.4	836.8

Tabla 1 Parámetros intramoleculares del modelo TIP4P-FB.

$q_O\ [e]$	$q_H\ [e]$	$q_M\ [e]$	$\sigma_O\ [nm]$	$\varepsilon_O\ \left[\frac{KJ}{mol}\right]$
0	0.5258	-1.0517	0.3165	0.7492

Tabla 2 Parámetros intermoleculares del modelo TIP4P-FB.

Los parámetros anteriores se utilizan para evaluar el potencial de interacción agua-agua considerando, en el potencial de L-J que el único par que contribuye a este término es el O – O.

3.2.2 AMOEBA

En 2003, Ren y Ponder propusieron un nuevo potencial clásico para agua, con éste se puede simular agua en diferentes ambientes, ya que el modelo es capaz de responder al campo eléctrico generado por el medio. El potencial recibe el nombre de AMOEBA (*Atomic Multipole Optimized Energetics for Biomolecular Applications*, por sus siglas en inglés)[23]. En el año 2015, se utilizó el método de optimización sistemática FB para mejorar el conjunto de parámetros del modelo[75]. Este es un modelo en el cual los grados de libertad estructurales de la molécula se consideran de manera explícita. Los 26 parámetros asociados a ello se ajustaron a resultados de cálculos *ab initio* de cúmulos de agua y datos experimentales de la fase líquida para un amplio rango de temperaturas. Uno de los atractivos de este modelo es el riguroso tratamiento del comportamiento electrostático de la molécula que se hace empleando un desarrollo multipolar. Además de asignar una carga puntual a cada centro atómico, se incorpora un dipolo, un cuadrupolo y la polarización se trata con dipolos atómicos inducidos en forma auto-consistente. Los parámetros de este desarrollo se ajustaron para reproducir los multipolos atómicos permanentes y la polarizabilidad dipolar atómica isotrópica.

AMOEBA consiste en potenciales anarmónicos para describir el enlace y el ángulo, a) y b), respectivamente; un término armónico para modelar el acoplamiento entre la geometría y las vibraciones, c); se utiliza el potencial de Halgren (*Buffered 14-7*) para modelar las interacciones de dispersión y repulsión, d)[81], y el potencial electrostático con un desarrollo de momentos multipolares permanentes, e)[82]. Los multipolos se obtuvieron a partir del análisis de multipolos distribuidos a un nivel MP2/aug-cc-pVTZ utilizando la geometría experimental de una molécula de agua en la fase gas. La expresión analítica para la interacción entre moléculas de agua es la siguiente:

a) $$V_{enlace} = k_l\left(l_{ij} - l_o\right)^2\left[1 - 2.55\left(l_{ij} - l_o\right) - 3.793125\left(l_{ij} - l_o\right)^2\right]$$

$$V_{\text{ángulo}} = k_\theta\left(\theta_{ijk} - \theta_o\right)^2\left[1\right.$$

b) $$- 0.014\left(\theta_{ijk} - \theta_o\right) + 5.5\times10^{-5}\left(\theta_{ijk} - \theta_o\right)^2$$

$$\left. - 7.0\times10^{-7}\left(\theta_{ijk} - \theta_o\right)^3 + 2.2\times10^{-8}\left(\theta_{ijk} - \theta_o\right)^4\right]$$

c) $$V_{Urey-Bradley} = k_b\left(b_{ij} - b_o\right)^2 \qquad (34)$$

d) $$V_{Buff} = \varepsilon_{ij}\left(\frac{1+\delta}{\frac{R_{ij}}{R_{ij}^0}+\delta}\right)^{n-m}\left(\frac{1+\gamma}{\left(\frac{R_{ij}}{R_{ij}^0}\right)^m+\gamma} - 2\right)$$

e) $$V_{multipolar} = \frac{1}{4\pi\varepsilon_0}\left(\frac{Q_{total}}{r} + \frac{\boldsymbol{\mu}_{total}\cdot\hat{\boldsymbol{r}}}{r^2} + \frac{\Theta}{r^3}\right)$$

En el *d*) de la expresión anterior, R_{ij}^0 es la distancia mínima para el par $i-j$; $n = 14$ y $m = 7$; $\delta = 0.07$ y $\gamma = 0.12$. Las reglas de combinación de los pares diferentes son como la ecuación 35 y 36, porque a diferencia del modelo TIP4P-FB, en el modelo AMOEBA los tres pares atómicos contribuyen al potencial de vdW.

$$R_{ij}^{0} = \frac{(R_{ii}^{0})^{3} + \left(R_{jj}^{0}\right)^{3}}{\left(R_{ii}^{0}\right)^{2} + \left(R_{jj}^{0}\right)^{2}} \tag{35}$$

$$\varepsilon_{ij} = \frac{4\varepsilon_{ii}\varepsilon_{jj}}{\left(\varepsilon_{ii}^{1/2} + \varepsilon_{jj}^{1/2}\right)^{2}} \tag{36}$$

- Polarización

La polarización se trata a través de la inducción mutua de los dipolos en cada átomo, es decir la generación de dipolos inducidos. Las polarizabilidades atómicas se ajustan a las moleculares mediante un modelo no aditivo, es decir, la ***polarización mutua*** se lleva a cabo entre todos los sitios atómicos, a diferencia del modelo aditivo, donde la contribución proviene de aquellos sitios que no pertenecen a una misma molécula, llamada ***polarización directa***.

El dipolo inducido total, que se menciona en el párrafo anterior, es proporcional al campo generado por el medio multiplicado por la constante de polarización, ver ecuación 37. El campo eléctrico total tiene dos contribuciones: 1) el campo generado por todos los momentos multipolares permanentes de los sitios de otras moléculas que, actúan en el átomo i y generan un dipolo inducido directo y, 2) la contribución de dipolos inducidos de todos los sitios, excepto el de interés, que generan dipolos mutuamente inducidos.

$$\boldsymbol{\mu}_{i}^{ind} = \alpha \boldsymbol{E}_{medio} = \alpha(\boldsymbol{E}_{directo} + \boldsymbol{E}_{mutuo}) \tag{37}$$

El cálculo de la polarización auto-consistente (dipolo mutuamente inducido) es iterativa y el criterio de convergencia consiste en que, los dipolos inducidos no induzcan otro más allá de 10^{-5} Debye.

A continuación, se reportan los parámetros inter- e intramoleculares del modelo AMOEBA:

Parámetro	Unidad	Valor
q_O	e	-0.51966
μ_z^O	e bohr	0.14279
Θ_{xx}^O	e bohr^2	0.37928
Θ_{yy}^O	e bohr^2	0.41809
Θ_{zz}^O	e bohr^2	0.03881
q_H	e	0.25983
μ_x^H	e bohr	-0.03859
μ_z^H	e bohr	-0.05818
Θ_{xx}^H	e bohr^2	-0.03673
Θ_{yy}^H	e bohr^2	-0.10739
Θ_{xz}^H	e bohr^2	-0.00203
Θ_{zz}^H	e bohr^2	0.14412
α_O	Å^3	0.837
α_H	Å^3	0.496
Factor de amortiguamiento	Å	0.39
R_O^0	Å	3.405
ε_O	kcal/mol	0.11
R_H^0	Å	2.655
ε_H	kcal/mol	0.0135
Factor de reducción H vdW	-	0.91
l_{O-H}^0	Å	0.9572
k_l^{O-H}	kcal/mol/Å^2	556.85
θ_{HOH}^0	Grados	108.5
k_θ^{HOH}	kcal/mol/rad^2	48.7
b_{H-H}^0	Å	1.5326
k_b^{H-H}	kcal/mol/Å^2	-7.6

Tabla 3 Parámetros de interacción del modelo AMOEBA-2013[83]

3.2.3 iAMOEBA

A fin de disminuir el costo computacional que un modelo como AMOEBA tiene, se buscó describir la aproximación a la densidad electrónica mediante una polarizabilidad directa, donde los dipolos inducidos son generados únicamente por los momentos multipolares permanentes y el par O – O es el único que contribuye al potencial de vdW. Esta versión, económica computacionalmente, del modelo AMOEBA se llamó *inexpensive AMOEBA* (***iAMOEBA*** por sus siglas en inglés)[48].

Los parámetros se obtuvieron con el método FB y a continuación se muestran en la siguiente tabla:

Parámetro	Unidad	Valor
q_O	e	-0.59402
μ_z^O	e bohr	0.08848
Θ_{xx}^O	e bohr^2	0.22618
Θ_{yy}^O	e bohr^2	-0.32244
Θ_{zz}^O	e bohr^2	0.09626
q_H	e	0.29701
μ_x^H	e bohr	-0.09391
μ_z^H	e bohr	-0.12560
Θ_{xx}^H	e bohr^2	0.18754
Θ_{yy}^H	e bohr^2	0.02174
Θ_{xz}^H	e bohr^2	-0.03635
Θ_{zz}^H	e bohr^2	-0.20928
α_O	Å^3	0.80636
α_H	Å^3	0.50484
Factor de amortiguamiento	Å	0.23616
R_O^0	Å	3.6453
ε_O	kcal/mol	0.19682
R_H^0	Å	0
ε_H	kcal/mol	0
Factor de reducción H vdW	-	0
l_{O-H}^0	Å	0.9584
k_l^{O-H}	kcal/mol/Å^2	557.63
θ_{HOH}^0	Grados	106.48
k_θ^{HOH}	kcal/mol/rad^2	49.9
b_{H-H}^0	Å	1.5357
k_b^{H-H}	kcal/mol/Å^2	-10.31

Tabla 4 Parámetros del potencial de interacción iAMOEBA.

Los tres modelos de agua están implementados en OpenMM y sus archivos de parámetros son el anexo 7.13 (TIP4P-FB), 7.14 (AMOEBA) y 7.15 (iAMOEBA).

3.3 Modelos de Superficie

De acuerdo con la metodología propuesta por Lee *et al.* de utilizar superficies continuas para simular la interacción agua-sólido[32] y, la forma funcional del potencial propuesto por Spohr[39]; se proponen los siguientes tres potenciales de interacción entre superficies sin estructura y el agua. Cada superficie se diferencia por tener carácter hidrofóbico, hidrofílico o cargada eléctricamente.

3.3.1 Superficie Hidrofóbica y Superficie Hidrofílica

La energía de adsorción de agua en Pt se obtuvo de datos experimentales[39] y se ajustó a funciones exponenciales que dependen de la coordenada y para reproducir la hidratación hidrofílica. La forma del potencial hidrofóbico es solo la parte repulsiva de la curva de la energía de adsorción.

El potencial agua-superficie, ver ecuación 38, viene de sumar el potencial oxígeno-superficie, V_{O-s}, más el hidrógeno–superficie, V_{H-s}:

$$V_{w-s} = V_{O-S} + V_{H_1-s} + V_{H_2-s} \tag{38}$$

donde los potenciales individuales para oxígeno e hidrógeno en la superficie hidrofóbica tienen la siguiente forma analítica, ecuación 39, que depende de la coordenada en el eje $\boldsymbol{Y}$:

$$V_{O-s} = V_{H-S} = Ae^{-By} + Ce^{-Dy} + Fe^{-Gy} \tag{39}$$

y, para la superficie hidrofílica son las expresiones 40 y 41, respectivamente.

$$V_{O-s} = A_o e^{-B_o y} - C_o e^{-D_o y} + F_o e^{-G_o y} \tag{40}$$

$$V_{H-s} = A_H e^{-B_H y} - C_H e^{-D_H y} + F_H e^{-G_H y} \tag{41}$$

Los coeficientes de los potenciales individuales para cada tipo de superficie, hidrofóbica e hidrofílica, son reportados en la siguiente tabla.

Átomo	Parámetro	Superficie Hidrofóbica [kJ/mol]	Superficie Hidrofílica [kJ/mol]
O	A	126763	-152785
	B	88.7494	94.7616
	C	-119802	-35653400
	D	89.3316	100.7009
	F	128810	-438.194
	G	88.7093	15.1075
H	A	126763	126763
	B	88.7494	88.7494
	C	-119802	-119802
	D	89.3316	89.3316
	F	128810	128810
	G	88.7093	88.7093

Tabla 5 Parámetros de las funciones del potencial de interacción entre el agua y los dos tipos de superficies: hidrofóbica e hidrofílica.

A continuación se muestran, para los dos tipos de superficies, la forma del potencial con los parámetros de la tabla anterior.

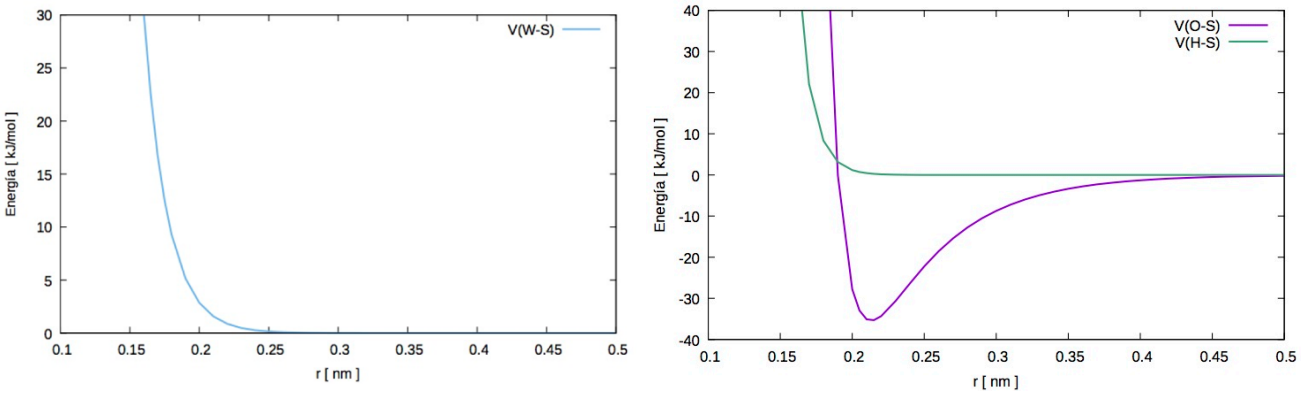

Figura 2 La superficie de energía potencial de interacción (SEPI) de los dos tipos de hidratación, en el lado izquierdo, la superficie hidrofóbica y en el lado derecho, la superficie hidrofílica. En este análisis, la SEPI representa una superficie sin estructura de Pt con una energía de adsorción de 35 kJ/mol.

3.3.2 Superficie Cargada Eléctricamente

En electroquímica se sabe que las reacciones ocurren en la superficie del electrodo, para ello, las especies moleculares deben viajar del seno de la disolución a la región interfacial, a este fenómeno se le llama transporte[10] y está dominado por un potencial externo aplicado al sistema, el cual recibe el nombre de potencial de Volta (**V**) y depende explícitamente de la posición de los átomos[1]. En el proceso de carga de un material, se aplica una diferencia de potencial (potencial de volta, **V**); el potencial externo genera dos regiones con cargas opuestas y de igual magnitud, que dependen de la distancia de separación y la corriente eléctrica utilizada. Aplicar un potencial a dos superficies planas paralelas con área y distancia de separación conocida, origina una densidad uniforme de carga superficial (σ). A partir de la ley de Gauss se determina el campo eléctrico ($\boldsymbol{E}$) generado por las dos láminas planas infinitas; la dirección del campo es perpendicular al plano y su magnitud es $\|\boldsymbol{E}\| = \frac{\sigma}{\varepsilon_0}$ [84,85]. La expresión del potencial eléctrico (V) se obtiene a partir del campo ($\boldsymbol{E} = -\boldsymbol{\nabla} V$), por lo que la energía potencial tiene la siguiente forma $U = -\frac{q\sigma y}{\varepsilon_0}$.

La energía potencial oxígeno-superficies se obtiene de la expresión 42, donde la coordenada $\boldsymbol{Y}$ está en nm:

$$V_{O-s} = \frac{(6.022\times10^{20})V_{volta}q_O(l_y - y)}{2l_y} \tag{42}$$

De la expresión anterior, l_y es la distancia de separación entre las superficies (en nm), V_{volta}, es la diferencia de potencial aplicado a las placas paralelas en el experimento y q_O, es la carga puntual del oxígeno, que depende del modelo de agua utilizado (ver sección 3.2). En el caso de hidrógenos, solo cambia la magnitud de la carga y el potencial total será expresado como la ecuación 38.

A continuación, se muestran las gráficas de la energía potencial entre oxígenos e hidrógenos con las superficies cargadas eléctricamente.

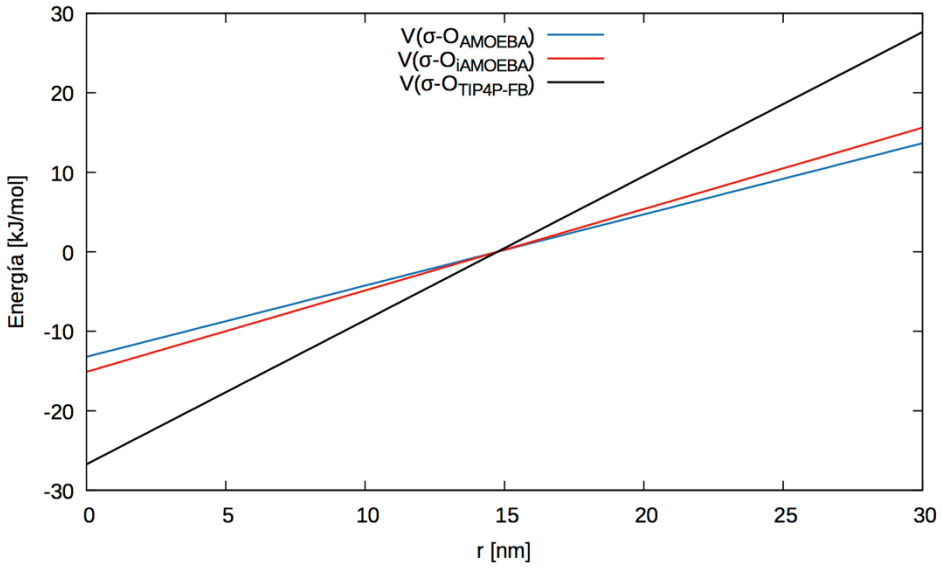

Figura 3 Superficie de energía potencial entre el oxígeno y las superficies cargadas. El plano con densidad de carga positiva está colocado en $y = 0$ y el plano contrario en $y = l_y$.

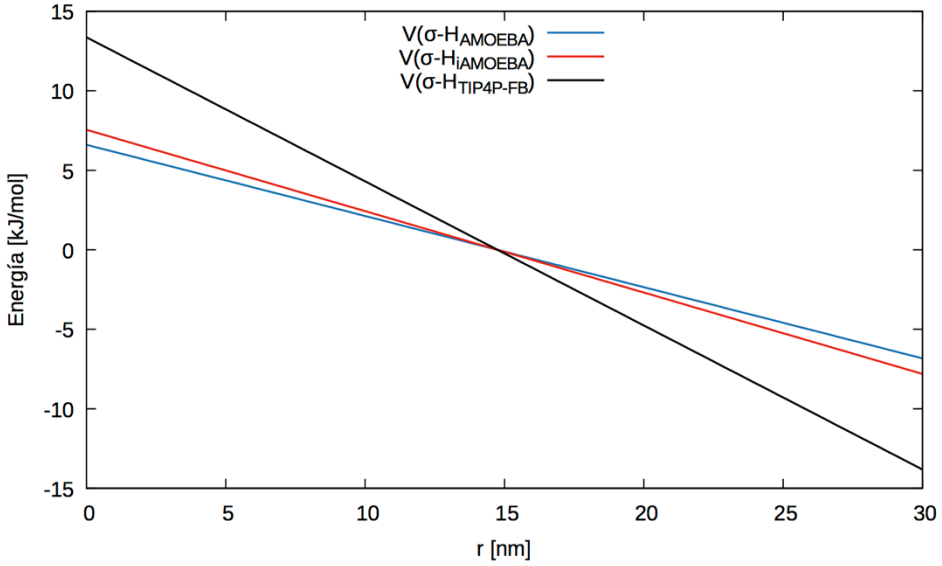

Figura 4 Superficie de energía potencial entre el hidrógeno y las superficies cargadas eléctricamente.

3.4 Implementación de interfases acuosas en OpenMM

Se plantearon tres modelos para el estudio del agua con interfases por DM, donde la diferencia principal es el tipo de interacción dominante entre el disolvente y la superficie sin estructura. Se validó un sistema de agua bulto para calcular propiedades de referencia (ver sección 3.5), otro con una superficie hidrofóbica con características similares al modelo de Rossky y un sistema con una pared hidrofílica como el modelo propuesto por Spohr. Ambos modelos de hidratación son sin átomos explícitos, es decir, modelos de paredes continuas. En el caso de la superficie cargada eléctricamente, se considera una superficie de tamaño infinito con densidad de carga superficial uniforme, esta aproximación es válida ya que las CPF permiten tener una pared con dichas características.

A continuación, se describen las características termodinámicas de los sistemas de estudio, la implementación de las superficies y la forma de crear un protocolo de simulación para calcular trayectorias en OpenMM.

- Modelo Agua Bulto

El sistema de agua bulto se simuló por DM con un ensamble NVT, fijando el volumen de la celda a una densidad de 1 g/mL, temperatura de 298.15 K, se aplicaron condiciones periódicas a la frontera y se utilizaron los potenciales de agua seleccionados.

- Modelo de Agua con superficies: hidrofóbica, hidrofílica y cargada eléctricamente

El agua confinada por dos superficies se simuló utilizando potenciales externos en las caras ($\boldsymbol{X}$,$\boldsymbol{Z}$) paralelas de la celda (dimensión igual a 19.705 x 19.705 $\AA$). El sistema se simuló en un ensamble NVT en una celda alargada en el eje $\boldsymbol{Y}$, se utilizaron CPF en dos dimensiones, las interacciones de largo alcance se calcularon con el método PME, la temperatura fue de 298.15 K, y densidad por celda de 1 g/mL.

Para los sistemas con paredes, se utiliza un potencial en cada una de ellas; los potenciales dependen de la distancia unidireccional a cada superficie. La distancia al plano $\pi_1 = y$ es el valor de y y, la distancia al plano definido como $\pi_2 = y - l_y$ se obtiene como $l_y - y$, donde l_y es el tamaño del eje $\boldsymbol{Y}$ de la celda de simulación (295.584 Å).

Se implementaron los protocolos de simulación en OpenMM[86], versión 7.1.1[87]. Se decidió utilizar este programa por las siguientes razones: se trata de un código libre; tiene implementados los potenciales para agua que se escogieron[88]; cuenta con una clase que permite implementar campos externos cuyas derivadas se calculan en forma analítica[89] y es multiplaforma[90,91].

Los sistemas con el campo de fuerzas de AMOEBA e iAMOEBA utilizaron un tamaño de paso de integración de 0.5 *fs*, un integrador-termostato de Langevin, las interacciones de largo alcance se calcularon con PME[73], se guardó información de la trayectoria cada 2000 pasos (1 *ps*) y se calcularon 50,000,000 de pasos en una unidad de procesamiento gráfico (GPU, por sus siglas en inglés). En el caso de AMOEBA se utilizó la polarización mutua con una tolerancia de 0.00001 y, para iAMOEBA una polarización directa. En el modelo TIP4P-FB se restringieron las distancias y ángulos de enlace, se utilizó un tamaño de paso de integración de 2 *fs*, un integrador-termostato de Langevin, y las interacciones de largo alcance se calcularon con PME. En todos los casos, se simularon 3840 moléculas de agua en una celda rectangular de 1.9 nm x 29.5 nm x 1.9 nm.

A continuación, se describe el uso del *script* de Python para calcular trayectorias en OpenMM. El archivo del anexo 7.1 se ejecuta de la siguiente manera y en ese orden, porque en las primeras 20 líneas se asignan las opciones de simulación a las variables termodinámicas y del cálculo de la trayectoria:

```
python nvt.py tip4pfb.xml AGP 0.002 500 25000 local 0.0
```

1. Las opciones para el modelo de agua son (ver sección 3.2): tip4pfb.xml, amoeba2013.xml e iamoeba.xml. En los siguientes puntos se hace referencia a modelos flexibles, pero son específicamente iAMOEBA y AMOEBA. Si el modelo necesita un sitio virtual se carga el archivo de coordenadas y topologías con la clase *modeller* que los calcula automáticamente (ver líneas 79-82, anexo 7.1). Además, con esta opción se crea el sistema de simulación, el cual depende de los grados de libertad estructurales (agua rígida/flexible), radios de corte de interacciones intermoleculares (vdW y electrostáticas), interacciones de largo alcance (PME con CPF), el tipo de polarización (modelos flexibles) y el criterio de convergencia (ver líneas 84-100 anexo 7.1).

2. Las opciones del tipo de simulación son (ver sección 3.3): AGP – agua bulto, HHFO- superficie hidrofóbica, HHFI- superficie hidrofílica y CHARGE- superficies cargadas.

3. La quinta variable es el tamaño de paso de simulación, para TIP4P-FB es de 0.002 *ps* y para los modelos flexibles es 0.0005 *ps*.

4. La sexta variable es la frecuencia en pasos para imprimir datos de la trayectoria. Como se recabo información cada *ps* y el paso de integración es diferente, entonces, en el modelo TIP4P-FB se imprime cada 500 pasos y en los otros cada 2000 pasos.

5. La séptima variable se refiere al factor necesario para obtener una trayectoria de x *ns*. Por ejemplo, si se calcula una trayectoria de 25 ns con el modelo iAMOEBA, el factor será de 25000 y se determina con la siguiente expresión:

$$f = \frac{longitud\ de\ simulación\ en\ ps}{tamaño\ de\ paso\ en\ ps * frecuencia\ de\ impresión} \tag{43}$$

6. Los laboratorios de cómputo nacional prestaron sus *clusters* con GPUS para calcular las trayectorias, se usó el mismo *script* pero se solicitaban más tarjetas gráficas y la ruta de donde se cargaban los archivos de entrada. La opción ocho puede ser local, abacus o yoltla (ver líneas 26-77 y 169-175 del anexo 7.1).

7. La última opción para el script es el potencial de volta aplicado, si el sistema es diferente de *CHARGE* siempre se da el valor de 0.0.

8. El potencial externo se implementa con la clase *CustomExternalForce* (ver líneas102-166 del anexo 7.1). Se crea un objeto fuerza que recibe como argumento una cadena de caracteres, es decir, la expresión del potencial de interacción de las superficies con el agua que depende de las coordenadas de los átomos (ver sección 3.3). Por ejemplo:

   ```
   force1 = CustomExternalForce('79.7884*exp(-12.4225*r);r=y')
   ```

   El objeto fuerza se llama “force1” y se agrega al sistema creado (ver paso 1) de la siguiente manera:

   ```
   system.addForce(force1)
   ```

   Si el campo es específico para cada átomo, se asigna con la comparación entre las masas de las especies presentes en el sistema:

   ```
   at2 = system.getParticleMass(i).value_in_unit(dalton)
   ```

   La implementación de los potenciales externos depende del tipo de superficie y para evitar interacciones entre copias de la celda en la dirección Y, se aumenta el tamaño de esta en 10 $Å$, distancia mayor al radio de corte utilizado.
9. El integrador-termostato utilizado es el de Langevin (ver línea 178 del anexo 7.1) y para el modelo TIP4P-FB se declara la tolerancia para la restricción de enlace (línea 183).
10. Hecho lo anterior, se crea el contexto de simulación y para ello, es necesario especificar las características del sistema como la topología, características de la simulación (paso 1), plataforma, configuración inicial, velocidades a partir de una distribución tipo Boltzmann (ver sección 3.1.5) y el integrador.

11. Además de las opciones básicas que OpenMM ofrece para guardar información de la DM, se implementaron dos funciones para guardar velocidades y fuerzas con la frecuencia seleccionada (ver anexo 7.18); clases que permiten imprimir los momentos dipolares permanentes, inducidos y totales de cada átomo, así como el momento dipolar total del sistema (ver anexo 7.17). Las clases solo se ocupan para los modelos flexibles, hay una forma específica de obtener la información de los momentos multipolares (ver líneas 194-200 del anexo 7.1) y las funciones sirven para cualquier modelo (línea 218 del anexo 7.1).

    Estos dos archivos se guardaron en la siguiente ruta: `~/anaconda2/lib/python2.7/site-packages/simtk/openmm/app/` para que estén disponibles en cualquier simulación y se importen con el contexto de openmm.
12. En el modelo TIP4P-FB se hace un cambio en la forma de calcular las interacciones intermoleculares, ya que se debe asegurar la continuidad del potencial, aunque se trunque a un radio de corte dado (ver líneas 202-210 del anexo 7.1).
13. La ultima estructura de control (*for*) se utiliza para guardar el momento multipolar del sistema y un archivo.xml que funciona para reiniciar el cálculo de la trayectoria (ver línea 214 del anexo 7.1). Si no se guarda el archivo.xml, el anexo 7.16 funciona para crearlo a partir de las posiciones y velocidades de las partículas.

Además de las características mencionadas acerca de OpenMM, también se pueden crear configuraciones iniciales del sistema de estudio, agregar diferentes especies moleculares como iones, minimizar el sistema (ver sección 3.1.5) o calcular trayectorias de DM en diferentes ensambles de forma continua.

3.5 Validación de modelos de agua en OpenMM

- Propiedades estructurales, dinámicas y termoquímicas

A continuación, se plantea el conjunto de diferentes pruebas empleadas en la caracterización del comportamiento del bulto. Esta parte del estudio sirvió para confirmar la implementación de los modelos y, posteriormente, comparar con la respuesta del solvente ante los diferentes tipos de superficies implementadas, lo que permite conocer el alcance del efecto de las paredes como función de la distancia a la superficie.

3.5.1 Entalpía de vaporización

La entalpía de vaporización es la energía necesaria para que una mol de sustancia cambie de la fase líquida a la fase gas, cuando las dos fases se encuentran a la misma presión y temperatura. En simulaciones numéricas, esta propiedad depende del modelo clásico de agua (flexible o rígido) y de la respuesta ante el medio en que se encuentra.

El problema de calcular la entalpía de vaporización para modelos como TIP4P-FB (no considerar la polarización) viene de la aproximación para un modelo rígido[68,76], porque al utilizar un sobrepotencial para fijar la geometría, es necesario hacer ajustes en el cálculo de las propiedades.

De acuerdo con Wang *et al.*[92], el cálculo de la entalpia de vaporización, para un modelo donde se considera la polarización, viene dado por la siguiente expresión:

$$\Delta H_v(T) = V_g(T) - V_l(T) + RT \tag{44}$$

donde, ya se despreció el término PV de la fase líquida por la diferencia entre volúmenes molares que existe con la fase gas. La energía potencial del líquido es la energía potencial promedio tomada de la dinámica molecular, el término RT es igual a

2.48 kJ/mol para una T=298.15 K y, el término de la energía potencial en la fase gas viene dado por la siguiente expresión[93]:

$$V_g(T) = E_g^{minimizada} + \frac{1}{2}RT(3N_{átomos} - 6 - N_{restricciones}) \quad (45)$$

La energía minimizada se calcula con el mismo potencial (AMOEBA e iAMOEBA), para los modelos flexibles es de -0.08 kJ/mol y, para TIP4P-FB es cero, porque no tiene grados de libertad estructurales. Por lo tanto, para modelos rígidos, el término de energía potencial en la fase gas es cero y esta expresión ya no es útil. De estudios previos se sabe que el valor calculado esta 15% arriba del reportado[74,94].

Los modelos que no consideran la polarización necesitan modificaciones para el cálculo de la entalpia de vaporización y otras propiedades termodinámicas. Berendsen *et al.* modifican la expresión 44-45, con un término que se denomina energía de corrección a la polarización[49,95,96]. El término de corrección tiene la siguiente forma analítica:

$$E_{pol} = \frac{1}{2}\frac{(\mu_{modelo} - \mu_g)^2}{\alpha} \quad (46)$$

Donde μ_{modelo} es el momento dipolar del modelo de agua (2.43 D para TIP4P-FB), μ_g es el momento dipolar del agua en la fase gas (1.85 D) y α es la polarizabilidad molecular experimental $(1.65x10^{-40}C^2m^2/J)$[1]. Este término disminuye en 6.84 kJ/mol a la expresión propuesta por Wang. La entalpía de vaporización del modelo TIP4P-FB es de **52.17 kJ/mol** sin corrección y **45.32 kJ/mol** (10.83 kcal/mol) corregido. En la siguiente tabla se muestra una comparación de esta propiedad para los tres modelos de agua.

MODELO DE AGUA	VALOR CALCULADO [kcal/mol]	VALOR REPORTADO [kcal/mol]
iAMOEBA	10.84	10.94
AMOEBA	10.45	10.48
TIP4P-FB	10.85	10.80

Tabla 6 Entalpía de vaporización de agua líquida con tres modelos diferentes y 25 ns de simulación.

3.5.2 Función de distribución radial

La función de distribución radial (FDR), es una cantidad estructural, indica la probabilidad de encontrar un par de átomos a una distancia dada y es relativa a la probabilidad esperada para una distribución aleatoria en el bulto[64,97,98]. Cuando se calcula de simulaciones numéricas, se toma el valor promedio sobre las réplicas de la trayectoria, es decir, se calcula la densidad promedio ($\langle \rho_{r+\Delta r} \rangle$) para una rebanada de una esfera y se normaliza entre la densidad del bulto, como en la siguiente ecuación:

$$g(r + \Delta r) = \frac{3V_{celda}\langle N_{r+\Delta r} \rangle}{4\pi N^2 (r_{r+\Delta r}^3 - r_r^3)} \tag{47}$$

A continuación, se reportan las FDR para los tres potenciales de agua y los tres pares atómicos, calculadas con código propio (ver anexo 7.24). En línea negra continua el modelo TIP4P-FB, en línea roja el modelo iAMOEBA, en azul el AMOEBA y en línea punteada la FDR experimental.

La g(O-O) presenta los picos característicos a la distancia esperada para la primera y segunda capa de vecinos; hay mayor altura en el primer pico con respecto al experimento, pero en acuerdo con lo reportado para cada uno de los modelos se esperaba este comportamiento.

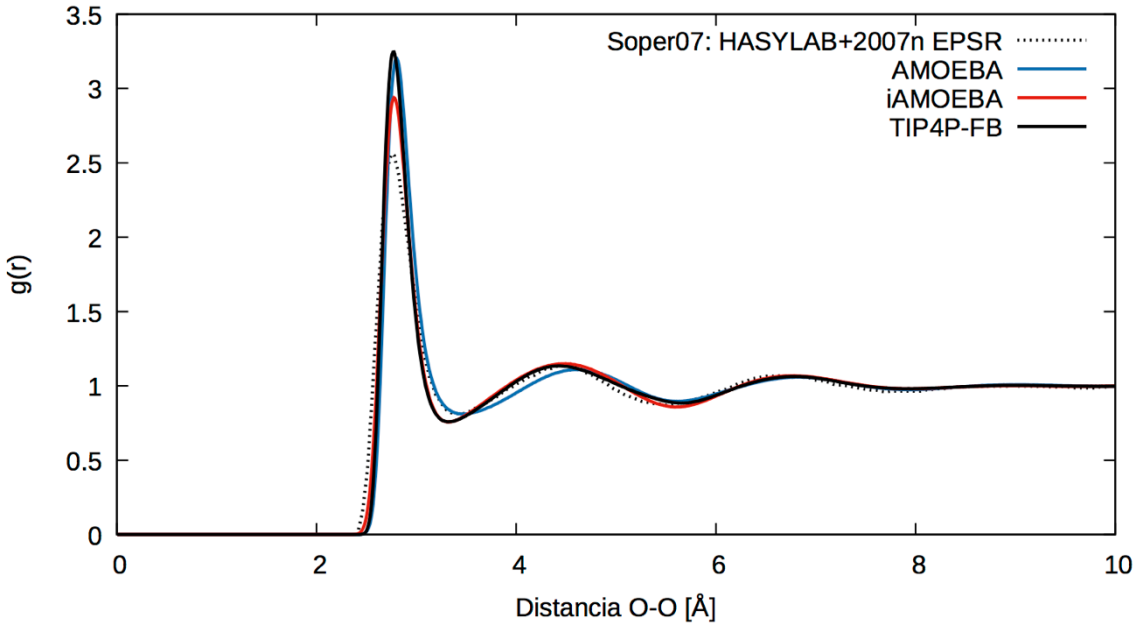

Figura 5 Función de distribución radial del par atómico O – O.

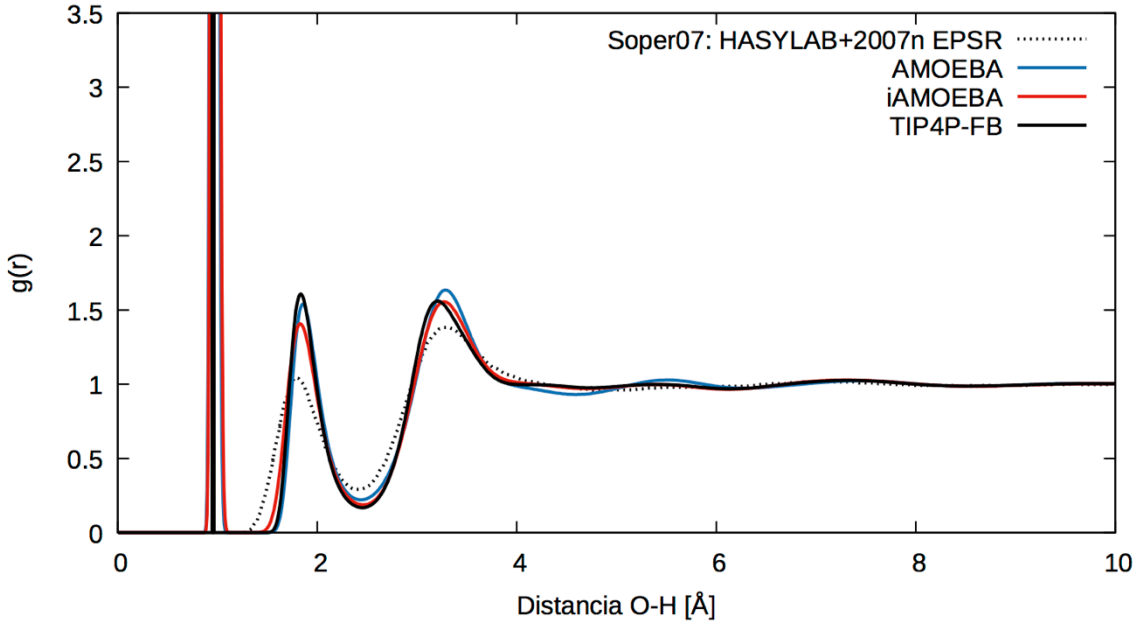

Figura 6 Función de distribución radial del par atómico O – H.

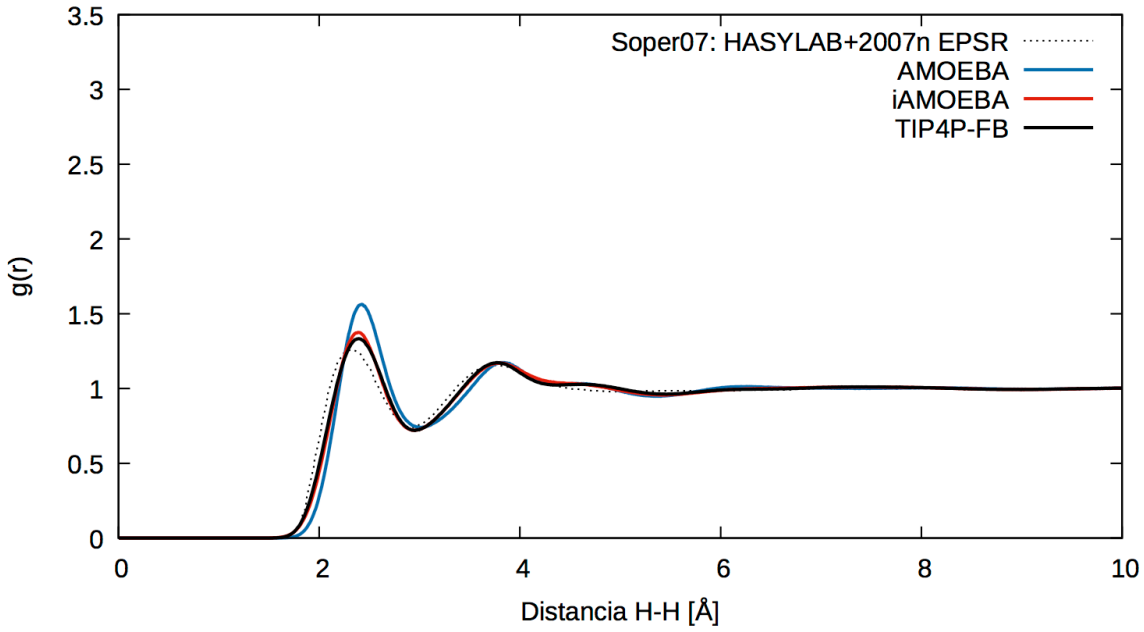

Figura 7 Función de distribución radial del par atómico H – H.

Tanto la g(O – H) como la g(H – H), tienen el comportamiento esperado y, no se encuentran diferencias significativas con respecto a los valores reportados.

3.5.3 Perfil de densidad

Se verificó que la distribución de las moléculas en la celda fuese homogénea. Como la celda empleada tiene forma de prisma cuadrangular, se dividió el eje más largo ($\boldsymbol{Y}$) en bloques del mismo tamaño y se calculó la densidad en cada bloque. A partir de la información de la FDR, es posible saber que una distancia de 5.6 $\AA$ es suficiente para agrupar primeros y segundos vecinos, por ello la celda se dividió en 52 bloques de la siguiente manera:

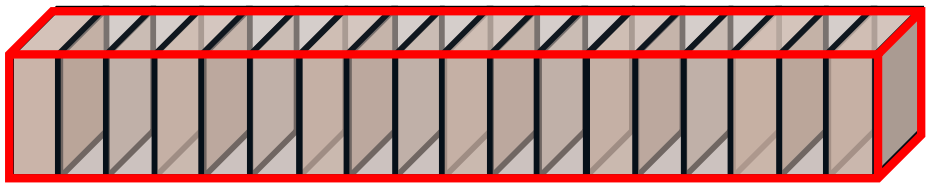

Figura 8 Representación de la celda dividida por bloques a lo largo del eje $\boldsymbol{Y}$.

En cada réplica o paso de la simulación archivado, se contó el número de oxígenos por bloque y, con el total de la trayectoria se obtiene un valor promedio (ver anexo 7.2). Se calculó el perfil de densidad para los tres modelos de bulto y a continuación se muestran:

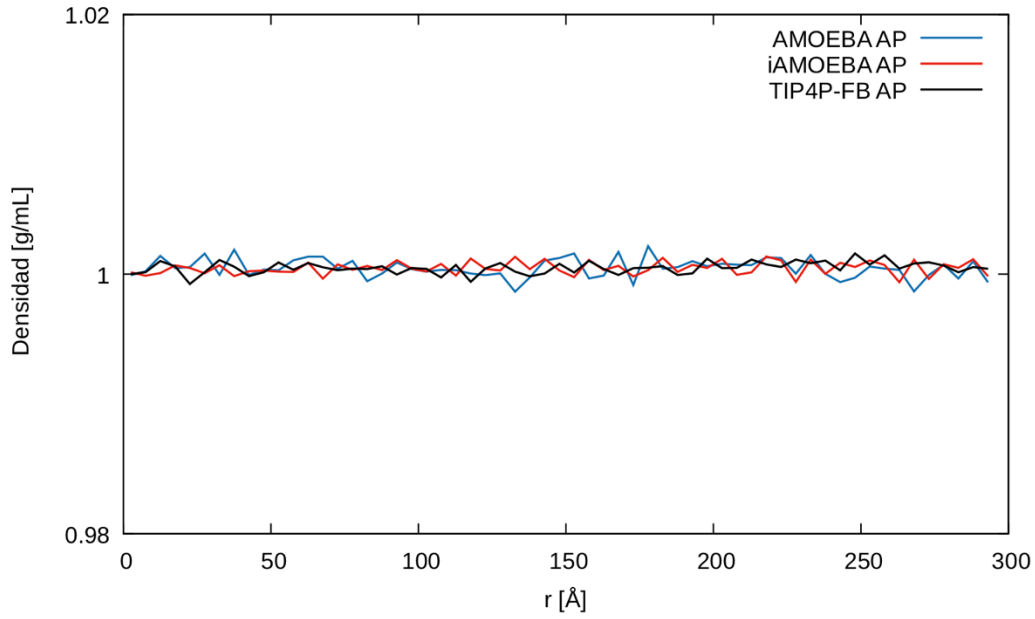

Figura 9 Perfil de densidad de agua bulto en 52 bloques a lo largo del eje $\boldsymbol{Y}$. El perfil oscila en torno a 1 g/mL que es la densidad impuesta en la celda. En línea color azul el modelo AMOEBA, en color rojo el iAMOEBA y en negro el modelo TIP4P-FB.

- Importancia del punto de partida de la simulación

En los sistemas de agua confinada se calculó el perfil de densidad promedio para un tiempo de simulación mayor a 10 ns. En la primera ocasión que esto se intentó se obtuvo un resultado como el que se muestra en la figura 10. Allí se observa que el perfil de densidad obtenido en el sistema simulado con las paredes hidrofóbicas se obtiene una pendiente diferente de cero $\left(-0.00018\frac{g}{mL \cdot ns}\right)$. Sabemos que a lo largo de la trayectoria las propiedades presentan fluctuaciones en torno a un valor medio por lo que se esperaría que, para tiempos de largos de simulación (orden de *ns*), el sistema se estabilizara. Aun dejando evolucionar el sistema hasta 30 ns, se obtenían perfiles como el mostrado en la figura 11.

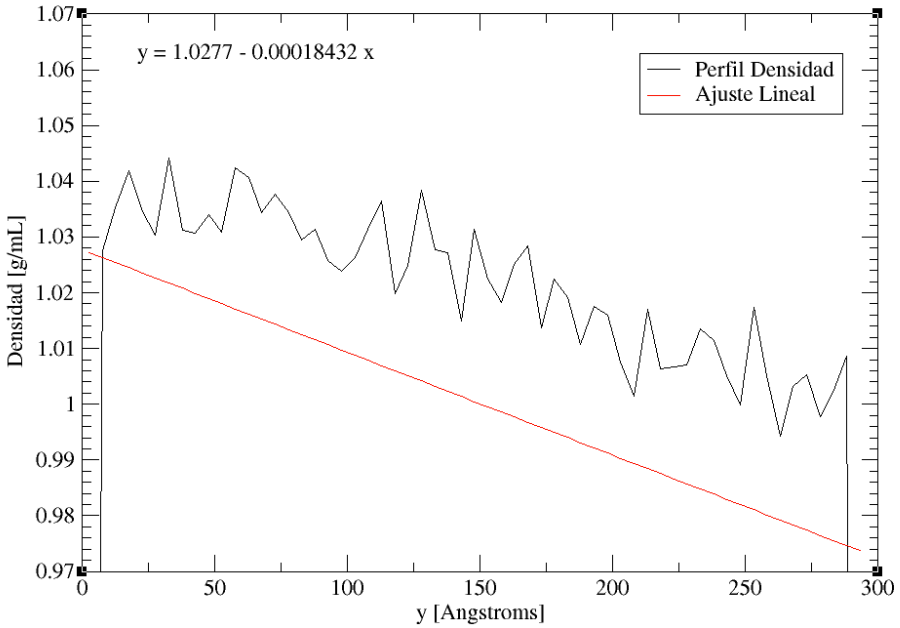

Figura 11 Perfil densidad de agua confinada entre paredes hidrofóbicas en una trayectoria con 15 ns de tiempo de simulación. El ajuste lineal tiene pendiente negativa. El sistema contiene 3840 aguas con el modelo TIP4P-FB. En línea roja se reporta el ajuste lineal del perfil.

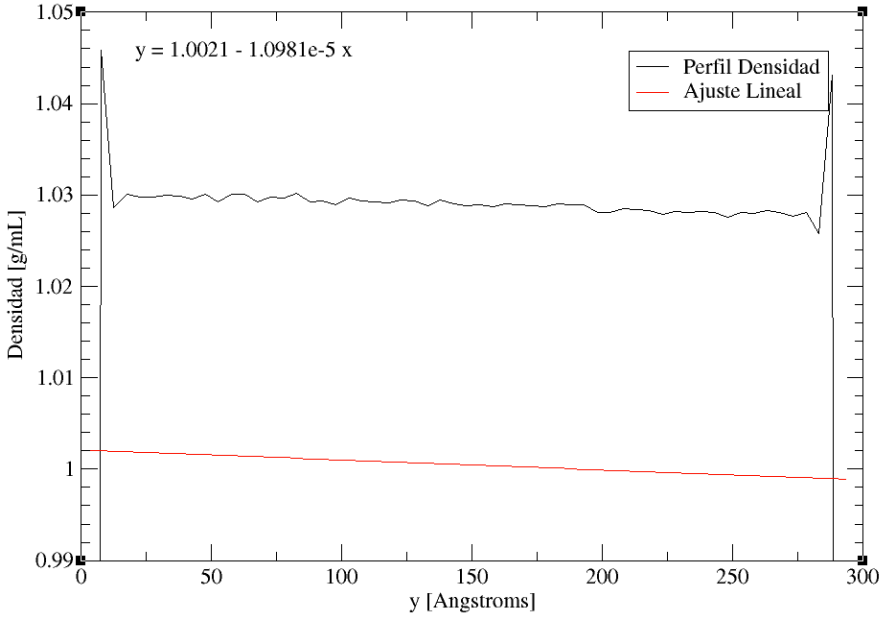

Figura 10 Perfil de densidad de agua bulto ante paredes hidrofóbicas para un tiempo de simulación de 30 *ns*. En línea roja se reporta el ajuste lineal para el perfil de densidad de 3840 moléculas de agua con el modelo TIP4P-FB.

Allí se observa que la pendiente aún es diferente de cero, pero se observa un perfil de densidad con simetría en los extremos. La pregunta más importante es: si el perfil de densidad converge con el tiempo, ¿cuánto tiempo de simulación es necesario para asegurarse que los transitorios observados en esta propiedad reflejan el equilibrio del sistema? El valor promedio del perfil de densidad no es útil para conocer el tiempo requerido para llegar al equilibrio, por lo tanto, se calculó el ajuste lineal para cada punto de la trayectoria en un t >30 *ns* (ver anexo 7.19).

En la siguiente imagen, se muestra la pendiente del perfil de densidad ante el tiempo de simulación, para el sistema de agua TIP4P-FB con los potenciales hidrofóbico e hidrofílico. En esta DM se guardó una configuración del sistema cada *ns* durante un tiempo de simulación mayor a 500 *ns* siendo la configuración inicial, la mostrada en el anexo 7.9. Esta configuración inicial fue obtenida empleando códigos que distribuyen las moléculas del sistema de manera regular en una malla tipo cristal *fcc*[2].

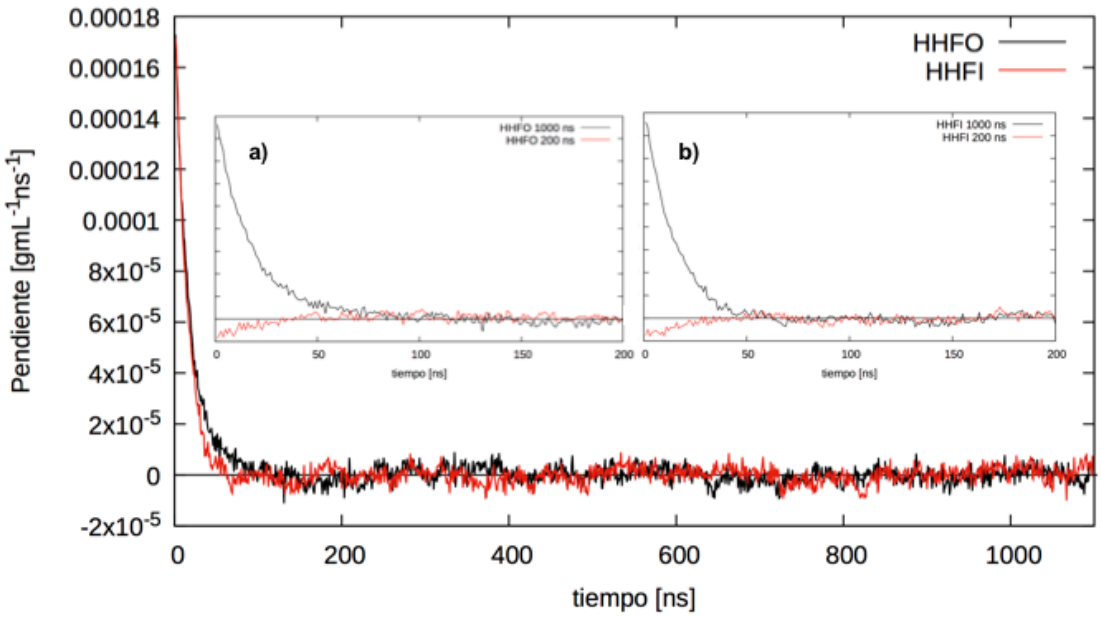

Figura 12 Gráfico de la pendiente del perfil de densidad del sistema en cada punto de la trayectoria. En el gráfico principal se muestra en color negro el sistema con superficie hidrofóbica y en color rojo la superficie hidrofílica. El tiempo de simulación fue de 1000 *ns.* En los gráficos a) y b) se compara el perfil de la pendiente con sistemas que inician con configuraciones diferentes; en color rojo, la configuración de agua bulto equilibrada por 100 *ns* (ver anexo 7.10) y en negro, la configuración tipo emparedado (ver anexo 7.9).

[2] Taller de Dinámica Molecular. Algoritmos, Análisis y Aplicaciones en programas paralelos. Disponible en: http://quimica.izt.uam.mx/TallerDM/TDM2016/

En el gráfico a) se observa que, iniciar con una configuración con mayor similitud al agua líquida permite alcanzar una distribución homogénea, pendiente ≈ 0, en un tiempo de simulación de 50 ns, a diferencia de la otra en la que requiere más tiempo de simulación para alcanzar el equilibrio. En el gráfico de color rojo en el caso b), se observa que el valor de la pendiente refleja el equilibrio en un tiempo de 40 ns, mucho más rápido que cuando se parte de la configuración de malla cristalina y que la superficie HHFO.

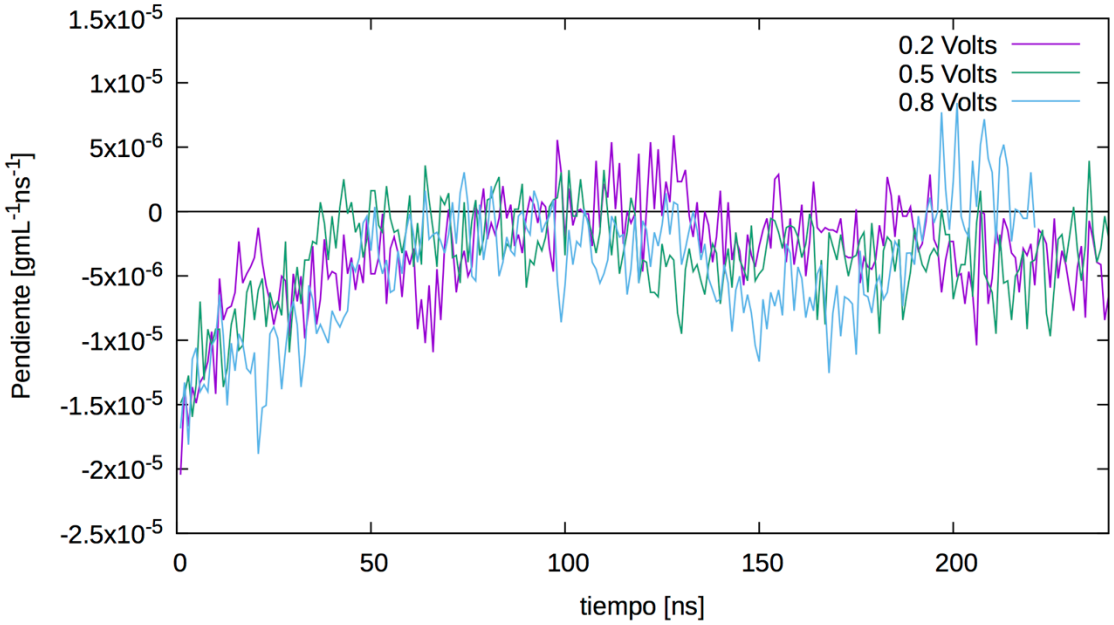

Figura 13 Gráfico de la pendiente contra el tiempo de simulación en sistemas con superficies cargadas eléctricamente. En color morado, el sistema con un potencial aplicado de 0.2 V; en color verde, un potencial aplicado de 0.5 V y en color azul de 0.8 V.

En la figura anterior (13), se reporta la pendiente del perfil de densidad como función del tiempo en sistemas a los que se aplicó un voltaje diferente. La configuración de inicio fue agua equilibrada durante 100 *ns*. A diferencia de las superficies hidratadas, este tipo de superficies requiere mayor tiempo para alcanzar el equilibrio. Al parecer, se requiere un tiempo de 100 *ns* para alcanzar la pendiente cero, sin embargo, los tres sistemas presentan oscilaciones con periodos distintos. Por ejemplo, en el sistema con el potencial más bajo, 0.2 V, las oscilaciones son más frecuentes, pero más cercanas a cero; a diferencia de la superficie con potencial de 0.8 V donde el periodo es más amplio pero el promedio está por debajo de cero.

Estos gráficos son inusuales; el perfil de densidad converge muy lentamente en comparación a otras propiedades y hasta el momento, Ferrara *et al*. son los únicos que han equilibrado sus sistemas de estudio durante algunas centenas de *ns*, pero no mencionan que esto se haya debido a problemas en el cálculo de las propiedades de interés. Hay que notar que, a diferencia de las superficies utilizadas en este proyecto, en el análisis hecho por Ferrara la fase sólida tiene estructura[99].

El uso de potenciales polarizables y flexibles como AMOEBA o incluso iAMOEBA, es caro computacionalmente por lo que requerir trayectorias de varias centenas de *ns* para equilibrar el sistema se vuelve poco práctico. Por lo que, consideramos necesario un protocolo de simulación específico para el estudio de sistemas de este tipo (ver sección 3.6).

3.5.4 Distribución de la orientación del momento dipolar

En este análisis, se calculó el coseno del ángulo formado entre el vector de momento dipolar y el vector normal que apunta hacia afuera de la superficie más cercana (ver anexo 7.3), la definición de este ángulo se muestra en la sección a) de la figura 14.

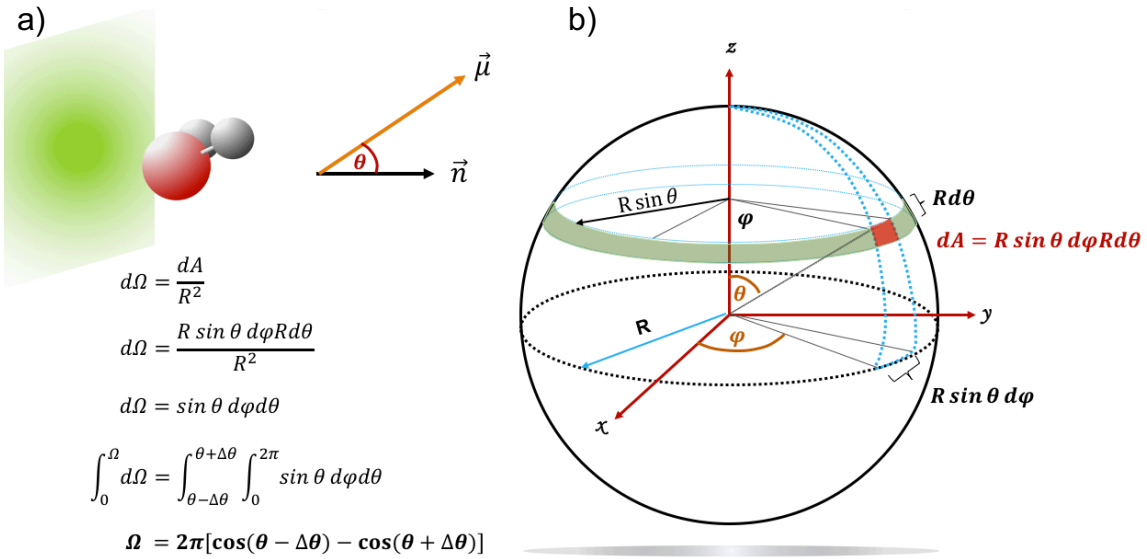

Figura 14 a) Definición del ángulo θ formado entre el vector normal a la superficie y el momento dipolar. b) El ángulo sólido generado para un mismo valor de θ pero en una franja de 10º de apertura ($\Delta\theta = 5°$).

La celda se divide por bloques como en el análisis del perfil de densidad, pero en cada bloque se calcula la probabilidad de tener diferentes orientaciones del coseno del ángulo en el rango $0 \leq \theta \leq 180°$ y se normaliza con el ángulo sólido en función de θ, con la franja de una esfera de radio uno.

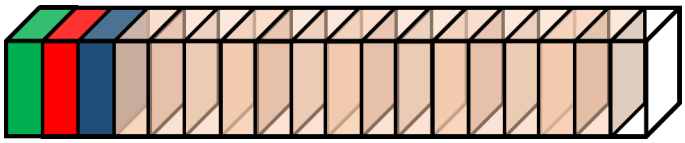

Figura 15 División de la celda de simulación por bloques, cada bloque tiene un ancho de 5.6 *Å*

En la siguiente figura se reporta la distribución de orientaciones del bulto hacia un plano imaginario $\pi_1 = y$, el cual corresponde a la posición de la superficie para los sistemas de agua con paredes. Se grafican las orientaciones preferenciales de las divisiones cercanas a la superficie; así los bloques en color verde, rojo y azul de la figura anterior, corresponden al bloque 1, 2 y 3, respectivamente; los modelos de agua se diferencian por el código de colores hasta ahora usado.

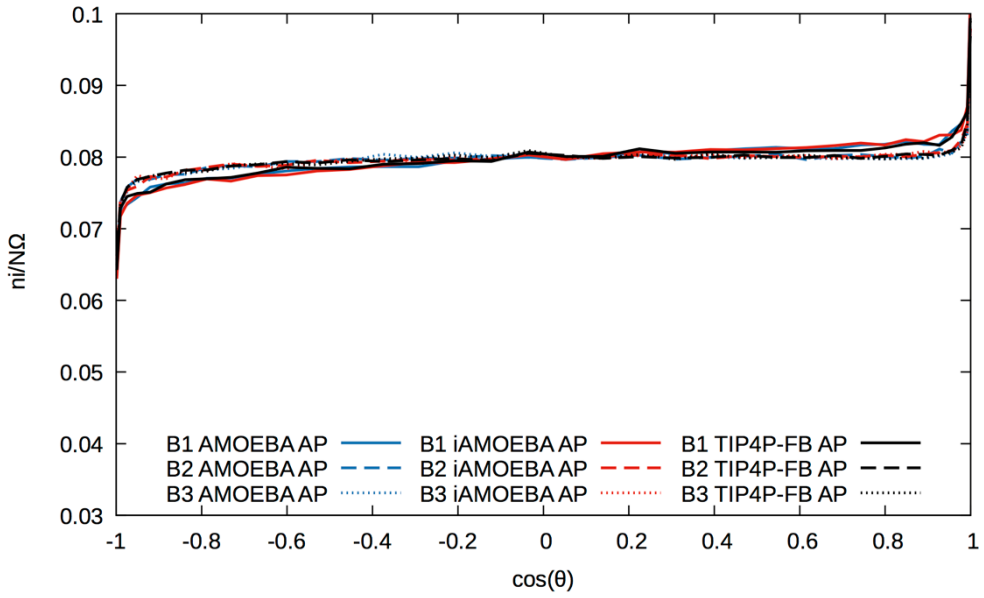

Figura 16 Distribución del coseno de θ en los bloques cercanos a la superficie imaginaria. Los tres modelos de agua, en color negro TIP4P-FB, en rojo iAMOEBA y en azul AMOEBA.

El gráfico confirma que las orientaciones del agua bulto no tienen una preferencia. La figura anterior servirá de referencia para identificar, en los sistemas confinados el alcance del efecto de la superficie.

3.5.5 Coeficientes de Auto difusión

Se sabe que los coeficientes de auto-difusión calculados con la ecuación de Einstein (48) dan resultados razonables para los modelos flexibles a partir de la información recabada de la trayectoria.

$$D = \lim_{t\to\infty} \frac{1}{6t} \langle |\boldsymbol{R}_i(t) - \boldsymbol{R}_i(0)|^2 \rangle \qquad (48)$$

De acuerdo con Zlenko[100], utilizar la ecuación de Einstein es suficiente para obtener un buen resultado del coeficiente de auto-difusión[49], bastaría contar con un número de configuraciones suficientes para tiempos de simulación del orden de algunos *ns*. Cada modelo tiene un tiempo de relajación específico; así que se ajustaron tiempos diferentes de simulación que resultasen en el mismo número de puntos guardados de la trayectoria a fin de tener posibilidad de comparar. Hecho lo anterior, los coeficientes de auto difusión se reportan en la siguiente tabla.

MODELO DE AGUA	VALOR CALCULADO [cm^2/s]	VALOR REPORTADO [cm^2/s]
AMOEBA	$2.0x10^{-5}$	$1.99x10^{-5}$
iAMOEBA	$2.2x10^{-5}$	$2.54x10^{-5}$
TIP4P-FB	$2.1x10^{-5}$	$2.21x10^{-5}$

Tabla 7 Coeficientes de auto-difusión del bulto para los tres campos de fuerza: iAMOEBA, AMOEBA y TIP4P-FB.

Lo anterior funciona porque el cálculo del coeficiente de auto-difusión es un evento de "*memoria*", donde cada configuración está relacionada; en este análisis la correlación es con la configuración inicial. Ver anexo 7.4 para el cálculo de los coeficientes.

3.5.6 Constante dieléctrica

La constante dieléctrica depende de la capacidad de respuesta del sistema al campo eléctrico externo homogéneo aplicado, es decir, está relacionada con las fluctuaciones del momento dipolar del sistema[101–104]($\vec{M} = \sum_{i=1}^{m} q_i \vec{r_i}$). De acuerdo con la teoría propuesta por Neumann, el sistema de interés está inmerso en un medio continuo (*Reaction Field*, RF por sus siglas en inglés) con ε_{RF} conocida[105]. La ε se calcula con la ecuación 49, donde el término del lado derecho se atribuye a las fluctuaciones del momento dipolar causadas por el campo eléctrico generado por el medio continuo.

$$\frac{4\pi}{3}\frac{\langle M^2 \rangle}{3VTk_B} = \frac{\varepsilon - 1}{\varepsilon + 2}\left[1 - \frac{\varepsilon - 1}{\varepsilon + 1}\frac{\varepsilon_{RF} - 1}{\varepsilon_{RF} + 1}\right]^{-1} \tag{49}$$

De la expresión anterior se obtiene la ecuación 50 donde se aplica la condición límite cuando ε_{RF} tiende a un valor infinito.

$$\frac{4\pi}{3}\frac{\langle M^2 \rangle}{3VTk_B} = \frac{\varepsilon - 1}{3} \tag{50}$$

Esta última ecuación es válida utilizando el sistema de unidades Gaussianas. La conversión al sistema internacional se hace al multiplicar el lado izquierdo por el factor $\frac{1}{4\pi\varepsilon_0}$. Este factor se obtiene a partir de la conversión, de la susceptibilidad eléctrica ($\chi^{SI} = 4\pi\chi^{G}$) y la constante dieléctrica ($\varepsilon^{SI} = \varepsilon_0 \varepsilon^{G}$), de unidades gaussianas al sistema internacional. La ecuación 51 es la que se usa para calcular la constante dieléctrica como el promedio acumulado en la trayectoria de DM (ver anexos 7.5, 7.6 y 7.7) [106–108].

$$\varepsilon = 1 + \frac{\langle M^2 \rangle}{3VTk_B\varepsilon_0} \tag{51}$$

Donde V es el volumen de la celda de simulación, T la temperatura absoluta, k_B la constante de Boltzmann y ε_0 la permitividad del vacío.

En la imagen 17, se muestra la gráfica de ε en función del tiempo de simulación para el bulto con los tres potenciales de agua. Se observa que el tiempo de relajación del sistema es diferente para cada modelo: con TIP4P-FB se necesitan 4 *ns* para tener un valor casi constante y en los modelos flexibles el tiempo es mayor que 10 *ns*.

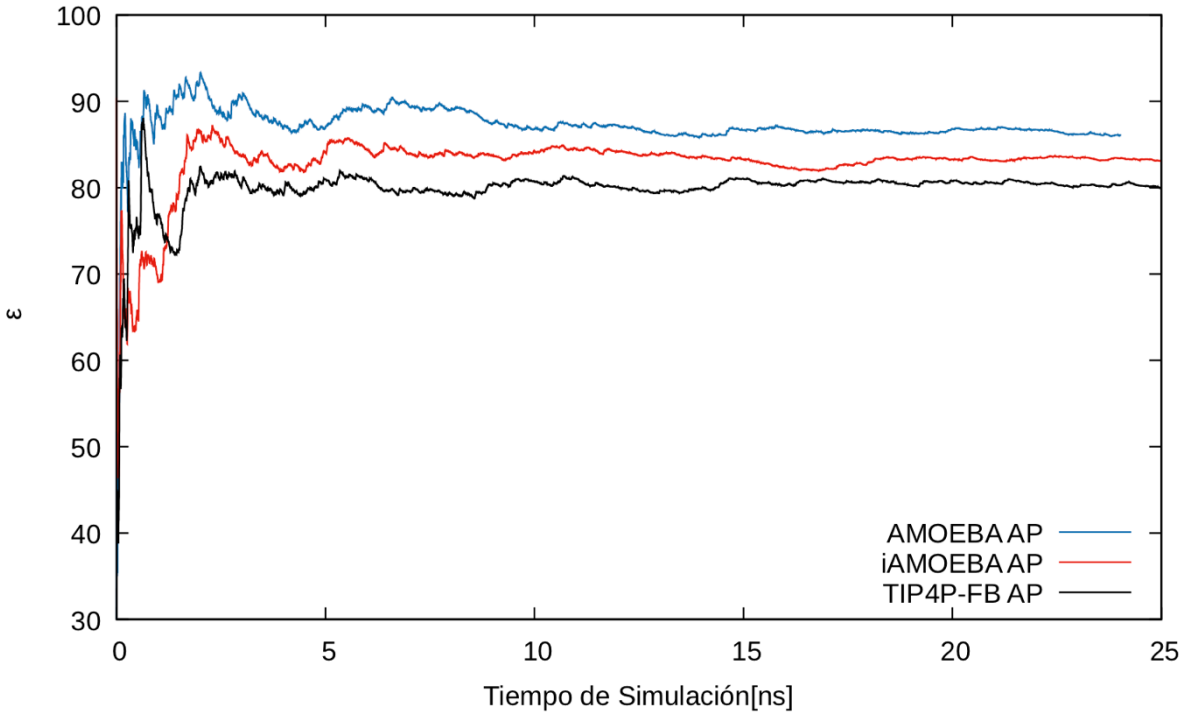

Figura 17 Promedio acumulado de la constante dieléctrica en función del tiempo de simulación.

Por un lado, el valor de ε para el modelo TIP4P-FB ha convergido a un valor cercano del experimental, ver tabla 8. Por el contrario, los modelos flexibles aún están por arriba tanto del valor reportado como del experimental, pero se ha reportado que esta propiedad converge muy lentamente[109]. En la siguiente tabla se muestran los valores de la constante dieléctrica calculados en el periodo de 10 a 15 *ns*.

MODELO DE AGUA	ε VALOR CALCULADO	ε VALOR REPORTADO
AMOEBA	86.8 ± 0.5	81.4
iAMOEBA	83.9 ± 0.4	80.7
TIP4P-FB	80.3 ± 0.5	78.4

Tabla 8 Constante dieléctrica del bulto de los tres potenciales de agua utilizados.

Existe otra metodología, propuesta por Thomas Simonson para calcular ε, sin embargo, la geometría esférica del sistema juega un papel importante, ya que de ello dependen las características de la simulación[110,111]. Por ejemplo, en la siguiente sección se detalla el protocolo propuesto para estudiar interfases acuosas: la interfase sólido-líquido.

3.6 Protocolo de Simulación para interfases acuosas

A continuación, se describen los pasos a seguir para la puesta en marcha del cálculo de trayectorias por dinámica molecular de agua bulto y agua ante superficies sin estructura.

1. Equilibrar agua con TIP4P-FB durante 100 *ns*, verificando que las propiedades que requieren mayor tiempo de simulación como la constante dieléctrica hayan convergido (ver anexo 7.5, 7.5 y 7.21).
2. Las dinámicas moleculares con los potenciales agua-superficie requieren que se inicie con una configuración de agua equilibrada (tomar la última configuración del paso anterior), y para ello, es necesario activar un potencial menos repulsivo al menos un par de centenas de *ps*, que desplace el agua lejos de las superficies; el único objetivo es alejar a las moléculas de la zona altamente repulsiva, y así, evitar que en la trayectoria las fuerzas sean muy grandes y generen errores numéricos. Los potenciales sugeridos para cada superficie son los siguientes:

$$V_{w-s1} = 70.3515e^{-12.4223y} \quad (52)$$

$$V_{w-s2} = 70.3515e^{-12.4223(l_y-y)} \quad (53)$$

3. Activar los potenciales: hidrofóbico e hidrofílico para el Pt y, equilibrar un tiempo de simulación mayor que 60 *ns* con TIP4P-FB (ver sección 3.2.3) para asegurar que el perfil de densidad tendrá pendiente cerca de cero. El sistema con las superficies cargadas requiere una barrera física (como las expresiones 52 y 53), porque el potencial está definido para actuar únicamente de 0 a l_y; no hacerlo permite que las moléculas salgan de la celda de simulación en la dirección $\boldsymbol{Y}$. El tiempo para equilibrar sistemas de este tipo debe ser mayor a 200 *ns*, se sugiere utilizar el potencial rígido para equilibrar y asegurar perfiles de densidad convergidos (ver anexo 7.20).
4. Tomar la última configuración del paso anterior para calcular trayectorias con los modelos iAMOEBA, AMOEBA y TIP4P-FB por un tiempo de 25 *ns* (ver anexo 7.11 y 7.12).

CAPÍTULO CUATRO

RESULTADOS Y DISCUSIÓN

4. RESULTADOS Y DISCUSIÓN

Los resultados que a continuación se presentan fueron hechos analizando el comportamiento en toda la celda de simulación, sin embargo, se hicieron dos análisis como función de la distancia a la superficie (bloques): los perfiles de densidad y la distribución de las moléculas de agua.

4.1 Entalpía de vaporización

En la tabla 9, se muestra la entalpía de vaporización del agua en bulto y ante las cinco interfases acuosas. La ΔH_V calculada para el sistema con superficies hidrofóbicas es, para todos los modelos, menor que la del bulto ($\approx -0.19\ \%$). Por el contrario, la superficie hidrofílica conduce a valores mayores de ΔH_V con todos los modelos, esto se debe a la atracción que ejerce sobre las moléculas de agua lo que hace que se requiera más de aproximadamente 1.4% más de energía para hacer el cambio de fase. En los sistemas con superficies cargadas eléctricamente se requiere menor energía en comparación con el bulto ($\approx -0.13\ \%$), pero más energía que en la superficie hidrofóbica ($\approx -0.05\ \%$). No se observó efecto sobre esta propiedad como función del campo eléctrico aplicado.

MODELO DE AGUA	ΔH_V Bulto [kcal/mol]	ΔH_V HHFO [kcal/mol]	ΔH_V HHFI [kcal/mol]	ΔH_V 0.2V [kcal/mol]	ΔH_V 0.5V [kcal/mol]	ΔH_V 0.8V [kcal/mol]
AMOEBA	10.45	10.43 (-0.21)	10.61 (1.50)	10.43 (-0.16)	10.43 (-0.16)	10.43 (-0.16)
iAMOEBA	10.84	10.82 (-0.19)	10.99 (1.43)	10.83 (-0.12)	10.83 (-0.12)	10.83 (-0.12)
TIP4P-FB	10.85	10.83 (-0.16)	11.03 (1.63)	10.84 (-0.11)	10.84 (-0.11)	10.84 (-0.11)

Tabla 9 Entalpía de vaporización de agua líquida con tres tipos de superficies; hidrofóbica (HHFO), hidrofílica (HHFI) y cargada eléctricamente: 0.2V. 0.5V y 0.8V. Entre paréntesis, está la diferencia porcentual entre el bulto y cada superficie.

4.2 Funciones de distribución radial

A continuación, se presentan las FDR de los pares atómicos O – O para los tres modelos de agua simulados en las cinco diferentes condiciones. En general, no se observaron diferencias significativas con respecto a la g(O-O) del bulto pero la posición y forma del primer mínimo refleja algunos ligeros cambios que se muestran

en los recuadros de cada figura. La superficie hidrofóbica presenta un aumento de la probabilidad en el primer mínimo. Este comportamiento, en mayor magnitud, también sucede en las trayectorias con superficies cargadas.

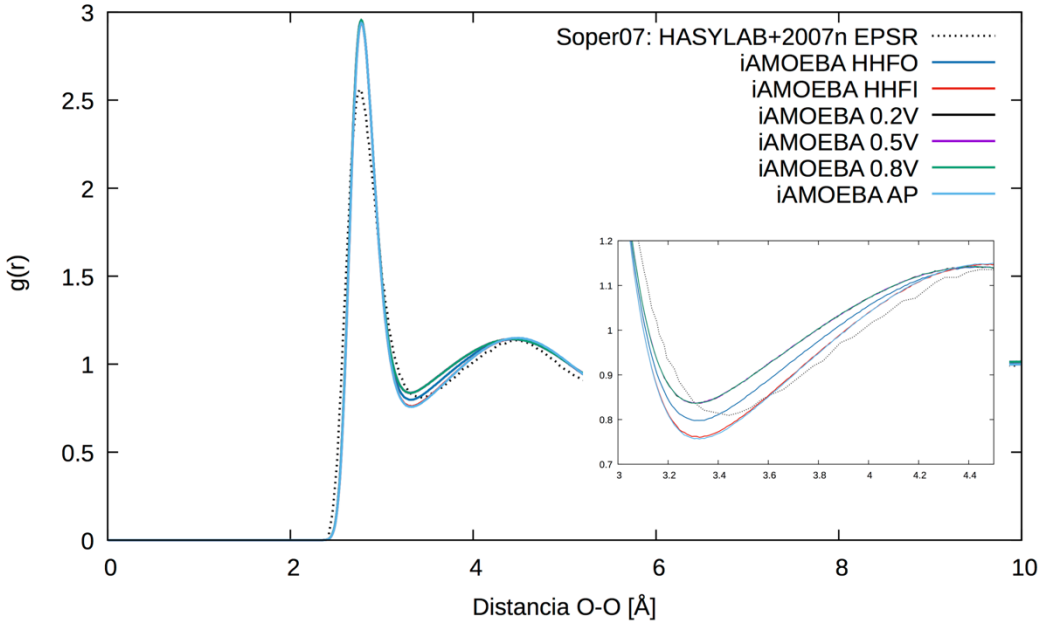

Figura 19 Función de distribución radial del par atómico O – O del modelo iAMOEBA y los cinco tipos de superficie. superficie hidrofóbica en color azul rey (HHFO); hidrofílica en rojo (HHFI); cargada con un potencial de 0.2 V en color negro, 0.5 V en color morado y 0.8 V en color verde; el bulto en color azul cielo y la referencia experimental en línea punteada. El gráfico más pequeño es un aumento de la zona del primer mínimo de la FDR.

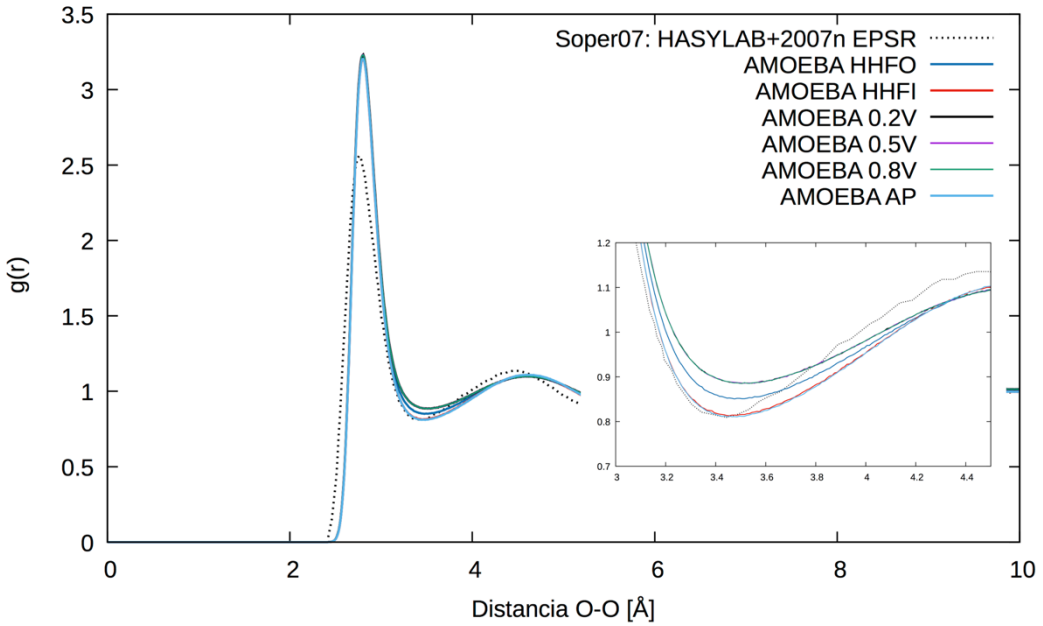

Figura 18 Función de distribución radial del par atómico O – O del modelo AMOEBA y los cinco tipos de superficie. El gráfico más pequeño es un aumento de la zona del primer mínimo de la FDR.

Al igual que en la figura 18, el potencial AMOEBA no presenta diferencias significativas en la FDR (ver figura 19), pero ocurre un aumento de la probabilidad en la zona del primer mínimo para la superficie HHFO y las superficies cargadas. La siguiente FDR refleja el mismo comportamiento que los modelos anteriores, no hay diferencias con respecto al bulto, sólo para la superficie hidrofílica.

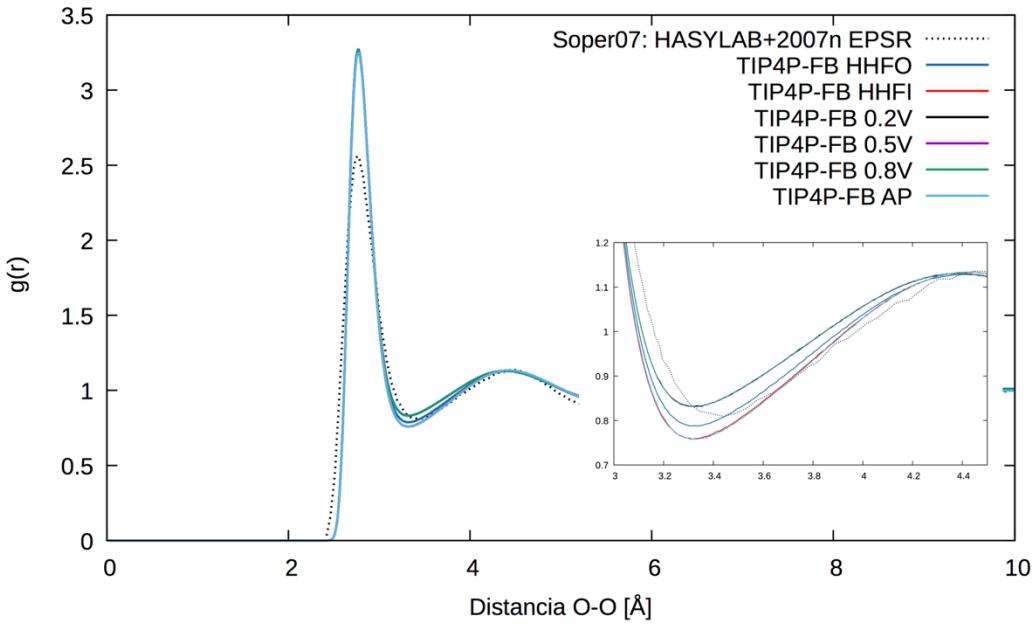

Figura 20 Función de distribución radial del par atómico O – O del modelo TIP4P-FB y los cinco tipos de superficie. El gráfico más pequeño es un aumento de la zona del primer mínimo de la FDR.

4.3 Perfiles de densidad

En este análisis se eligió dividir la celda en bloques de tamaño igual a la distancia de la posición del segundo mínimo en la FDR para el par atómico O – O de agua bulto, 5.6 *Å* de espesor.

En los perfiles de densidad que se muestran en las figuras 21, 22 y 23, se observa que el efecto del potencial HHFO y las superficies cargadas, sobre las moléculas de agua es el desplazamiento de estas hacia el centro de la celda de simulación. Las simulaciones con paredes cargadas eléctricamente conducen a un aumento de la densidad hacia el centro de la celda más marcado que el observado en el caso HHFO.

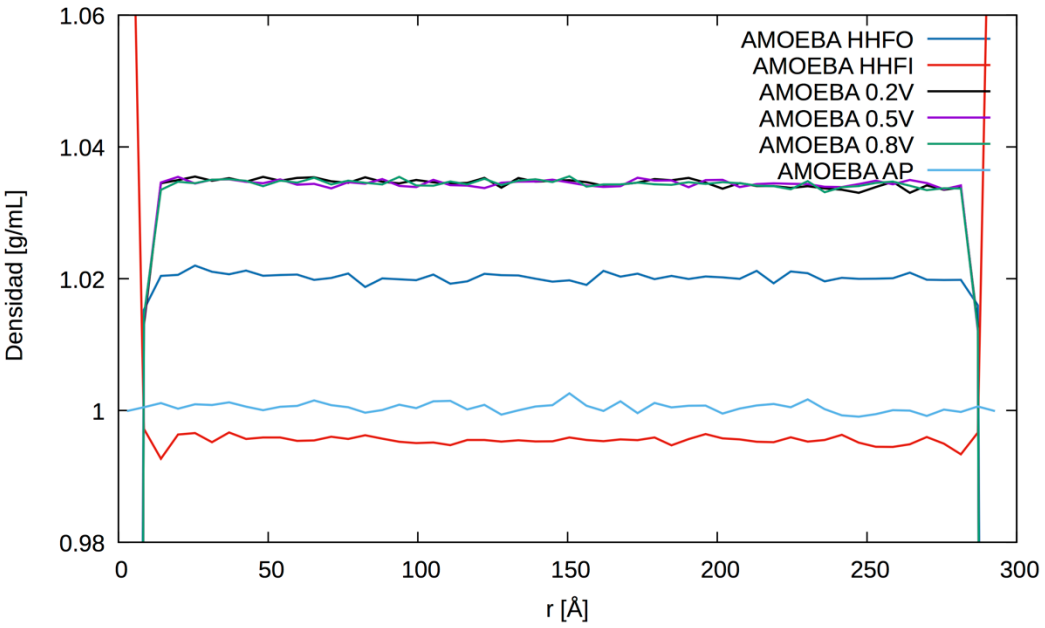

Figura 22 Perfiles de densidad del modelo de agua AMOEBA. El código de colores para identificar cada sistema es: HHFO en azul rey; HHFI en rojo; superficie cargada con diferente potencial: 0.2 V en color negro, 0.5 V en morado y 0.8 V en verde, y el bulto en color azul cielo. La diferencia principal entre las superficies es que la HHFI aumenta más del 6% en el primer bloque y en bloques posteriores la densidad es menor al bulto y sucede lo contrario en las otras superficies, donde la acumulación es en toda la celda.

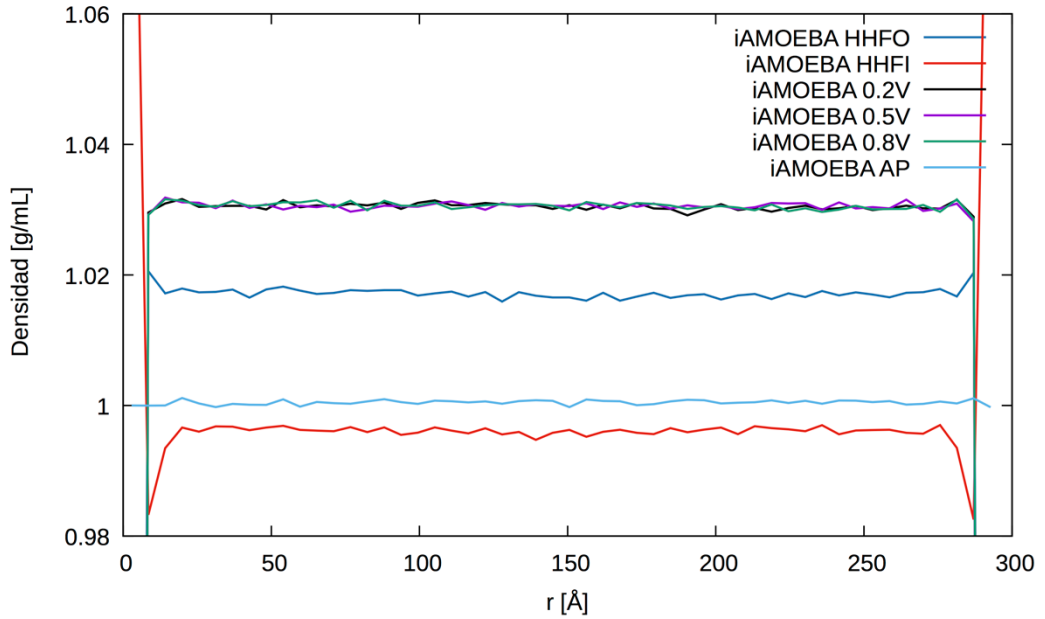

Figura 21 Perfiles de densidad del modelo de agua iAMOEBA. El código de colores corresponde al mencionado en la figura 22. Tanto la superficie HHFI como la superficie HHFO presentan un aumento de la densidad en el primer bloque, aunque en la hidrofóbica es en menor magnitud y se acumula en forma homogénea a lo largo de la celda. En la pared HHFI la densidad es menor que la del bulto a partir del segundo bloque. En las superficies cargadas, lo observado es similar que en el modelo AMOEBA, pero en menor magnitud.

En los tres modelos de agua, se observa que cerca de la superficie hidrofílica hay un aumento en el número de moléculas (densidad mayor 1.06 g/mL), que se traduce en una disminución en la densidad hacia el centro de la celda.

En los casos HFFO y superficies cargadas es posible distinguir diferencias en el comportamiento debidas a la naturaleza de cada uno de los modelos. En las paredes hidrofóbicas la densidad promedio aumenta porcentualmente en 1.3 para TIP4P-FB, 1.7 para iAMOEBA y 2.0 para AMOEBA. Las superficies cargadas incrementan la densidad a lo largo de la celda en 2.8, 3.1 y 3.4% para TIP4P-FB, iAMOEBA y AMOEBA, respectivamente. Por otra parte, los tres modelos confirman la acumulación de agua cerca de la superficie hidrofílica en cantidades muy similares $\approx$10%.

La acumulación cerca de las superficies (electrodos) que el modelo TIP4P-FB presenta, en el caso HHFO parece ser consecuencia de un modelo sin polarización ya que el modelo iAMOEBA lo presenta en menor magnitud, y el modelo con polarización mutua no lo presenta en absoluto.

Las moléculas TIP4P-FB cerca del electrodo actuarían como una pantalla de la superficie hacia el bulto. Al no polarizarse, no se modifica la interacción entre ellas y sólo existe una respuesta estructural a la heterogeneidad cerca del electrodo y así, amortiguan el efecto de éste hacia los vecinos en segundos o terceros bloques.

La polarización de las moléculas conduce a una interacción más fuerte entre ellas (enlaces de hidrógeno más cortos) que resulta en una densidad acumulada importante a lo largo de toda la celda (ver imágenes 18,19 y 20). Esto se ve reflejado en las trayectorias con superficies cargadas, donde el aumento es mayor que en la superficie HHFO (ver figuras 21, 22 y 23).

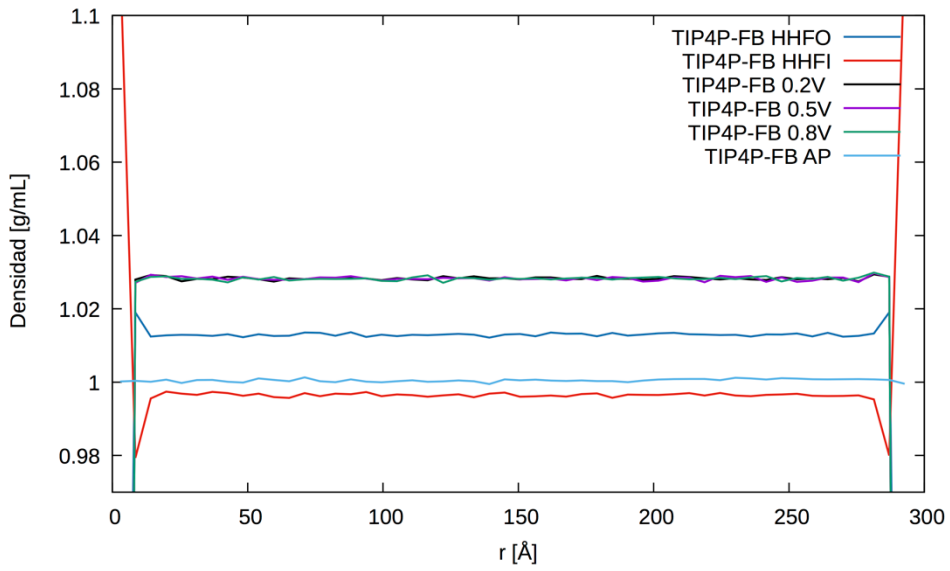

Figura 23 Perfiles de densidad del modelo de agua TIP4P-FB. El código de colores corresponde al mencionado en la figura 22. Tanto la superficie HHFI como la superficie HHFO presentan un comportamiento como él modelo iAMOEBA: mayor densidad en el primer bloque que en los bloques posteriores. Las superficies solo desplazan a las moléculas hacia el centro de la celda.

4.4 Distribución de la orientación del momento dipolar

En la hidratación hidrofóbica (ver la imagen 25), la orientación más probable de las moléculas cercanas a la superficie es aquella en la que el dipolo apunta hacia afuera de la superficie formando un ángulo promedio de 78º, sin embargo, la forma del gráfico sugiere una distribución de orientaciones entre 60 - 100º. Lo mismo sucede en el gráfico pequeño de la figura 25, que corresponde a la distribución del ángulo formado entre el vector de enlace OH y la normal al plano (véase la figura 24) en donde es posible apreciar que el ángulo más probable tiene valores de aproximadamente 100 o 0º. Este efecto disminuye conforme se alejan de la superficie, porque en el bloque número dos (línea a trozos) y tres (línea punteada), el perfil se comporta como el del bulto.

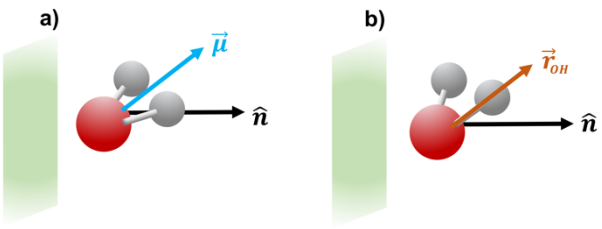

Figura 24 Definición de los vectores de momento dipolar a) y el vector de enlace OH b), que forman el ángulo θ respecto al vector normal a la superficie.

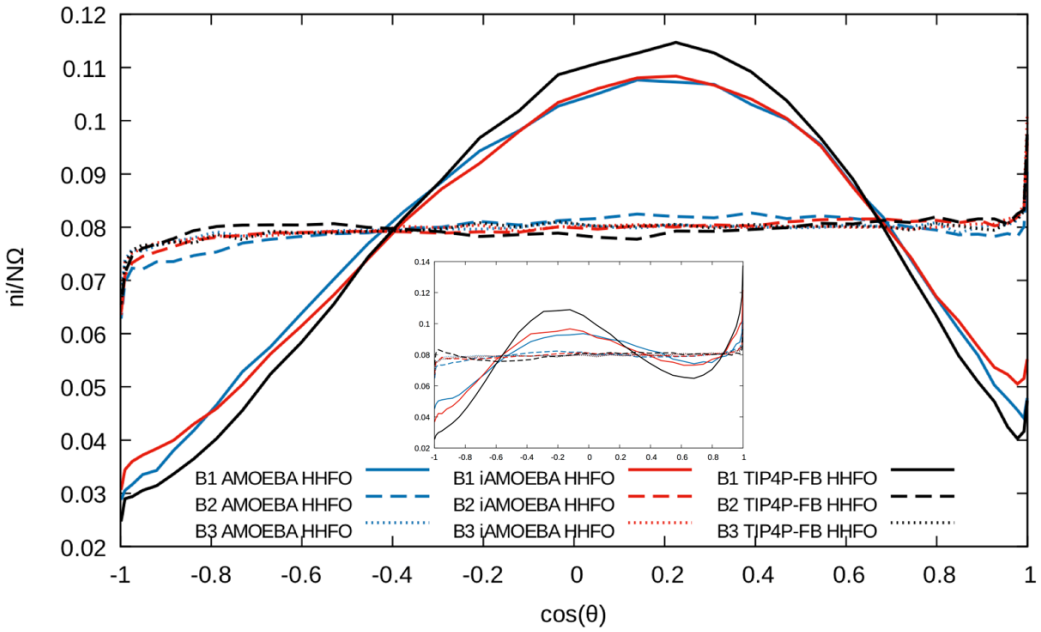

Figura 26 Orientaciones de las moléculas de agua en presencia de una superficie HHFO. Los bloques 1, 2 y 3, se representan con línea continua, línea a trozos y línea punteada, respectivamente. El modelo AMOEBA en color azul, en rojo el iAMOEBA y en negro el TIP4P-FB. En el gráfico más pequeño se presenta la distribución del vector de enlace OH y en el gráfico principal, la distribución para el momento dipolar de la molécula de agua.

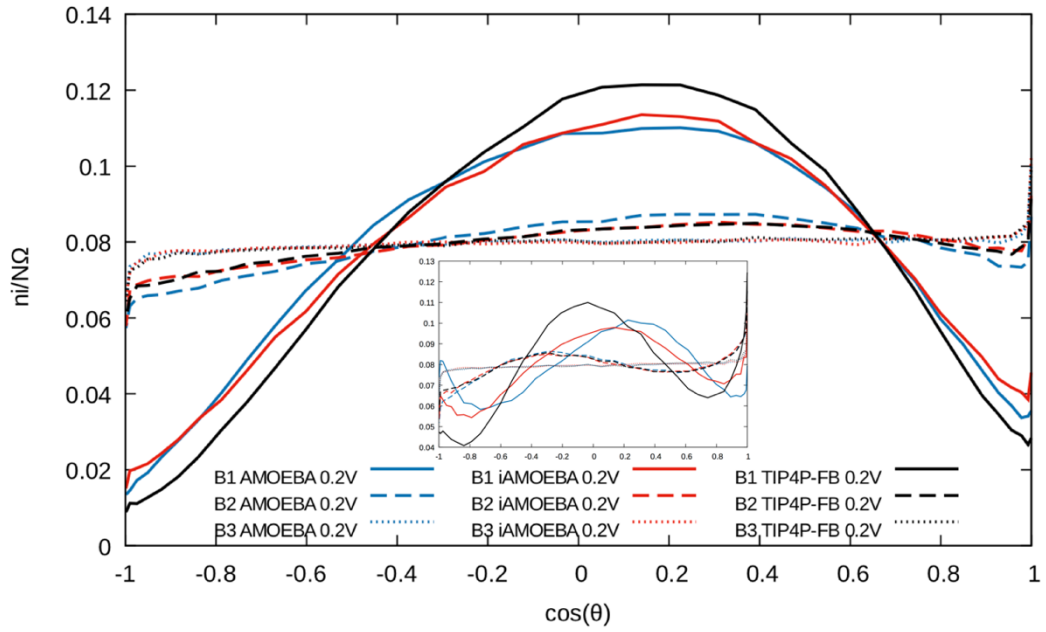

Figura 25 Orientaciones de las moléculas de agua en presencia de superficies con potencial de 0.2 V. Los bloques 1, 2 y 3, se representan con línea continua, línea a trozos y línea punteada, respectivamente. El modelo AMOEBA en color azul, en rojo el iAMOEBA y en negro el TIP4P-FB. En el gráfico más pequeño se presenta la distribución del vector de enlace OH y en el gráfico principal, la distribución para el momento dipolar.

Las figuras 26, 27 y 28 corresponden a los gráficos de las orientaciones del momento dipolar del agua y en los recuadros de cada uno, la orientación del vector de enlace OH. Las tres distribuciones muestran un comportamiento muy similar tanto para la orientación del dipolo del agua como para el vector de enlace OH. Es posible notar que a mayor campo (0.8V) la distribución se desplaza hacia valores ligeramente más grandes de cos(θ) lo que sugeriría un mayor número de moléculas alineadas con el campo. El efecto de las paredes cargadas es de mayor alcance que el de la pared hidrofóbica. Es posible observar que en el caso HHFO, en el bloque 2 se ha recuperado el comportamiento del bulto mientras que en los sistemas sujetos a un campo eléctrico, el perfil de orientaciones del bloque 2 todavía no se comporta como el bulto. Esta diferencia se magnifica en la orientación del vector OH que se muestra en los gráficos más pequeños. En el bloque 1, los modelos flexibles tienen una respuesta completamente opuesta a la del modelo TIP4P-FB, esto se atribuye al tipo de polarización que cada modelo tiene y que está siendo perturbada por las densidades de carga superficial, por ejemplo, los enlaces OH del

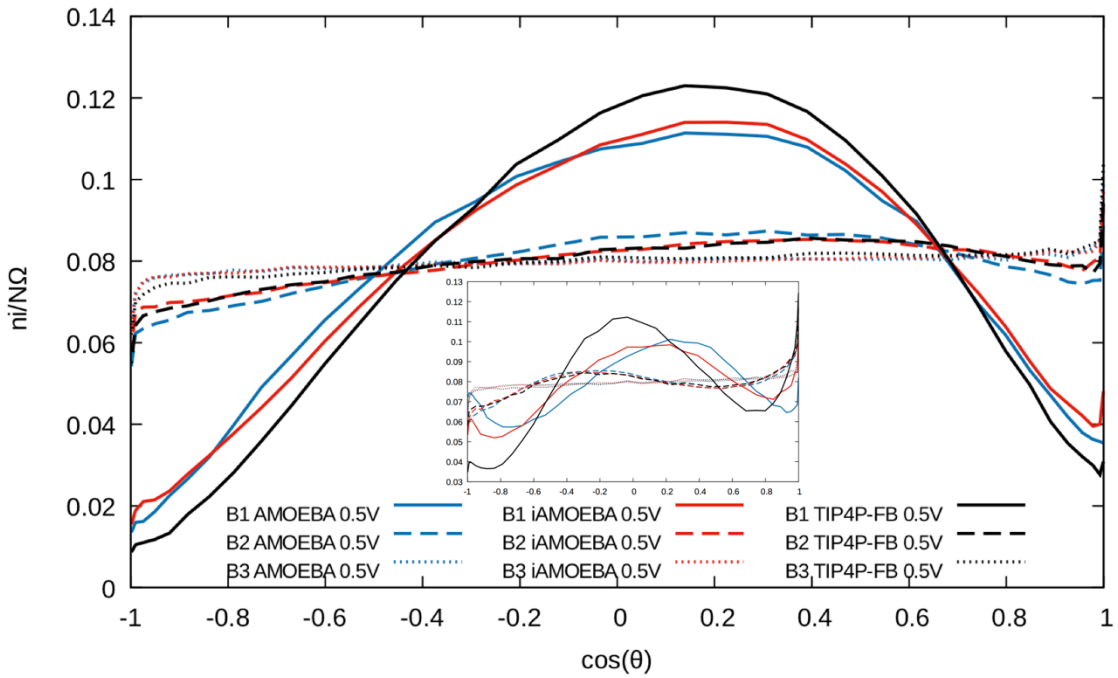

Figura 27 Orientaciones de las moléculas de agua en presencia de una superficie con potencial de 0.5 V. Los bloques 1, 2 y 3, se representan con línea continua, línea a trozos y línea punteada, respectivamente. El modelo AMOEBA en color azul, en rojo el iAMOEBA y en negro el TIP4P-FB. En el gráfico más pequeño se presenta la distribución del vector de enlace OH y en el gráfico principal, la distribución para el momento dipolar de la molécula de agua

modelo con polarización mutua, AMOEBA, se orientan preferentemente hacia fuera de la superficie y esto es menos marcado al utilizar el modelo iAMOEBA.

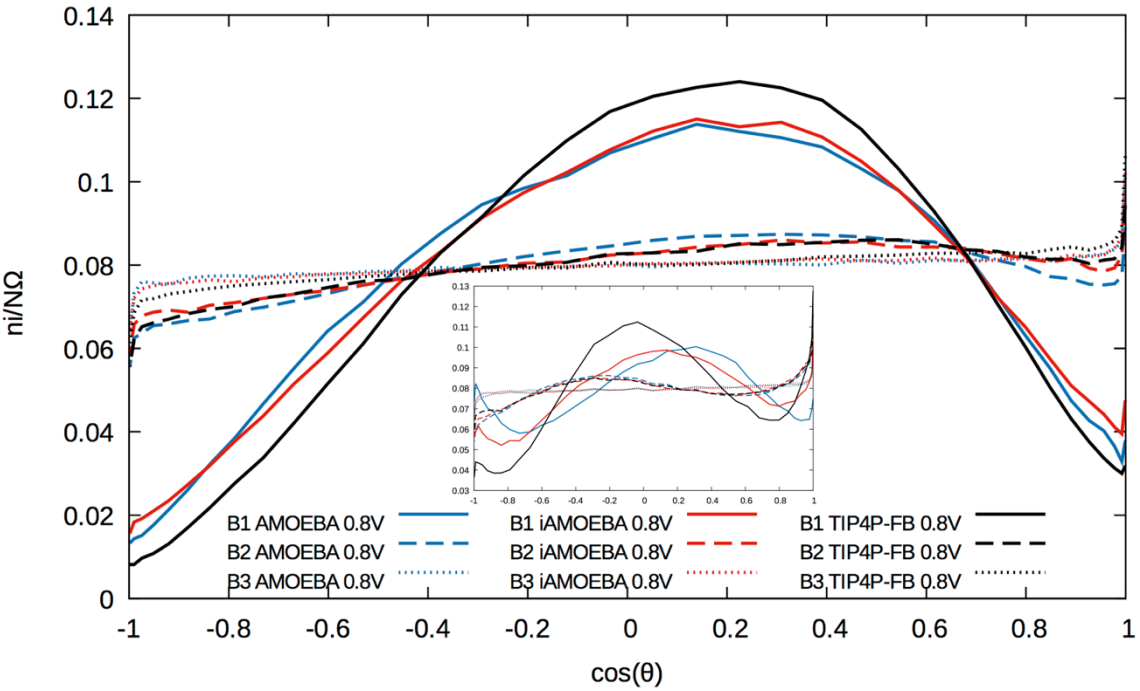

Figura 28 Orientaciones de las moléculas de agua en presencia de una superficie con potencial de 0.8 V. Los bloques 1, 2 y 3, se representan con línea continua, línea a trozos y línea punteada, respectivamente. El modelo AMOEBA en color azul, en rojo el iAMOEBA y en negro el TIP4P-FB. En el gráfico más pequeño se presenta la distribución del vector de enlace OH y en el gráfico principal, la distribución para el momento dipolar de la molécula de agua

La hidratación hidrofílica (Figura 29) es un caso especial, ya que las moléculas cercanas a la superficie tienen dos orientaciones preferenciales. Las distribuciones son angostas, lo que indica que los dipolos están más ordenados en comparación con las superficies anteriores. En el modelo TIP4P-FB, el dipolo está paralelo a la superficie y hay dos posibilidades de la orientación de los hidrógenos: 1) los dos están paralelos al plano o 2), solo uno apunta hacia afuera. También puede suceder que el dipolo apunte hacia afuera de la superficie, es decir, los dos enlaces OH estarían apuntando hacia el segundo bloque.

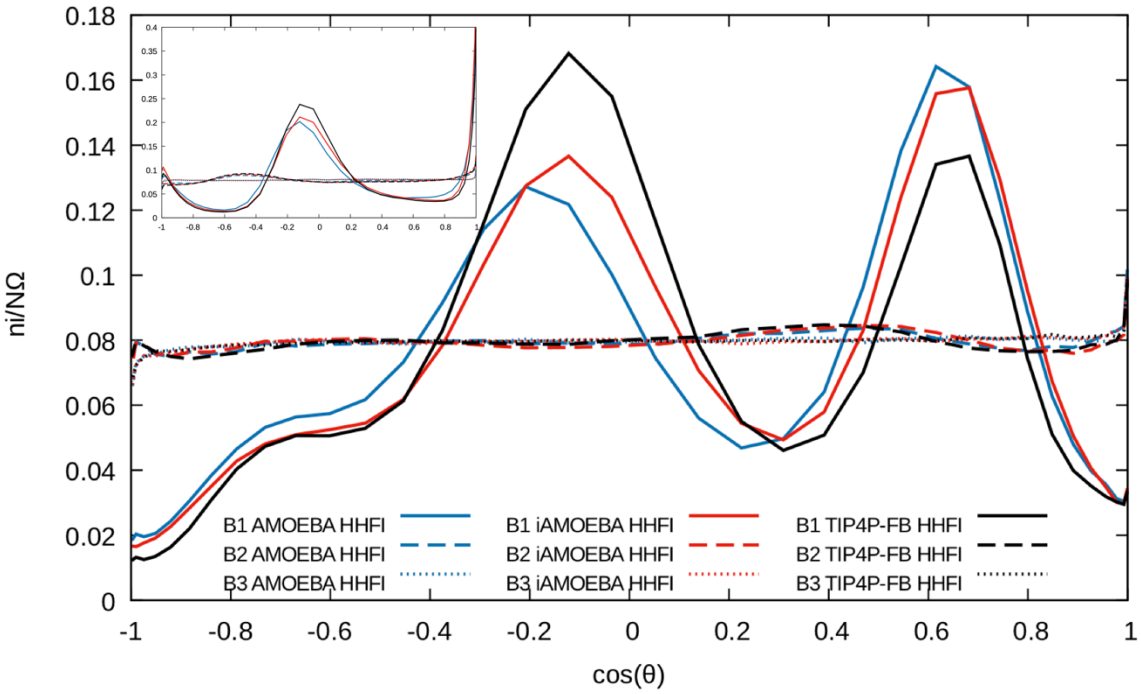

Figura 29 Orientaciones de las moléculas de agua en presencia de una superficie HHFI. Los bloques 1, 2 y 3, se representan con línea continua, línea a trozos y línea punteada, respectivamente. El modelo AMOEBA en color azul, en rojo el iAMOEBA y en negro el TIP4P-FB. En el gráfico más pequeño se presenta la distribución del vector de enlace OH y en el gráfico principal, la distribución para el momento dipolar de la molécula de agua

En los modelos flexibles sucede lo contrario, es más probable encontrar dipolos apuntando hacia fuera de la superficie, con un OH perpendicular al plano y el otro casi paralelo al plano. Las moléculas que aparecen en el segundo bloque, líneas a trazos, no tienen un comportamiento de bulto y podría suponerse que se debe a su papel como moléculas de intercambio, es decir, son aquellas que entran y salen de la primera capa de solvatación de la superficie.

4.5 Coeficiente de auto-difusión

El coeficiente de auto-difusión fue calculado para la celda con cada uno de los modelos de superficie. Se observa que en los tres potenciales de agua la diferencia porcentual es del 62 al 70%; esto indica que las superficies se oponen al desplazamiento de las moléculas en la dirección $\boldsymbol{Y}$, por lo que no pueden moverse como en el bulto porque el volumen disponible es menor que en agua pura.

MODELO DE AGUA	D Bulto [cm^2/s]	D HHFO [cm^2/s]	D HHFI [cm^2/s]	D 0.2 V [cm^2/s]	D 0.5 V [cm^2/s]	D 0.8 V [cm^2/s]
AMOEBA	$2.0x10^{-5}$	7.7 (-62)	7.7 (-62)	7.6 (-62)	7.7 (-62)	7.8 (-61)
iAMOEBA	$2.2x10^{-5}$	7.4 (-63)	7.1 (-65)	7.1 (-65)	7.3 (-64)	7.5 (-63)
TIP4P-FB	$2.2x10^{-5}$	6.3 (-70)	6.5 (-69)	6.5 (-69)	6.5 (-69)	6.6 (-69)

Tabla 10 Coeficiente de auto-difusión de agua líquida y de los tipos de superficies: hidrofóbica (HHFO); hidrofílica (HHFI); cargada eléctricamente con potenciales de 0.2 V, 0.5 V y 0.8 V. Entre paréntesis, se muestra la diferencia porcentual entre el bulto y el agua entre las superficies. Todos los valores de las superficies deben multiplicarse por el factor 1×10^{-6}.

4.6 Constante Dieléctrica

La respuesta dieléctrica depende de la fluctuación del momento dipolar del sistema. En la tabla 11 se muestran los valores de la constante dieléctrica siguiendo la metodología propuesta por Neumann. De acuerdo con los análisis de orientaciones y del perfil de densidad, se observa que el efecto generado por las superficies hacia el disolvente es de confinamiento. Los dipolos moleculares tienen menos libertad de movimiento en comparación con el bulto, es decir, oscilan menos; esto se traduce en tener dipolos alineados con las superficies y, por lo tanto, la ε es menor que en el bulto (ver imágenes 29, 30 y 31).

Por un lado, en los tres modelos con superficies HHFO o HHFI, se observa que la resistividad del medio disminuye del 23 al 26% en comparación con el bulto. Por el otro, en las superficies cargadas hay una relación directa entre el campo generado y la constante dieléctrica, es decir, a menor voltaje la ε es más pequeña que a campos mayores. Los modelos flexibles presentan este comportamiento, pero siempre están por debajo del valor para el bulto. A diferencia del modelo TIP4P-FB, que en la superficie con 0.8 V la ε es mayor que en el bulto (+3%).

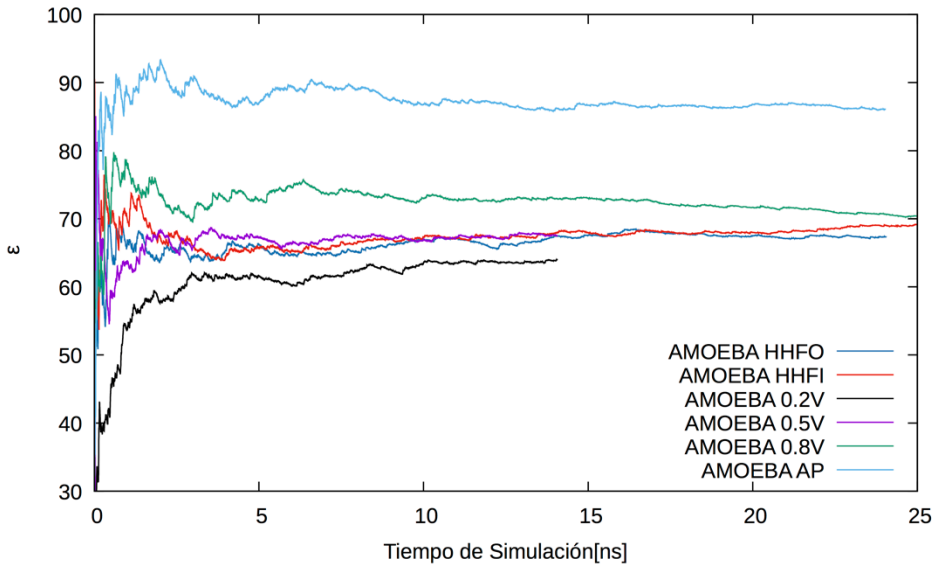

Figura 30 Constante dieléctrica del modelo AMOEBA contra el tiempo de simulación. En color azul rey se muestra el promedio acumulado de la ε en el sistema con superficie hidrofóbica; en rojo la hidrofílica; en negro la superficie con potencial 0.2 V; en morado con 0.5 V y en verde 0.8 V. En color azul se grafica la ε para el agua bulto.

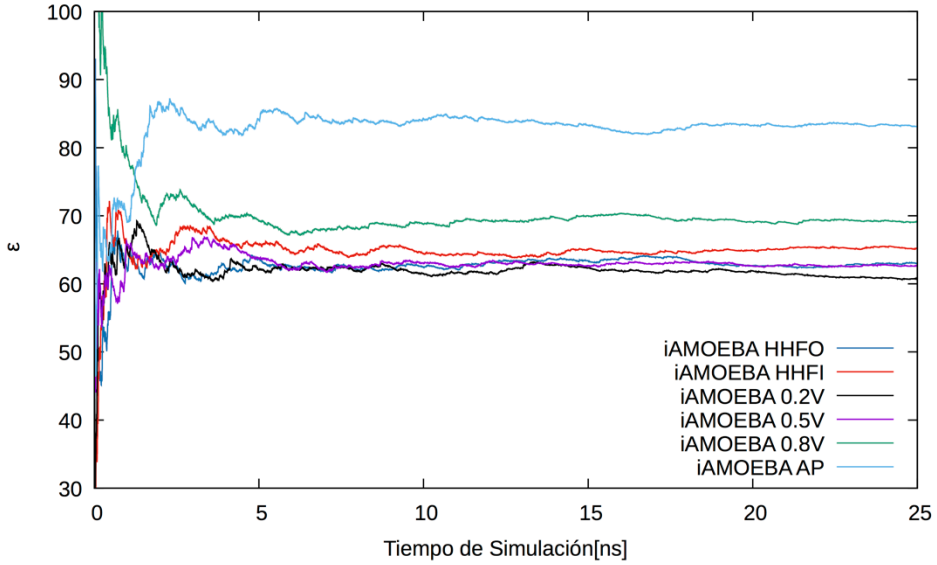

Figura 31 Promedio acumulado de la constante dieléctrica del modelo iAMOEBA contra el tiempo de simulación. El código de colores es el siguiente: en color azul rey para el sistema con superficie hidrofóbica; en rojo la hidrofílica; en negro con potencial de 0.2 V; en morado con 0.5 V y en verde 0.8 V. En color azul cielo se grafica la ε para el agua bulto.

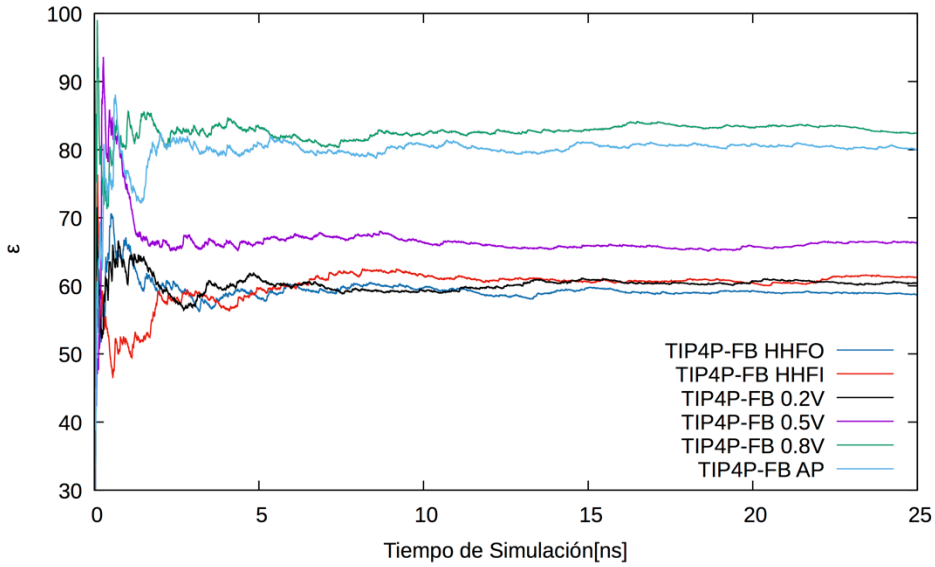

Figura 32 Constante dieléctrica del modelo TIP4P-FB contra el tiempo de simulación. El código de colores es el siguiente: en color azul rey el promedio acumulado de la ε en el sistema con superficie hidrofóbica; en rojo la hidrofílica; en negro con potencial de 0.2 V; en morado con 0.5 V y en verde, 0.8 V. La ε del agua bulto se grafica en color azul cielo.

MODELO DE AGUA	ε Bulto	ε HHFO	ε HHFI	ε 0.2 V	ε 0.5 V	ε 0.8 V
AMOEBA	86.7 ± 0.5	66.7 ± 0.5 (-23.1)	67.4 ± 0.4 (-22.3)	63.5 ± 0.5 (-26.8)	67.3 ± 0.3 (-22.4)	72.9 ± 0.2 (-15.9)
iAMOEBA	83.9 ± 0.4	63.0 ± 0.5 (-24.9)	64.5 ± 0.4 (-23.)	62.1 ± 0.6 (-26.0)	62.9 ± 0.2 (-25.0)	69.2 ± 0.4 (-17.5)
TIP4P-FB	80.2 ± 0.5	59.2 ± 0.5 (-26.2)	61.1 ± 0.4 (-23.8)	59.9 ± 0.6 (-25.3)	66.0 ± 0.5 (-17.7)	82.6 ± 0.2 (+3.0)

Tabla 11 Constante dieléctrica del bulto; las superficies hidrofóbica (HHFO) e hidrofílica (HHFI) y las superficies cargadas por un potencial de 0.2 V, 0.5 V y 0.8 V. La diferencia porcentual entre el valor calculado para la superficie y el bulto esta entre paréntesis. Los valores mostrados se obtuvieron en el lapso de 10 a 15 *ns* en los promedios acumulados de la constante dieléctrica.

4.7 Mapas Estructurales

En la siguiente imagen, se muestran los vecinos que hidratan a la superficie a una distancia máxima de 5 *Å* de ella: a) hidrofóbica y b) hidrofílica (línea en color azul). (ver anexo 7.26).

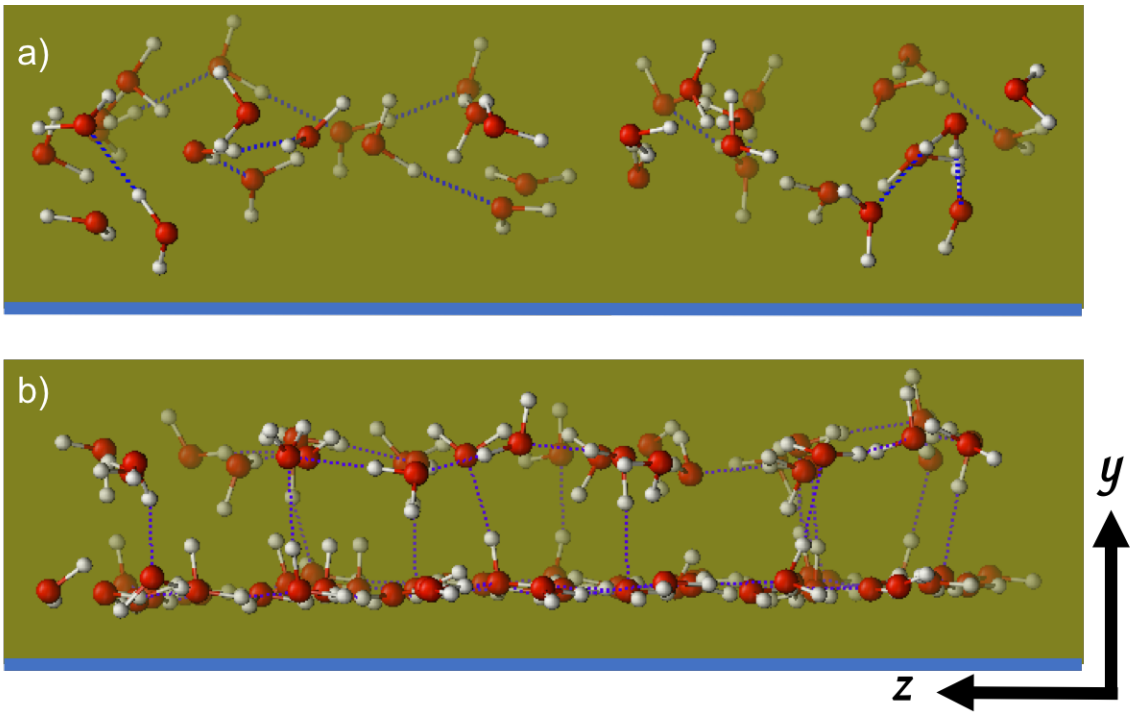

Figura 33 Distribución de las moléculas de agua cerca de la superficie (línea en color azul): a) superficie hidrofóbica y b) superficie hidrofílica.

En la imagen anterior (imágenes obtenidas con VMD[112]), se observa que en la superficie hidrofílica hay una primera capa de vecinos bien definida, se formó una red de enlaces de hidrógeno entre primeros vecinos, que presentan una orientación tipo *"dangling proton*", con una segunda capa. En 2017, Willard *et al.* encontraron que el cambio de moléculas de agua entre la primera y segunda capa es un evento raro, ya que las aguas en primera capa permanecen allí durante casi 40 *ns*[113]. En la siguiente imagen (34) se muestra que, en la superficie hidrofílica (gráfico del lado derecho), las moléculas de agua permanecen algunos pares de *ns* en la primera capa de solvatación, sin embargo, los tiempos son menores en comparación con lo

reportado por Willard. Lo anterior se atribuye a dos cosas: 1) el modelo de electrodo propuesto tiene estructura y 2) la fase sólida responde a la fase líquida y viceversa. Por el contrario, en la superficie hidrofóbica (gráfico del lado izquierdo), el cambio entre las moléculas de primera capa y capas subsecuentes sucede con mayor frecuencia.

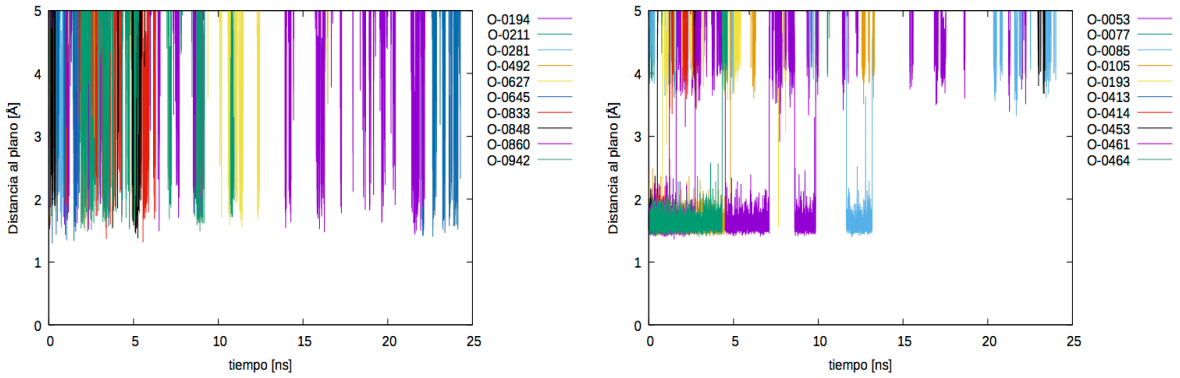

Figura 34 Distancia a la pared contra el tiempo de simulación. Se grafican las distancias de diez átomos de oxígeno que están en la primera configuración de toda la trayectoria. En la gráfica del lado izquierdo, se muestran las distancias a la superficie hidrofóbica y del lado derecho, las distancias a la superficie hidrofílica. Los oxígenos de cada molécula son diferentes para cada superficie.

En los modelos de electrodo implementados en este proyecto no se considera la respuesta ante la fase líquida.

En las figuras 35 y 36, se muestra la probabilidad de encontrar una molécula de agua cerca de las caras $\boldsymbol{X}$, $\boldsymbol{Z}$ de la celda de simulación. La gráfica de la izquierda corresponde al análisis hecho para la superficie en el plano π_1 y la imagen de la derecha para el plano π_2 (ver sección 3.3). Las caras se dividieron en una malla de 38x38 y se consideró la posición de los oxígenos que están a una distancia menor de 3 Å de la superficie, es decir, aquellas moléculas que pertenecen a la primera capa de hidratación (ver anexo 7.25). En la figura 35, se observa que en los tres modelos de agua hay regiones donde es más probable encontrar moléculas de agua cerca de la superficie. Esto se atribuye a características intrínsecas del potencial, es decir, es la propia respuesta de cada potencial para minimizar el efecto de las paredes sobre la fase líquida.

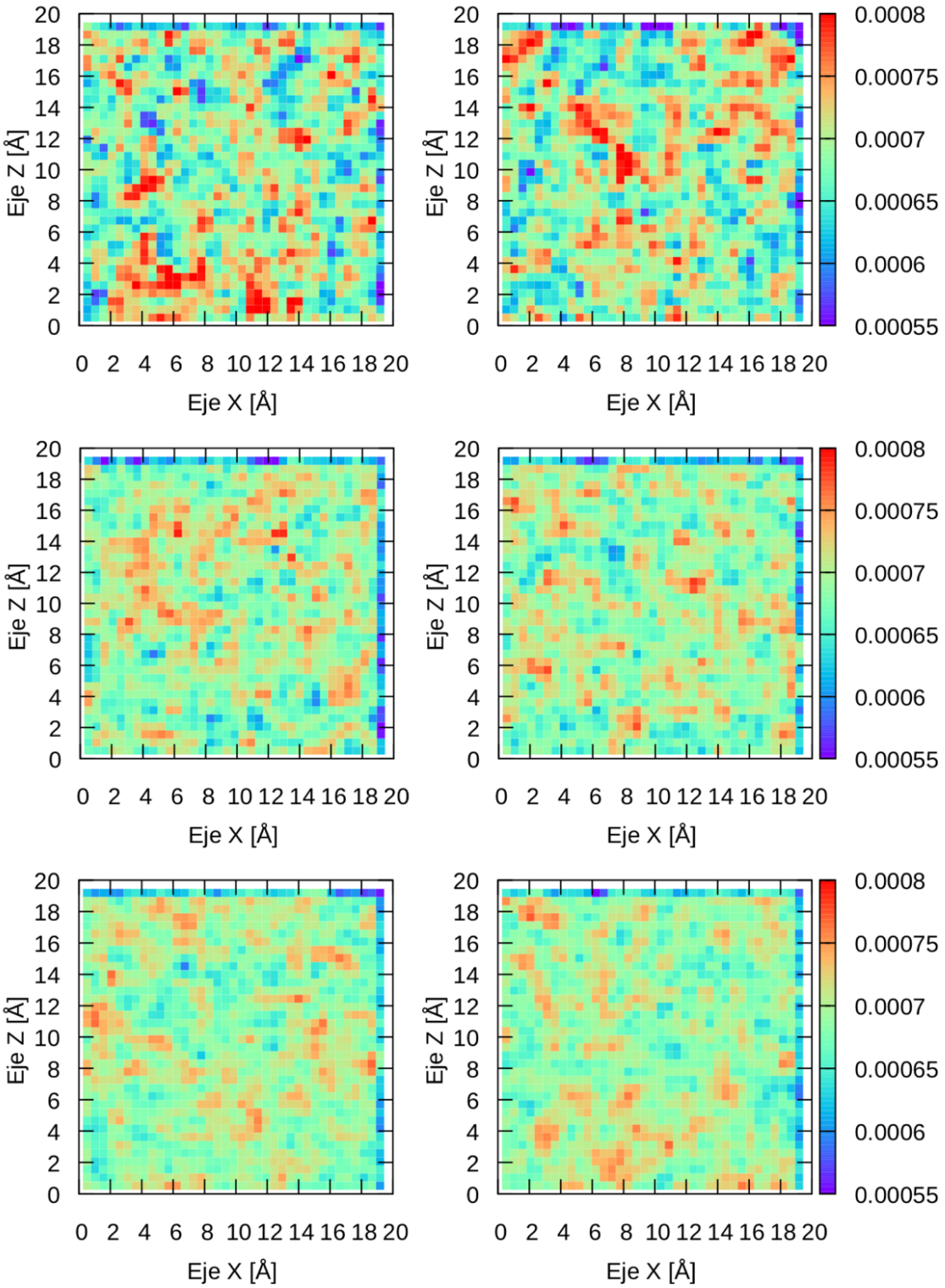

Figura 35 Mapas bidimensionales que muestran la estructura del agua líquida ante las superficies hidrofóbicas. El gráfico del lado izquierdo corresponde al análisis hecho para la superficie en $y = 0$ y del lado derecho la superficie en $y = l_y$. Los gráficos superiores corresponden al modelo AMOEBA, en el medio al modelo iAMOEBA y los inferiores al TIP4P-FB.

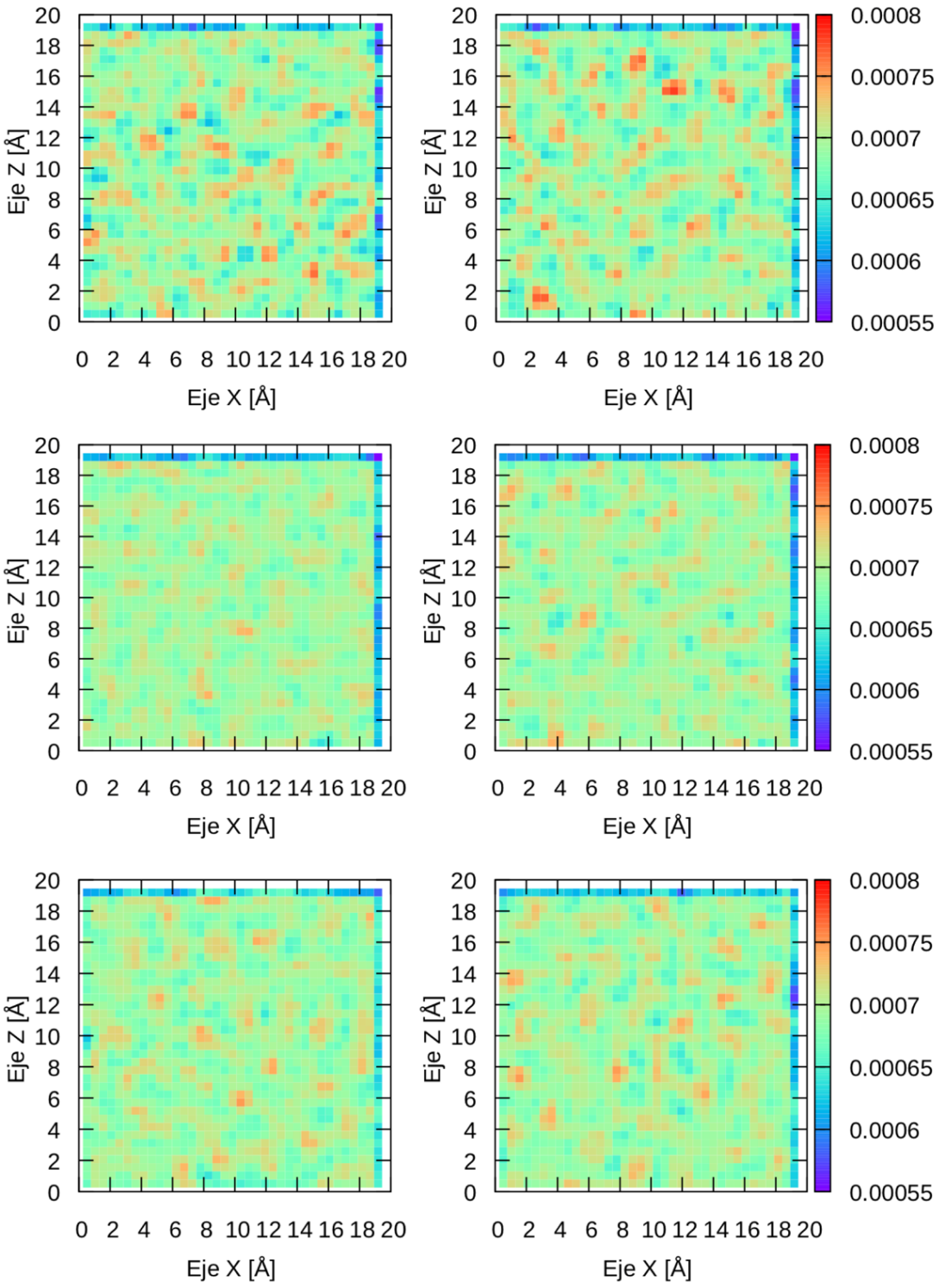

Figura 36 Mapas bidimensionales que muestran la estructura del agua líquida ante las superficies hidrofílicas. El gráfico del lado izquierdo corresponde al análisis hecho para la superficie en $y = 0$ y del lado derecho la superficie en $y = l_y$. Los gráficos superiores corresponden al modelo AMOEBA, en el medio al modelo iAMOEBA y los inferiores al TIP4P-FB.

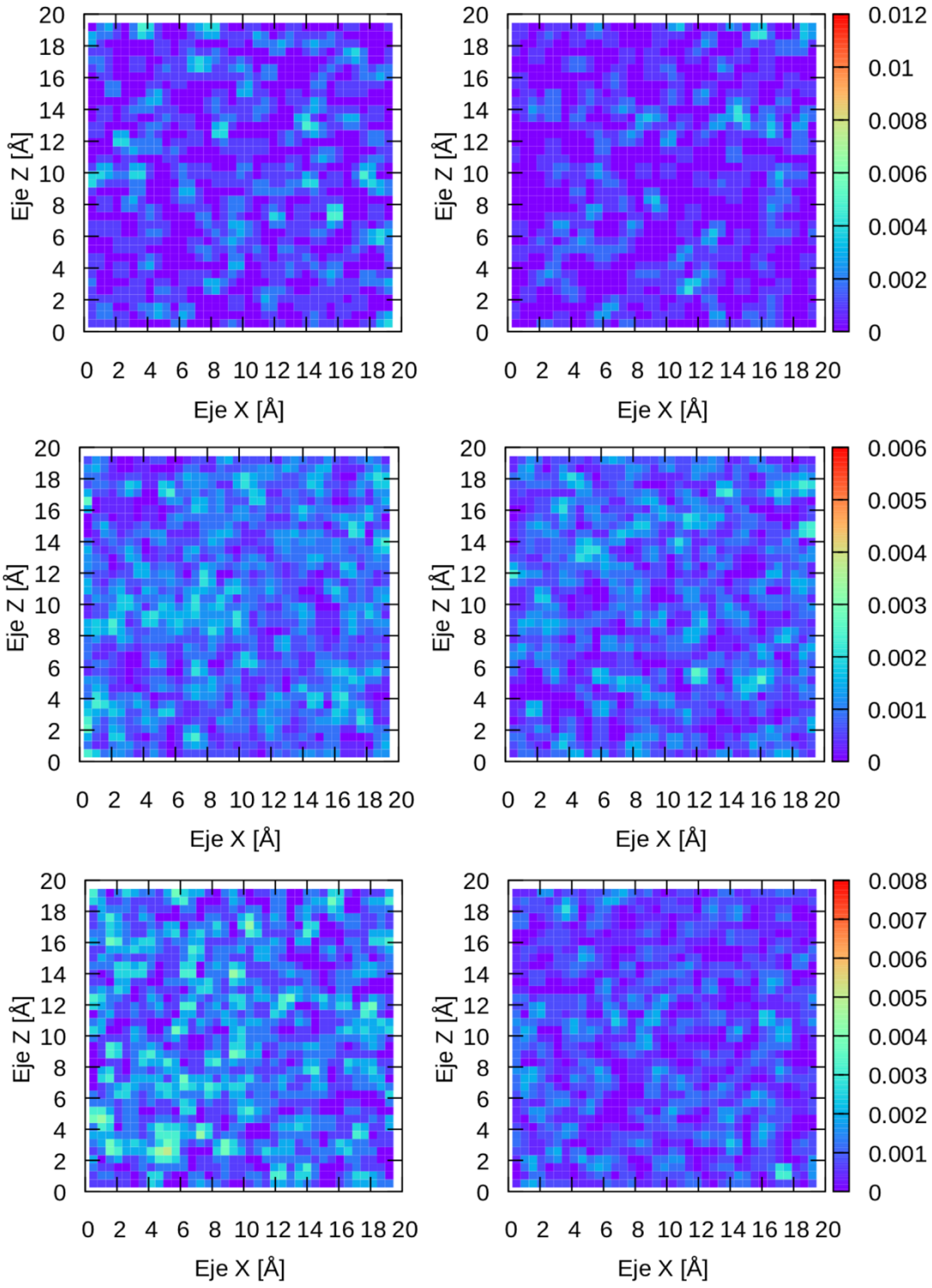

Figura 37 Mapas bidimensionales que muestran la estructura del agua líquida ante las superficies cargadas con el potencial de 0.5 V. El gráfico del lado izquierdo corresponde al análisis hecho para la superficie en $y = 0$ y del lado derecho la superficie en $y = l_y$. Los gráficos superiores corresponden al modelo AMOEBA, en el medio al modelo iAMOEBA y los inferiores al TIP4P-FB.

En las figuras 36 y 37, se observa que hay zonas predominantes donde están las aguas, pero los patrones no son semejantes entre sí. En los mapas de la superficie cargada se muestra que las probabilidades son menores en comparación con la superficie HHFI y la HHFO, esto se debe a que hay un menor número de aguas en la distancia máxima de análisis que se consideró.

En los tres mapas bidimensionales mostrados se puede decir que, las zonas con mayor probabilidad contiguas forman enlaces de hidrógeno bidimensionales, sin embargo, este análisis no es suficiente para mostrarlo. Por ejemplo, Bellarosa *et al.* estudiaron por DM la interfase metal-agua y caracterizaron la estructura en primera capa aplicando el método de Voronoi (triangulación de puntos); en esa región encontraron que las moléculas de agua forman polígonos irregulares de entre 4 y 7 lados[114].

CAPÍTULO CINCO

CONCLUSIONES

5. CONCLUSIONES

La metodología resultante de este proyecto se alcanzó después de numerosas pruebas y tiene las siguientes ventajas:

- La forma en que se implementan los potenciales de interacción de la superficie con el disolvente en OpenMM favorece la prueba de distintas expresiones.
- La versatilidad de los modelos de agua implementados es importante en los casos en los que se desea conocer el efecto de algún grado de libertad particular sobre el fenómeno de estudio.

En el caso particular que estudiamos:

- Se han determinado los tiempos necesarios para alcanzar el equilibrio de sistemas ante superficies con diferentes características.
- Se obtuvieron configuraciones iniciales, de los sistemas de estudio, donde diferentes propiedades han alcanzado el equilibrio.

El análisis de los diferentes modelos de agua ante superficies (con características diferentes de solvatación) mostró lo siguiente:

- La hidratación de la superficie puede ser de tipo hidrofóbico, donde las moléculas de agua son desplazadas hacia el seno de la disolución y la hidratación hidrofílica, que compite contra la red tridimensional de enlaces de hidrógeno, porque genera una primera capa de vecinos bien definida y el efecto de este orden influye en la formación de la red de enlaces de hidrógeno para capas posteriores y en la respuesta global medida con diferentes propiedades.
- La respuesta del disolvente, en este caso analizada como función de la distancia a la superficie sólida muestra que, la FDR por celda no refleja cambios significativos en comparación con la FDR del bulto; esto se debe a que el efecto de la pared es de corto alcance (<13 *Å*) y no tiene suficiente peso en el promedio estadístico, sin embargo, se observaron diferencias sutiles.

- La energía necesaria para cambiar de fase líquida a gas, en el sistema con superficies hidrofílicas, requiere más del 1.4% de energía comparada con el bulto y con las otras paredes. Entre las paredes opuestas de la celda se reduce el volumen disponible en 1.86% y, la respuesta estructural depende de las características del modelo de agua. Los modelos rígidos requieren de un radio efectivo cercano al valor del enlace O-H; al disminuir el volumen disponible aumenta la densidad en el centro de la celda, pero el efecto de acumulación depende del tipo de superficie. Tanto la superficie HHFO como la HHFI, muestran que el aumento de densidad es más pronunciado en zonas cercanas a la superficie; sin embargo, las superficies cargadas generan un efecto únicamente de desplazamiento. Por el contrario, los modelos flexibles responden homogéneamente a la acumulación. Lo anterior es menos marcado para la superficie HHFI, porque la primera capa amortigua el efecto de la pared y en capas subsecuentes, la densidad es menor a la del bulto.
- Debido a que la pared HHFI impone la primera capa, hay dos orientaciones preferenciales en esta región: 1) la orientación donde las moléculas crean una red bidimensional de enlaces de hidrógeno (más probable para TIP4P-FB) y, 2) donde las aguas se orientan hacia afuera de la superficie y favorecen la formación de la red tridimensional (más probable en los modelos polarizables). No hay una orientación del dipolo más probable en las superficies cargadas y la HHFO, ya que los picos son más anchos y hay más orientaciones disponibles. Sin embargo, se cree que la orientación de los hidrógenos es el resultado de considerar la interacción de vdW y una descripción rigurosa de la polarización en los modelos de agua. Por lo tanto, los estudios previos hechos con modelos rígidos que no consideran el tratamiento de la polarización, no tienen una descripción completa de lo que sucede en la interfase agua-sólido.
- Las superficies restringen el movimiento de las moléculas en una dirección y esa restricción es tan marcada que conduce a un valor promedio de toda la celda un 70% menos. Tanto la disminución del movimiento y del volumen

efectivo como la polarización de los modelos de agua generan fluctuaciones menores en la orientación de las moléculas. La respuesta dieléctrica del medio es $\approx$ 20% menor que en el bulto, es decir, permiten que la corriente eléctrica fluya más rápido en sistemas con superficies que en agua bulto.

En sistemas de agua ante superficies, estudiados por DM, no es posible, todavía, hacer un análisis por bloque para cualquier propiedad debido a la forma en que se calculan y la absoluta necesidad de contar con suficiente estadística para que el cálculo tenga sentido. Sin embargo, la metodología y las herramientas de análisis desarrolladas (ver sección 7) sirven para caracterizar con detalle molecular la estructura y dinámica de interfases acuosas.

CAPÍTULO SEIS

BIBLIOGRAFÍA

6. BIBLIOGRAFÍA

1. Butt, H.-J., Graf, K. & Kappl, M. *Física y Química de Interfases*. (2003).
2. Edwards, M., Triantafyllidou, S. & Best, D. Elevated Blood Lead in Young Children Due to Lead-Contaminated Drinking Water: Washington, DC, 2001−2004. *Environ. Sci. Technol.* **43,** 1618–1623 (2009).
3. Kesavan, D., Gopiraman, M. & Sulochana, N. Green inhibitors for corrosion of metals: A review. *Chem. Sci. Rev. Lett* **1,** 1–8 (2012).
4. Rani, B. E. A. & Basu, B. B. J. Green Inhibitors for Corrosion Protection of Metals and Alloys: An Overview. *Int. J. Corros.* **2012,** 1–15 (2012).
5. Conway, B. E. Transition from “Supercapacitor” to “Battery” Behavior in Electrochemical Energy Storage. *J. Electrochem. Soc.* **138,** 1539 (1991).
6. Harnett, C. K., Templeton, J., Dunphy-Guzman, K. A., Senousy, Y. M. & Kanouff, M. P. Model based design of a microfluidic mixer driven by induced charge electroosmosis. *Lab Chip* **8,** 565 (2008).
7. Liu, Y., Wiek, A., Dzhagan, V. & Holze, R. Improved Electrochemical Behavior of Amorphous Carbon-Coated Copper/CNT Composites as Negative Electrode Material and Their Energy Storage Mechanism. *J. Electrochem. Soc.* **163,** A1247–A1253 (2016).
8. Damaskin, B. B. & Petrii, O. A. Historical development of theories of the electrochemical double layer. *J. Solid State Electrochem.* **15,** 1317–1334 (2011).
9. Schmickler, W. Electronic Effects in the Electric Double Layer. 3177–3200 (1996).
10. Bockris, J. O. & Reddy, A. K. N. *Electroquímica Moderna*. (Editorial Reverté S.A., 2003).
11. Taylor, C. D. & Neurock, M. Theoretical insights into the structure and reactivity of the aqueous/metal interface. *Curr. Opin. Solid State Mater. Sci.* **9,**

49–65 (2005).

12. Wilson, M. A., Nguyen, T. H. & Pohorille, A. Combining molecular dynamics and an electrodiffusion model to calculate ion channel conductance. *J. Chem. Phys.* **141,** (2014).

13. Spohr, E. Some recent trends in computer simulations of aqueous double layers. **49,** 23–27 (2003).

14. Yeh, K.-Y., Janik, M. J. & Maranas, J. K. Molecular dynamics simulations of an electrified water/Pt(111) interface using point charge dissociative water. *Electrochim. Acta* **101,** 308–325 (2013).

15. Reymond, F., Fermín, D., Lee, H. J. & Girault, H. H. Electrochemistry at liquid/liquid interfaces: methodology and potential applications. *Electrochim. Acta* **45,** 2647–2662 (2000).

16. Guidelli, R. & Schmickler, W. Recent developments in models for the interface between a metal and an aqueous solution. *Electrochim. Acta* **45,** 2317–2338 (2000).

17. Shin, S. & Willard, A. P. Characterizing Hydration Properties Based on the Orientational Structure of Interfacial Water Molecules. *J. Chem. Theory Comput.* **14,** 461–465 (2018).

18. Schnur, S. & Groß, A. Properties of metal–water interfaces studied from first principles. *New J. Phys.* **11,** 125003 (2009).

19. Li, P. & Merz, K. M. Metal Ion Modeling Using Classical Mechanics. *Chem. Rev.* **117,** 1564–1686 (2017).

20. Guillot, B. A reappraisal of what we have learnt during three decades of computer simulations on water. *J. Mol. Liq.* **101,** 219–260 (2002).

21. Onufriev, A. V. & Izadi, S. Water models for biomolecular simulations. *Wiley Interdiscip. Rev. Comput. Mol. Sci.* e1347 (2017). doi:10.1002/wcms.1347

22. Saint-Martin, H., Hernández-Cobos, J., Bernal-Uruchurtu, M. I., Ortega-Blake, I. & Berendsen, H. J. C. A mobile charge densities in harmonic oscillators

(MCDHO) molecular model for numerical simulations: The water–water interaction. *J. Chem. Phys.* **113,** 10899–10912 (2000).

23. Ren, P. & Ponder, J. W. Polarizable Atomic Multipole Water Model for Molecular Mechanics Simulation. *J. Phys. Chem. B* **107,** 5933–5947 (2003).

24. Pauling, L. *General chemistry*. (W. H.Freeman and Company, 1988).

25. Bernal, J. D. & Fowler, R. H. A Theory of Water and Ionic Solution, with Particular Reference to Hydrogen and Hydroxyl Ions. *J. Chem. Phys.* **1,** 515–548 (1933).

26. Willard, A. P., Reed, S. K., Madden, P. A. & Chandler, D. Water at an electrochemical interface—a simulation study. *Faraday Discuss.* **141,** 423–441 (2009).

27. Jedlovszky, P., Předota, M. & Nezbeda, I. Hydration of Apolar Solutes of Varying Size: a Systematic Study. *Mol. Phys.* **104,** 2465–2476 (2006).

28. Guymon, C. G., Rowley, R. L., Harb, J. N. & Wheeler, D. R. Simulating an electrochemical interface using charge dynamics. **8,** 335–356 (2005).

29. Chang, R. & College, W. *Química*. (Mc Graw Hill, 2002).

30. Kohlmeyer, A., Hartnig, C. & Spohr, E. Orientational correlations near interfaces. Computer simulations of water and electrolyte solutions in confined environments. *J. Mol. Liq.* **78,** 233–253 (1998).

31. Björneholm, O. *et al.* Water at Interfaces. *Chem. Rev.* **116,** 7698–7726 (2016).

32. Lee, C., McCammon, J. A. & Rossky, P. J. The structure of liquid water at an extended hydrophobic surface. *J. Chem. Phys.* **80,** 4448–4455 (1984).

33. Stillinger, F. H. & Rahman, A. Improved simulation of liquid water by molecular dynamics. *J. Chem. Phys.* **60,** 1545–1557 (1974).

34. Marcus, Y. Effect of Ions on the Structure of Water: Structure Making and Breaking. *Chem. Rev.* **109,** 1346–1370 (2009).

35. Spohr, E. & Heinzinger, K. Molecular dynamics simulation of a water/metal

interface. *Chem. Phys. Lett.* **123,** 218–221 (1986).

36. Kong, C. L. Combining rules for intermolecular potential parameters. II. Rules for the Lennard-Jones (12–6) potential and the Morse potential. *J. Chem. Phys.* **59,** 2464–2467 (1973).

37. Jorgensen, W. L., Chandrasekhar, J., Madura, J. D., Impey, R. W. & Klein, M. L. Comparison of simple potential functions for simulating liquid water. *J. Chem. Phys.* **79,** 926–935 (1983).

38. Halicioglu, T. & Pound, G. M. Calculation of potential energy parameters form crystalline state properties. *Phys. Status Solidi* **30,** 619–623 (1975).

39. Spohr, E. Computer simulation of the water/platinum interface. *J. Phys. Chem.* **93,** 6171–6180 (1989).

40. Bopp, P., Jancsó, G. & Heinzinger, K. An improved potential for non-rigid water molecules in the liquid phase. *Chem. Phys. Lett.* **98,** 129–133 (1983).

41. Hautman, J., Halley, J. W. & Rhee, Y. -J. Molecular dynamics simulation of water beween two ideal classical metal walls. *J. Chem. Phys.* **91,** 467–472 (1989).

42. Toukan, K. & Rahman, A. Molecular-dynamics study of atomic motions in water. *Phys. Rev. B* **31,** 2643–2648 (1985).

43. Spohr, E. Molecular simulation of the electrochemical double layer. *Electrochim. Acta* **44,** 1697–1705 (1999).

44. Bernal-Uruchurtu, M. I. & Ortega-Blake, I. A refined Monte Carlo study of Mg 2+ and Ca 2+ hydration. *J. Chem. Phys.* **103,** 1588–1598 (1995).

45. Mao, Y., Demerdash, O., Head-Gordon, M. & Head-Gordon, T. Assessing Ion–Water Interactions in the AMOEBA Force Field Using Energy Decomposition Analysis of Electronic Structure Calculations. *J. Chem. Theory Comput.* **12,** 5422–5437 (2016).

46. Martínez, J. M., Pappalardo, R. R. & Sánchez Marcos, E. First-Principles Ion−Water Interaction Potentials for Highly Charged Monatomic Cations.

Computer Simulations of Al 3+ , Mg 2+ , and Be 2+ in Water. *J. Am. Chem. Soc.* **121,** 3175–3184 (1999).

47. Jedlovszky, P., Vincze, A. & Horvai, G. New insight into the orientational order of water molecules at the water/1,2-dichloroethane interface: A Monte Carlo simulation study. *J. Chem. Phys.* **117,** 2271 (2002).

48. Wang, L.-P. *et al.* Systematic Improvement of a Classical Molecular Model of Water. *J. Phys. Chem. B* **117,** 9956–9972 (2013).

49. Fuentes-Azcatl, R. & Alejandre, J. Non-polarizable force field of water based on the dielectric constant: TIP4P/ε. *J. Phys. Chem. B* **118,** 1263–72 (2014).

50. Wang, L.-P., Martinez, T. J. & Pande, V. S. Building Force Fields: An Automatic, Systematic, and Reproducible Approach. *J. Phys. Chem. Lett.* **5,** 1885–1891 (2014).

51. Izadi, S., Anandakrishnan, R. & Onufriev, A. V. Building Water Models: A Different Approach. *J. Phys. Chem. Lett.* **5,** 3863–3871 (2014).

52. Xiao, S., Figge, F., Stirnemann, G., Laage, D. & McGuire, J. A. Orientational Dynamics of Water at an Extended Hydrophobic Interface. *J. Am. Chem. Soc.* **138,** 5551–5560 (2016).

53. Duboué-Dijon, E., Fogarty, A. C., Hynes, J. T. & Laage, D. Dynamical Disorder in the DNA Hydration Shell. *J. Am. Chem. Soc.* **138,** 7610–7620 (2016).

54. Laage, D., Elsaesser, T. & Hynes, J. T. Water Dynamics in the Hydration Shells of Biomolecules. *Chem. Rev.* acs.chemrev.6b00765 (2017). doi:10.1021/acs.chemrev.6b00765

55. Abascal, J. L. F., Sanz, E., García Fernández, R. & Vega, C. A potential model for the study of ices and amorphous water: TIP4P/Ice. *J. Chem. Phys.* **122,** 234511 (2005).

56. Kiss, P. T. & Baranyai, A. Sources of the deficiencies in the popular SPC/E and TIP3P models of water. *J. Chem. Phys.* **134,** 054106 (2011).

57. Jensen, F. *Introduction to Computational Chemistry*. (John Wiley & Sons, Inc,

2007).

58. McQuarrie, D. A. *Statistical Thermodynamics*. (Harper & Row, 1973).

59. Young, D. C. *Computational Chemistry: a Practical Guide for Applying Techniques to Real-World Problems*. (John Wiley & Sons, Inc, 2001).

60. Frenkel, D. & Smit, B. *Understanding Molecular Simulation from Algorithms to Application*. (Academic Press, 2002).

61. Grant, G. H. & Richards, W. G. *Computational Chemistry*. (Oxford Science Publication, 1998).

62. Pathria, R. K. *Statistical Mechanics*. (A. Wheaton & Co. Ltd., 1986).

63. Leach, A. R. *Molecular Modelling Principles and Applications*. (Pearson Educación S. A., 2001).

64. Allen, M. P. & Tildesley, D. J. *Computer simulation of liquids*. (Oxford University Press, 1991).

65. Beale, P. D. & Pathria, R. K. *Statistical Mechanics*. (Elsevier, 2011).

66. Schlick, T. *Molecular Modeling and Simulation: an Interdisciplinary Guide*. (Springer, 2010).

67. Tuckerman, M. E. *Statistical Mechanics: Theory and Molecular Simulation*. (Oxford University Press Inc., 2010).

68. Horn, H. W. *et al.* Development of an improved four-site water model for biomolecular simulations: TIP4P-Ew. *J. Chem. Phys.* **120,** 9665–9678 (2004).

69. Kaplan, I. G. *Intermolecular Interactions: Physical Picture, Computational Methods and Model Potentials*. (John Wiley & Sons, Ltd, 2006).

70. Goodman, J. M. *Chemical Applications of Molecular Modelling*. (The Royal Society of Chemistry, 1998).

71. Halgren, T. A. Potential Energy Functions. *Curr. Opin. Struct. Biol.* **5,** 205–210 (1995).

72. Darden, T., York, D. & Pedersen, L. Particle mesh Ewald: An N ·log(N)

method for Ewald sums in large systems. *J. Chem. Phys.* **98,** 10089–10092 (1993).

73. Essmann, U. *et al.* A smooth particle mesh Ewald method. *J. Chem. Phys.* **103,** 8577–8593 (1995).

74. Vega, C. & Abascal, J. L. F. Simulating water with rigid non-polarizable models: a general perspective. *Phys. Chem. Chem. Phys.* **13,** 19663 (2011).

75. Laury, M. L., Wang, L.-P., Pande, V. S., Head-Gordon, T. & Ponder, J. W. Revised Parameters for the AMOEBA Polarizable Atomic Multipole Water Model. *J. Phys. Chem. B* **119,** 9423–9437 (2015).

76. Abascal, J. L. F. & Vega, C. A general purpose model for the condensed phases of water: TIP4P/2005. *J. Chem. Phys.* **123,** 234505 (2005).

77. Article, C. Simulating water with rigid non-polarizable models : a general perspective w. **13,** (2011).

78. Wang, L.-P., Chen, J. & Van Voorhis, T. Systematic Parametrization of Polarizable Force Fields from Quantum Chemistry Data. *J. Chem. Theory Comput.* **9,** 452–460 (2013).

79. Wagner, W. & Pruß, A. The IAPWS Formulation 1995 for the Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use. *J. Phys. Chem. Ref. Data* **31,** 387–535 (2002).

80. Fuentes-Azcatl, R. & Alejandre, J. Non-Polarizable Force Field of Water Based on the Dielectric Constant: TIP4P/ε. *J. Phys. Chem. B* **118,** 1263–1272 (2014).

81. Halgren, T. A. The representation of van der Waals (vdW) interactions in molecular mechanics force fields: potential form, combination rules, and vdW parameters. *J. Am. Chem. Soc.* **114,** 7827–7843 (1992).

82. Lacava, F. *Classical Electrodynamics*. (Springer International Publishing, 2016). doi:10.1007/978-3-319-39474-9

83. Shi, Y. *et al.* Polarizable Atomic Multipole-Based AMOEBA Force Field for Proteins. *J. Chem. Theory Comput.* **9,** 4046–4063 (2013).

84. Hugh D., Y. & Freedman, R. A. *Física universitaria, con física moderna volumen 2*. (Pearson Educación S. A., 2009).

85. Tipler, P. A. & Mosca, G. *Physics for Scientists and Engineers*. (W. H. Freeman and Company, 2008).

86. Eastman, P. *et al.* OpenMM 4: A Reusable, Extensible, Hardware Independent Library for High Performance Molecular Simulation. *J. Chem. Theory Comput.* **9,** 461–469 (2013).

87. Eastman, P. *et al.* OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. *PLOS Comput. Biol.* **13,** e1005659 (2017).

88. Lindert, S., Bucher, D., Eastman, P., Pande, V. & McCammon, J. A. Accelerated Molecular Dynamics Simulations with the AMOEBA Polarizable Force Field on Graphics Processing Units. *J. Chem. Theory Comput.* **9,** 4684–4691 (2013).

89. Eastman, P. & Pande, V. Accelerating Development and Execution Speed with Just-in-Time GPU Code Generation. in *GPU Computing Gems Jade Edition* 399–407 (Elsevier, 2012). doi:10.1016/B978-0-12-385963-1.00029-0

90. Friedrichs, M. S. *et al.* Accelerating molecular dynamic simulation on graphics processing units. *J. Comput. Chem.* **30,** 864–872 (2009).

91. Eastman, P. & Pande, V. S. Efficient nonbonded interactions for molecular dynamics on a graphics processing unit. *J. Comput. Chem.* NA-NA (2009). doi:10.1002/jcc.21413

92. Wang, J. & Hou, T. Application of Molecular Dynamics Simulations in Molecular Property Prediction. 1. Density and Heat of Vaporization. *J. Chem. Theory Comput.* **7,** 2151–2165 (2011).

93. Caldwell, J. W. & Kollman, P. A. Structure and Properties of Neat Liquids Using Nonadditive Molecular Dynamics: Water, Methanol, and N-Methylacetamide. *J. Phys. Chem.* **99,** 6208–6219 (1995).

94. Abascal, J. L. F. & Vega, C. A general purpose model for the condensed

phases of water: TIP4P/2005. *J. Chem. Phys.* **123,** 234505 (2005).

95. Berendsen, H. J. C., Grigera, J. R. & Straatsma, T. P. The missing term in effective pair potentials. *J. Phys. Chem.* **91,** 6269–6271 (1987).

96. Leontyev, I. V. & Stuchebrukhov, A. A. Electronic Polarizability and the Effective Pair Potentials of Water. *J. Chem. Theory Comput.* **6,** 3153–3161 (2010).

97. Hinchliffe, A. *Molecular Modelling for Beginners*. (John Wiley & Sons, Ltd, 2013).

98. Haile, J. M. *Molecular Dynamics Simulation: elementary methods*. (John Wiley & Sons, Inc, 1992).

99. Ferrara, C. G. & Grigera, T. S. Dynamics and structural behavior of water in large confinement with planar amorphous walls. *J. Chem. Phys.* **147,** 024705 (2017).

100. Zlenko, D. V. Computing the self-diffusion coefficient for TIP4P water. *Biophysics (Oxf).* **57,** 127–132 (2012).

101. Izadi, S., Anandakrishnan, R. & Onufriev, A. V. Building Water Models : A Di ff erent Approach. (2014).

102. Fuentes-Azcatl, R., Mendoza, N. & Alejandre, J. Improved SPC force field of water based on the dielectric constant: SPC/?? *Phys. A Stat. Mech. its Appl.* **420,** 116–123 (2015).

103. Elton, D. C. & Fernández-Serra, M.-V. Polar nanoregions in water: A study of the dielectric properties of TIP4P/2005, TIP4P/2005f and TTM3F. *J. Chem. Phys.* **140,** 124504 (2014).

104. Joshi, R. P., Qian, J., Schoenbach, K. H. & Schamiloglu, E. Microscopic analysis for water stressed by high electric fields in the prebreakdown regime. *J. Appl. Phys.* **96,** 3617–3625 (2004).

105. Neumann, M. Dipole moment fluctuation formulas in computer simulations of polar systems. *Mol. Phys.* **50,** 841–858 (1983).

106. Abeyrathne, C. D., Halgamuge, M. N., Farrell, P. M. & Skafidas, E. An ab-initio Computational Method to Determine Dielectric Properties of Biological Materials. *Sci. Rep.* **3,** 1796 (2013).

107. Neumann, M. The dielectric constant of water. Computer simulations with the MCY potential. *J. Chem. Phys.* **82,** 5663–5672 (1985).

108. Neumann, M. Dielectric relaxation in water. Computer simulations with the TIP4P potential. *J. Chem. Phys.* **85,** 1567–1580 (1986).

109. Wu, Y., Tepper, H. L. & Voth, G. A. Flexible simple point-charge water model with improved liquid-state properties. *J. Chem. Phys.* **124,** 024503 (2006).

110. Simonson, T. & Perahia, D. Internal and Interfacial Dielectric Properties of Cytochrome c from Molecular Dynamics in Aqueous Solution. *Natl. Acad. Sci.* **92,** 1082–1086 (1995).

111. Simonson, T. Accurate calculation of the dielectric constant of water from simulations of a microscopic droplet in vacuum. *Chem. Phys. Lett.* **250,** 450–454 (1996).

112. Humphrey, W., Dalke, A. & Schulten, K. VMD - Visual Molecular Dynamics. 33–38 (1996).

113. Limmer, D. T., Willard, A. P., Madden, P. A. & Chandler, D. Water Exchange at a Hydrated Platinum Electrode is Rare and Collective. *J. Phys. Chem. C* **119,** 24016–24024 (2015).

114. Bellarosa, L., García-Muelas, R., Revilla-López, G. & López, N. Diversity at the Water–Metal Interface: Metal, Water Thickness, and Confinement Effects. *ACS Cent. Sci.* **2,** 109–116 (2016).

CAPÍTULO SIETE

ANEXOS

7 ANEXOS

7.1 Script de python para calcular dinámicas moleculares en OpenMM

```
from simtk.openmm.app import *
from simtk.openmm import *
from simtk.unit import *
import os
import sys
from simtk.openmm.app.clasesReporteros import *
from simtk.openmm.app.funcionesMomentosDip import *
# Parameters
#ffield amoeba2013.xml, iamoeba.xml, tip4pfb.xml
#en consola: python nvt.py tip4pfb.xml HHFO 0.002 500 25000
forceField = sys.argv[1]
Hidratacion = sys.argv[2]
switch_width = 1.0 * angstroms
cutoff = 9.0 * angstroms
use_switch = True
timestep= float(sys.argv[3])
temperatura=298.15
stepImp= int(sys.argv[4])
pasosTotales=int(sys.argv[5])
computer = sys.argv[6]
volt = float(sys.argv[7])
elec = 1*elementary_charge
e = elec.value_in_unit(coulombs)

#Seleccion tipo de hidratacion y configuracion de inicio
if Hidratacion == "HHFO":
    if computer == "local":
        pdb = PDBFile('/home/anthoni/Documents/AGUA_OPENMM/ESFERA/INPUTS_CONF_INICIAL/HHFO_1micros.pdb')
    elif computer == "yoltla":
        pdb = PDBFile('/LUSTRE/home/lancad/2018/mabel/alumnos/toni/INPUTS/HHFO_1micros.pdb')
    elif computer == "abacus":
        pdb = PDBFile('/lustre/home/CE-2-2017/EISL/aalcaraz/scratch/INPUTS/HHFO_1micros.pdb')
    else:
        computer = ""
elif Hidratacion == "HHFI":
    if computer == "local":
        pdb = PDBFile('/home/anthoni/Documents/AGUA_OPENMM/ESFERA/INPUTS_CONF_INICIAL/HHFI_1micros.pdb')
    elif computer == "yoltla":
        pdb = PDBFile('/LUSTRE/home/lancad/2018/mabel/alumnos/toni/INPUTS/HHFI_1micros.pdb')
    elif computer == "abacus":
        pdb = PDBFile('/lustre/home/CE-2-2017/EISL/aalcaraz/scratch/INPUTS/HHFI_1micros.pdb')
    else:
        computer = ""
elif Hidratacion == "CHARGE":
#0.8, 0.5 y 0.2 son las DM que estan equilibradas 500 ns
    if computer == "local":
        if volt == 0.8 or volt == 0.5 or volt == 0.2:
            fileinput = '/home/anthoni/Documents/AGUA_OPENMM/ESFERA/INPUTS_CONF_INICIAL/{0}v.pdb'.format(volt)
        else:
            fileinput = '/home/anthoni/Documents/AGUA_OPENMM/ESFERA/INPUTS_CONF_INICIAL/0.5v.pdb'
        pdb = PDBFile(fileinput)
    elif computer == "yoltla":
        if volt == 0.8 or volt == 0.5 or volt == 0.2:
```

7 ANEXOS

7.1 Script de python para calcular dinámicas moleculares en OpenMM

```
from simtk.openmm.app import *
from simtk.openmm import *
from simtk.unit import *
import os
import sys
from simtk.openmm.app.clasesReporteros import *
from simtk.openmm.app.funcionesMomentosDip import *
# Parameters
#ffield amoeba2013.xml, iamoeba.xml, tip4pfb.xml
#en consola: python nvt.py tip4pfb.xml HHFO 0.002 500 25000
forceField = sys.argv[1]
Hidratacion = sys.argv[2]
switch_width = 1.0 * angstroms
cutoff = 9.0 * angstroms
use_switch = True
timestep= float(sys.argv[3])
temperatura=298.15
stepImp= int(sys.argv[4])
pasosTotales=int(sys.argv[5])
computer = sys.argv[6]
volt = float(sys.argv[7])
elec = 1*elementary_charge
e = elec.value_in_unit(coulombs)

#Seleccion tipo de hidratacion y configuracion de inicio
if Hidratacion == "HHFO":
    if computer == "local":
        pdb = PDBFile('/home/anthoni/Documents/AGUA_OPENMM/ESFERA/INPUTS_CONF_INICIAL/
    HHFO_1micros.pdb')
    elif computer == "yoltla":
        pdb = PDBFile('/LUSTRE/home/lancad/2018/mabel/alumnos/toni/INPUTS/HHFO_1micros.
    pdb')
    elif computer == "abacus":
        pdb = PDBFile('/lustre/home/CE-2-2017/EISL/aalcaraz/scratch/INPUTS/HHFO_1micros.
    pdb')
    else:
        computer = ""
elif Hidratacion == "HHFI":
    if computer == "local":
        pdb = PDBFile('/home/anthoni/Documents/AGUA_OPENMM/ESFERA/INPUTS_CONF_INICIAL/
    HHFI_1micros.pdb')
    elif computer == "yoltla":
        pdb = PDBFile('/LUSTRE/home/lancad/2018/mabel/alumnos/toni/INPUTS/HHFI_1micros.
    pdb')
    elif computer == "abacus":
        pdb = PDBFile('/lustre/home/CE-2-2017/EISL/aalcaraz/scratch/INPUTS/HHFI_1micros.
    pdb')
    else:
        computer = ""
elif Hidratacion == "CHARGE":
#0.8, 0.5 y 0.2 son las DM que estan equilibradas 500 ns
    if computer == "local":
        if volt == 0.8 or volt == 0.5 or volt == 0.2:
            fileinput = '/home/anthoni/Documents/AGUA_OPENMM/ESFERA/INPUTS_CONF_INICIAL
    /{0}v.pdb'.format(volt)
        else:
            fileinput = '/home/anthoni/Documents/AGUA_OPENMM/ESFERA/INPUTS_CONF_INICIAL/0
    .5v.pdb'
        pdb = PDBFile(fileinput)
    elif computer == "yoltla":
        if volt == 0.8 or volt == 0.5 or volt == 0.2:
```

```
54              fileinput = '/LUSTRE/home/lancad/2018/mabel/alumnos/toni/INPUTS/{0}v.pdb'.
    format(volt)
55          else:
56              fileinput = '/LUSTRE/home/lancad/2018/mabel/alumnos/toni/INPUTS/0.5v.pdb'
57          pdb = PDBFile(fileinput)
58      elif computer == "abacus":
59          if volt == 0.8 or volt == 0.5 or volt == 0.2:
60              fileinput = '/lustre/home/CE-2-2017/EISL/aalcaraz/scratch/INPUTS/{0}v.pdb'.
    format(volt)
61          else:
62              fileinput = '/lustre/home/CE-2-2017/EISL/aalcaraz/scratch/INPUTS/0.5v.pdb'
63          pdb = PDBFile(fileinput)
64      else:
65          computer = ""
66  elif Hidratacion == "AGP":
67      if computer == "local":
68          pdb = PDBFile('/home/anthoni/Documents/AGUA_OPENMM/ESFERA/INPUTS_CONF_INICIAL/
    AguaPura100ns.pdb')
69      elif computer == "yoltla":
70          pdb = PDBFile('/LUSTRE/home/lancad/2018/mabel/alumnos/toni/INPUTS/AguaPura100ns.
    pdb')
71      elif computer == "abacus":
72          pdb = PDBFile('/lustre/home/CE-2-2017/EISL/aalcaraz/scratch/INPUTS/AguaPura100ns.
    pdb')
73      else:
74          computer = ""
75  else:
76      print "La hidratacion no fue seleccionada, la eleccion es: HHFO, HHFI,CHARGE o AGP"
77      sys.exit(0)
78
79  #Seleccion del campo de fuerzas
80  forcefield = ForceField(forceField)
81  modeller = Modeller(pdb.topology, pdb.positions)
82  modeller.addExtraParticles(forcefield)
83
84  #Crear Sistema dependiendo del campo de fuerzas
85  if forceField == "tip4pfb.xml":
86      system = forcefield.createSystem(modeller.topology, nonbondedMethod=PME,
    nonbondedCutoff=0.7*nanometer,vdwCutoff=0.9*nanometer, constraints=HBonds, rigidWater
    =True)
87      chargeH = 0.5258681106763*e
88      chargeO = -1.0517362213526*e
89    # se aplica la carga de la particula extra(PE) en el oxigeno, porque no influye en el
    calculo del potencial, solo se va calculando de las posiciones de O, e H's
90  elif forceField == "amoeba2013.xml":
91      system = forcefield.createSystem(modeller.topology, nonbondedMethod=PME,
    nonbondedCutoff=0.7*nanometer,vdwCutoff=0.9*nanometer, constraints=None, rigidWater=
    False,polarization='mutual',mutualInducedTargetEpsilon=0.00001)
92      chargeO = -0.51966*e
93      chargeH = 0.25983*e
94  elif forceField == "iamoeba.xml":
95      system = forcefield.createSystem(modeller.topology, nonbondedMethod=PME,
    nonbondedCutoff=0.7*nanometer,vdwCutoff=0.9*nanometer, constraints=None, rigidWater=
    False,polarization='direct')
96      chargeO = -0.594024*e
97      chargeH = 0.297012*e
98  else:
99      print "El potencial seleccionado no tiene un sistema creado"
100     sys.exit(0)
101
102 #Potencial Externo que depende del tipo de hidratacion
103 if Hidratacion == "CHARGE":
104     boxVector=(1.97056,30.55846,1.97056)
105 #Potencial tipo hidrofobico para barrera fisica
106     force1 = CustomExternalForce('79.7884*exp(-12.4225*r);r=y')
107     force2 = CustomExternalForce('79.7884*exp(-12.4225*r2);r2=29.5584-y')
```

```
#Potencial de superficie cargada
    forceO = CustomExternalForce('f*{0}*(ly-2*y);f=volt*(6.022E20)/(2*ly);ly=29.5584;volt
    ={1}'.format(chargeO,volt))
    forceH = CustomExternalForce('f*{0}*(ly-2*y);f=volt*(6.022E20)/(2*ly);ly=29.5584;volt
    ={1}'.format(chargeH,volt))
    system.addForce(force1)
    system.addForce(force2)
    system.addForce(forceO)
    system.addForce(forceH)

    for i in range(system.getNumParticles()):
        force1.addParticle(i,[])
        force2.addParticle(i,[])
        at2 = system.getParticleMass(i).value_in_unit(dalton)
        if at2 >= 1.007 and at2 <= 1.009:
            #print 'Es hidrogeno'
            forceH.addParticle(i,[])
        elif at2 >= 15.0 and at2 <= 16.0:
            #print 'Es Oxigeno'
            forceO.addParticle(i,[])
        else:
            nada = 0
elif Hidratacion == "HHFO":
    boxVector=(1.97056,30.55846,1.97056)
    force1 = CustomExternalForce('a*exp(-b*r)-c*exp(-d*r)+e*exp(-f*r);a=126763;b=88.7494;
    c=-119802;d=89.3316;e=128810;f=88.7093;r=y')
    force2 = CustomExternalForce('a*exp(-b*r)-c*exp(-d*r)+e*exp(-f*r);a=126763;b=88.7494;
    c=-119802;d=89.3316;e=128810;f=88.7093;r=ly-y;ly=29.5584')
    system.addForce(force1)
    system.addForce(force2)

    for i in range(system.getNumParticles()):
        force1.addParticle(i,[])
        force2.addParticle(i,[])
elif Hidratacion == "HHFI":
    boxVector=(1.97056,30.55846,1.97056)
    forceH1 = CustomExternalForce('a*exp(-b*r)-c*exp(-d*r)+e*exp(-f*r);a=126763;b=88.7494
    ;c=-119802;d=89.3316;e=128810;f=88.7093;r=y')
    forceH2 = CustomExternalForce('a*exp(-b*r)-c*exp(-d*r)+e*exp(-f*r);a=126763;b=88.7494
    ;c=-119802;d=89.3316;e=128810;f=88.7093;r=ly-y;ly=29.5584')
    forceO1 = CustomExternalForce('a*exp(-b*r)-c*exp(-d*r)+e*exp(-f*r);a=-152785;b=94.
    7616;c=-35653400;d=100.7009;e=-438.194;f=15.1075;r=y')
    forceO2 = CustomExternalForce('a*exp(-b*r)-c*exp(-d*r)+e*exp(-f*r);a=-152785;b=94.
    7616;c=-35653400;d=100.7009;e=-438.194;f=15.1075;r=ly-y;ly=29.5584')
    system.addForce(forceH1)
    system.addForce(forceO1)
    system.addForce(forceH2)
    system.addForce(forceO2)

    for i in range(system.getNumParticles()):
        at2 = system.getParticleMass(i).value_in_unit(dalton)
        if at2 >= 1.007 and at2 <= 1.009:
            #print 'Es hidrogeno'
            forceH1.addParticle(i,[])
            forceH2.addParticle(i,[])
        elif at2 >= 15 and at2 <= 16:
            #print 'Es Oxygeno'
            forceO1.addParticle(i,[])
            forceO2.addParticle(i,[])
        else:
            nada = 0
            #print 'Problema con: ',i,system.getParticleMass(i)
elif Hidratacion == "AGP":
    boxVector=(1.97056,29.55846,1.97056)
else:
#seleccion para sistema sin potenciales externos
```

```
166     nada = 0
167 
168 #Seleccion de plataforma y numero de GPU's
169 platform = Platform.getPlatformByName('CUDA')
170 if computer == "local":
171     properties = {'CudaPrecision': 'double'}
172 elif computer == "yoltla" or computer == "abacus":
173     properties = {'CudaPrecision': 'double','CudaDeviceIndex': '0,1'}
174 else:
175     computer = ""
176 
177 #system.addForce(MonteCarloBarostat(presion*atmospheres, temperatura*kelvin,
        intervaloBarost))
178 integrator = LangevinIntegrator(temperatura*kelvin, 1/picosecond, timestep*picoseconds)
179 #integrator = VerletIntegrator(timestep*picoseconds)
180 
181 #Tolerancia en restriccion de enlace
182 if forceField == "tip4pfb.xml":
183     integrator.setConstraintTolerance(1e-8)
184 else:
185     var = "no se necesita"
186 
187 #Creacion del contexto de simulacion
188 simulation = app.Simulation(modeller.topology, system, integrator,platform)
189 simulation.context.setPositions(modeller.positions)
190 simulation.context.setVelocitiesToTemperature(temperatura*kelvin)
191 
192 
193 #Impresion de momento multipolar para iamoeba y amoeba
194 if forceField == "iamoeba.xml" or forceField == "amoeba2013.xml":
195     forces = [system.getForce(force_index) for force_index in range(system.getNumForces()
        )]
196     forces = [force for force in forces if isinstance(force, openmm.AmoebaMultipoleForce)
        ]
197     force = forces[0]
198     #fileInducedDipole = open('DipoloInducido.txt','w')
199     #fileTotDipole = open('DipolosTotales.txt','w')
200     fileSystMomMultipolar = open('MomentoMultipolarSistema.txt','w')
201     #filePermanentDipoles = open('DipolosPermanentes.txt','w')
202 if forceField == "tip4pfb.xml":
203 #Modificacion NonbondedForce
204     forcs = { system.getForce(index).__class__.__name__ : system.getForce(index) for
        index in range(system.getNumForces()) }
205     f = forcs['NonbondedForce']
206     f.setCutoffDistance(cutoff)
207     f.setSwitchingDistance(cutoff - switch_width)
208     f.setUseDispersionCorrection(False)
209     f.setUseSwitchingFunction(use_switch)
210 else:
211     var2 = "no implementado para otro potencial"
212 
213 #Reinicio de simulacion a partir del contexto guardado
214 #simulation.loadState('Input.xml')
215 
216 simulation.context.setPeriodicBoxVectors(Vec3(boxVector[0],0,0),Vec3(0,boxVector[1],0),
        Vec3(0,0,boxVector[2]))
217 #file1= open('posiciones.xyz','w')
218 #simulation.reporters.append(PosReporter('velocidades.dat', stepImp))
219 #simulation.reporters.append(ForceReporter('fuerzas.dat',stepImp))
220 simulation.reporters.append(PDBReporter('trayectoria.pdb', stepImp))
221 simulation.reporters.append(StateDataReporter('dinamica.dat', stepImp, step=True,
        potentialEnergy=True, kineticEnergy=True, totalEnergy=True, temperature=True,volume=
        True,density=True, speed=True,time=True,remainingTime=True, totalSteps=pasosTotales,
        separator='\t'))
222 
223 for i in range(pasosTotales):
```

```
    simulation.saveState('output.xml')
    simulation.step(stepImp)
    #state1 = simulation.context.getState(getPositions=True,enforcePeriodicBox = False,
    groups={0})
    #file1.write("%d\n\n"% (system.getNumParticles()))
    #pos1=state1.getPositions().value_in_unit(nanometers)
    #i=0
    #for pos in pos1:
    #    at2 = system.getParticleMass(i).value_in_unit(dalton)
    #    if at2 >= 1.007 and at2 <= 1.009:
    #        symbol = "H"
    #    elif at2 >= 15 and at2 <= 16:
    #        symbol = "O"
    #    elif at2 == 0.0:
    #        symbol = "M"
    #    else:
    #        nada = 0
    #    i=i+1
    #    file1.write("{:} \t{:+20.3f}\t{:+20.3f}\t{:+20.3f}\n".format(symbol,pos[0]*10,
    pos[1]*10,pos[2]*10))

    if forceField == "iamoeba.xml" or forceField == "amoeba2013.xml":
       #Dipolos inducidos
    #    inducedDipoleMoments = force.getInducedDipoles(simulation.context)
    #    imprimeMultipolos(fileInducedDipole,inducedDipoleMoments,simulation.currentStep,
    timestep)

       #Dipolos totales
    #    totaldipoles = force.getTotalDipoles(simulation.context)
    #    imprimeMultipolos(fileTotDipole,totaldipoles,simulation.currentStep,timestep)

       #Momento multipolar del sistema
        systemMultipolarMoment = force.getSystemMultipoleMoments(simulation.context)
        imprimeMomentoMultipolar(fileSystMomMultipolar,systemMultipolarMoment,simulation.
    currentStep,timestep)

       #Dipolos Permanentes
    #    permanentDipole = force.getLabFramePermanentDipoles(simulation.context)
    #    imprimeMultipolos(filePermanentDipoles,permanentDipole,simulation.currentStep,
    timestep)
    else:
       var2 = ""
```

7.2 Subrutina del cálculo de los perfiles de densidad

```
#include "librerias.h"

//la unidad de medida de la caja es en angstroms
#define SideX 19.7056
#define SideY 295.5846
#define SideZ 19.7056

int main(int argc,char** argv){
  //Archivo xyz
  strcpy(Archivo,argv[1]);

  //Coordenada de analisis
  strcpy(Coordinate,argv[2]);

  //Divisiones a considerar
  Bins=atof(argv[3]);

  //Atomo a considerar
  strcpy(TypeAtomsConsider,argv[4]);
```

```
20
21    //Atomo a considerar numero
22    AssignProfile();
23
24    //Coordenada de analisis
25    SelectCoordinate();
26
27    //Lineas de archivo
28    LinesFile = LongitudArchivo(Archivo);
29
30    //Asigna numero de configuraciones
31    fileParticlesConfiguration(Archivo);
32
33    //Alertas tamano de arreglos
34    if(Alerts()!=2){return 0;}
35
36    //Analiza trayectoria
37          file_xyz=fopen(Archivo,"r");
38
39              for(Replicas=0;Replicas<Configuration;Replicas++){
40      //Asigna valores a arreglos
41                  ReadFileValues();
42
43                  //Calcula densidad
44                  AssignDensity();
45                  }
46          //Imprime perfil de densidad
47          PrintProfile();
48
49          fclose(file_xyz);
50
51          return 0;
52
53  }
```

7.3 Subrutina del cálculo de la orientación

```
1  #include "librerias.h"
2
3  //la unidad de medida de la caja es en angstroms
4  #define SideX 19.7056
5  #define SideY 295.5846
6  #define SideZ 19.7056
7
8  int main(int argc,char** argv){
9    //Archivo xyz      //argv[contador] empieza en 1 porque cero es el ejecutable
10   strcpy(Archivo,argv[1]);
11
12   //Coordenada de analisis
13   strcpy(Coordinate,argv[2]);
14
15   //Divisiones del eje de analisis
16   Bins=atof(argv[3]);
17
18   //Atomo a considerar, siempre O
19   strcpy(TypeAtomsConsider,argv[4]);
20
21   //Atomo a considerar, numero
22   AssignProfile();
23
24   //seleccion de la coordenada de analisis
25   SelectCoordinate();
26
27   //Calcula las lineas del archivo
```

```
  LinesFile = LongitudArchivo(Archivo);

  //Calcula el numero de configuraciones
  fileParticlesConfiguration(Archivo);

  //Verifica que el tamano de los arreglos
  if(Alerts()!=2){return 0;}

  //Analiza la trayectoria
        file_xyz=fopen(Archivo,"r");

        for(Replicas=0;Replicas<Configuration;Replicas++){
      //Asigna coordenadas a los arreglos
            ReadFileValues();
             //Calcula angulos        //asigna valores de cada replica
            AssignAngles();
            }

        //Imprime Angulos
        PrintProfileAngles();

        fclose(file_xyz);

        return 0;

}
```

7.4 Subrutina del cálculo del coeficiente de auto-difusión

```
#include "librerias.h"

//la unidad de medida de la caja es en angstroms
#define SideX 19.7056
#define SideY 295.5846
#define SideZ 19.7056

int main(int argc,char** argv){
  //argv[contador] empieza en 1 porque cero es el ejecutable
  //Asigna valores leidos

  //Archivo a analizar
  strcpy(Archivo,argv[1]);

  //tiempo de simulacion en ns
  timeSimul = atof(argv[2]);

  //pasos de guardado en la trayectoria
  Steps = atof(argv[3]);

  //Delta t recibido en ps
  Deltat = atof(argv[4]);

  //Delta t en ns
  Deltat = Deltat/1000;

  //Coordenada de analisis
  strcpy(Coordinate,argv[5]);

  //seleccion de coordenada
  SelectCoordinate();

  //lee lineas totales, calcula el numero de configuraciones
  LinesFile=LongitudArchivo(Archivo);

```

```
  //Asigna el numero de configuraciones
  fileParticlesConfiguration(Archivo);

  //TypeAtoms es igual a numero de atomos por molecula
  NumberOfMolec = Particles/TypeAtoms;

  //Alertas si pasa el tamano de arreglos
  if(Alerts()!=2){return 0;}

  //Analiza el archivo
  file_xyz=fopen(Archivo,"r");
  for(Replicas=0;Replicas<Configuration;Replicas++){
    //Asigna valores de coordenadas a los arreglos
      ReadFileValues();

    //Asigna la configuracion inicial
      if(Replicas == 0){
                          //asigna valores de cada replica
                AssignRinicial();
        }
      else{
           AssignRtemp();
          }
    //Calcula la diferencia entre los vectores
    ComputeDifference();

    //Tiempo en el que se calculo la replica
    // 1(500)(0.002) = 1e-3 [ns]
    tiempoVar = (Replicas + 1)*Steps*Deltat;

    //Imprime el MSD por componente y total en cada replica
    PrintProfileCD();
    }

  // Calcula el MSD final por componente y total
  //MSD promedio
  MSD = MSD/Configuration;
  //MSD = sqrtl(MSD);

  //MSD promedio por componente
  for(int j=0;j<3;j++){
      MSD_comp[j] = MSD_comp[j]/Configuration;
      //MSD_comp[j] = sqrtl(MSD_comp[j]);
     }

  //Datos MSD total de la trayectoria
  cout << "MSD: " << AngCua2CmCua(MSD) << "\t ln(D) [cm^2/s]: " << log(AngCua2CmCua(MSD)
    /(6*Ns2s(timeSimul)));
  cout << "\t D [cm^2/s]: " << AngCua2CmCua(MSD)/(6*Ns2s(timeSimul))<<endl;

  //Datos MSD por componente de la trayectoria
  for(int j=0;j<3;j++){
    cout << "MSD componente: " << j+1 << "\t" << MSD_comp[j] << "\t ln(D) [cm^2/s]: " <<
    log(AngCua2CmCua(MSD_comp[j])/(6*Ns2s(timeSimul)));
    cout << "\t D [cm^2/s]: " << AngCua2CmCua(MSD_comp[j])/(6*Ns2s(timeSimul)) << endl;
    }

  fclose(file_xyz);

        return 0;

}
```

7.5 Subrutina del cálculo de la constante dieléctrica para modelos rígidos

```
#include "librerias.h"

//la unidad de medida de la caja es en angstroms
#define SideX 19.7056
#define SideY 295.5846
#define SideZ 19.7056

int main(int argc,char** argv){

  //Archivo xyz
  strcpy(Archivo,argv[1]);

  //Modelo de agua
  strcpy(WaterModel,argv[2]);

        //Replicas sin considerar
  NumberOfReplica=atoi(argv[3]);

  //Calcula lineas de archivo
  LinesFile=LongitudArchivo(Archivo);

  //Calcula el numero de configuraciones
  fileParticlesConfiguration(Archivo);

  //Detiene el programa si pasa limites de arreglos
  if(Alerts()!=2){return 0;}

  //Coordenada de analisis
  CoordinateSelection();

  //volumen por celda y volumen de la celda
  ComputeDelta(valor);

  //Seleccion del modelo y cargas
  ModelSelection();

  //Creacion archivo de salida
  strcat(Output,"Dielectrica.dat");
  OutPut.open(Output,std::ofstream::out | std::ofstream::trunc);

  //Analiza los n puntos de la trayectoria
  file_xyz=fopen(Archivo,"r");

  for(Replicas=0;Replicas<Configuration;Replicas++){
    //Asigna coordenadas a los arreglos
    ReadFileValues();

    //Replicas sin considerar
    if(Replicas >= NumberOfReplica){
        ComputeDipoleMomentSystem();
        //Imprime constante dielectrica en funcion del tiempo
        PrintProfileDielectricConstant();
    }
  }

  //Imprime perfil de constante dielelectrica
  //PrintProfileDielectricConstant();

  fclose(file_xyz);
        OutPut.close();

   return 0;

```

```
}
```

7.6 Subrutina del cálculo de la constante dieléctrica para modelos flexibles

```
#include "librerias.h"

//la unidad de medida de la caja es en angstroms
#define SideX 19.7056
#define SideY 295.5846
#define SideZ 19.7056

int main(int argc,char** argv){
  int lineaInteres = 0;
  char ArchivoSalida[100]="";
  //Archivo xyz
  strcpy(Archivo,argv[1]);

  //Linea donde se encuentran las coordenadas del momento dipolar
  lineaInteres = atoi(argv[2]);

        //Replicas sin considerar
  NumberOfReplica=atoi(argv[3]);

  //numero de datos por bloque
  //bloque = atoi(argv[4]);

  //Calcula lineas de archivo
  LinesFile=LongitudArchivo(Archivo);

  //7 es el numero de lineas del archivo
  Configuration = LinesFile/7;

  file_xyz = fopen(Archivo,"r");

  //Crea archivo de salida
  strcat(Output,"Dielectrica.dat");
  OutPut.open(Output,std::ofstream::out | std::ofstream::trunc);

  //analiza archivo
  for(Replicas=0;Replicas<Configuration;Replicas++){

    //Obtiene Momento dipolar
    MomentoDipolarModeloFlexible(lineaInteres);

    if(Replicas >= NumberOfReplica){
         //Imprime constante dielectrica en funcion del tiempo
         PrintProfileDielectricConstant();
                }
         }

  //Cierra el archivo de lectura y el de escritura
       fclose(file_xyz);
       OutPut.close();

   return 0;

}
```

7.7 Subrutina del cálculo de la Función de distribución radial

```
/************************************************
```

```
 * calculate radial distribution function g(r)
 * of ensemble configurations in "configuration.xyz"
 * output results in "gr.dat"
 * Kai Zhang, Duke University, 2011
 * Calculate rdf only of 2 atoms and you can give its symbol
 * Anthoni Alcaraz Torres CIQ, UAEM 2015
 ****************************************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
/**
 *Datos que se modifican
 */
#define MAXConfig 150000
#define MAXEqui 30000
#define MAXNumberOfParticles 50000
#define MAXNumberOfBins 10000
#define SideY 295.5846
#define SideZ 19.7056
#define SideX 19.7056            //la unidad de medida de la caja es en angstroms

#define SQR(x) ((x)*(x))
#define CUB(x) ((x)*(x)*(x))
//Apuntador a la ruta donde se encuentra el archivo con la trayectoria de la DM con las coordenas en xyz
char dirArchivoDinamica[100];
char *dirArchivogrAB="gAB2.dat";
int getLength(char*);
int main(int argc,char** argv){
  int NumberOfConfig,NumberOfEqui,NumberOfParticles;
  int SampleNumber;
  int snapshot,line;
  double rho,vol;
  double dr;
  double g[MAXNumberOfBins], gAB[MAXNumberOfBins];
  int NumberOfBins, NA, NB;
  int i,j, k,l,NumLineas;
  double dx,dy,dz;
  double rij;
  char AtomID[MAXNumberOfParticles][4], AtomA[4], AtomB[4], lineaUsada[100];
  FILE *fpxyz;
  FILE *fpoutput;
  FILE *fpoutput2;
  double x[MAXNumberOfParticles],y[MAXNumberOfParticles],z[MAXNumberOfParticles];
  vol = SideX*SideY*SideZ;
  //printf("*************************** 3D g(r) ************************************\n");
  strcpy(dirArchivoDinamica,argv[1]);
  //printf("Number of Equilibrations ?\n");
  NumberOfEqui = atoi(argv[5]);
  //printf("Number of bins of the histogram ?\n");
  dr = atof(argv[4]);
  //printf("Number of atoms of A in each configuration\n");
  NA =atoi(argv[6]);
  //printf("Number of atoms of B in each configuration\n");
  NB = atoi(argv[7]);
  //printf("Symbol of atom A\n");
  strcpy(AtomA,argv[2]);
  //printf("Symbol of atom B\n");
  strcpy(AtomB,argv[3]);
  NumberOfBins = (int)(15.0/dr);
  int contadorAtomA[NA+1], contadorAtomB[NB+1], asignaA=1, asignaB=1;
  SampleNumber = 0;
  for(i=0;i<NumberOfBins;i++){
   g[i] = 0;
```

```
    gAB[i] = 0;
  }
  SampleNumber = 0;
NumLineas=getLength(dirArchivoDinamica); //printf("Lineas %d\n",NumLineas);
fpxyz = fopen(dirArchivoDinamica,"r");
fgets(lineaUsada,100,fpxyz);sscanf(lineaUsada,"%d",&NumberOfParticles);
fclose(fpxyz);
rho = NumberOfParticles/vol;
NumberOfConfig=NumLineas/(NumberOfParticles+2); //printf("Configuraciones%d \n",
    NumberOfConfig);
  fpxyz = fopen(dirArchivoDinamica,"r");
  for(snapshot=0;snapshot<NumberOfConfig;snapshot++){
  fgets(lineaUsada,100,fpxyz);fgets(lineaUsada,100,fpxyz);
       for(line=0;line<NumberOfParticles;line++){
  fgets(lineaUsada,100,fpxyz);
       sscanf(lineaUsada,"%s %lf %lf %lf",AtomID[line],&x[line],&y[line],&z[line]);
  //printf("%d \t%s %10.6lf %10.6lf %10.6lf\n",line,AtomID[line],x[line],y[line],z[line])
    ;
  }
//Busca en el arreglo donde se encuentra el atomo A para tener coordenadas
if(snapshot == 0){
printf("Entro\n");
for(i=0;i<NumberOfParticles;i++){
 if(strstr(AtomID[i],AtomA)!=0){
   if(asignaA<=NA){
  contadorAtomA[asignaA]=i;
  //printf("A %s \t %d \t %d\n",AtomID[i],i, contadorAtomA[asignaA]);
  asignaA++;
    }
  }
 if(strstr(AtomID[i],AtomB)!=0){
    if(asignaB<=NB){
      contadorAtomB[asignaB]=i;
     //printf("B %s \t %d \t %d\n",AtomID[i],i, contadorAtomB[asignaB]);
      asignaB++;
    }
  }
}
printf("salio\n");
}

   if((snapshot+1) > NumberOfEqui){
    SampleNumber++;
      for(k=1;k<NA;k++){
        for(l=k+1;l<=NB;l++){
       i=contadorAtomA[k];
       j=contadorAtomB[l];
       dx = x[i] - x[j];
       dx = dx - SideX*round(dx/SideX);

       dy = y[i] - y[j];
       dy = dy - SideY*round(dy/SideY);

       dz = z[i] - z[j];
       dz = dz - SideZ*round(dz/SideZ);

       rij = sqrt(SQR(dx)+SQR(dy)+SQR(dz));

       gAB[(int)(rij/dr)] += 2.0;
     }
    }
  }
}
  fclose(fpxyz);

fpoutput2 = fopen(dirArchivogrAB,"w");
```

```
  for(i=0;i<NumberOfBins;i++){
   gAB[i] /= SampleNumber;
   gAB[i] /= 4.0*M_PI/3.0*(CUB(i+1)-CUB(i))*CUB(dr);
   fprintf(fpoutput2,"%f\t %f\n",(i+0.5)*dr,gAB[i]*vol/(NA*NB));
  }
  fclose(fpoutput2);
 return 0;
}
int getLength(char* dir){
        FILE *f;
        //obtiene la cantidad de lineas del archivo
        int l=0;
        char linea[100];
        f=fopen(dir, "r");
        if(f!=NULL){
                while(!feof(f)){
                        fgets(linea,100,f);
                        l++;
                }
                fclose(f);
        }
        return l-1;
}
```

7.8 Subrutina para convertir de formato PDB a XYZ

```
/*
Esta rutina convierte un archivo.pdb a un archivo.xyz de N moleculas de un solo tipo, por ejemplo agua.
+++Si el PDB tiene mas de un tipo de molecula no lo hara bien, porque cuando llama a la subrutina ReadingChange()
esta depende de dos parametros (AtomsByMol y OneLine). Estos parametros indican hasta que linea del archivo PDB algunos datos se juntaran. Por ejemplo:
+HETATM    1  H1  HOH A   1       5.444  37.814   6.499  1.00  0.00           H
En la linea anterior, los 12 datos tienen un espacio que los separa, pero cuando el numero de la molecula ocupa mas de 3 espacios; dos datos
se juntan y la linea se parece a esto:
+HETATM    1  H1  HOH A1000       5.444  37.814   6.499  1.00  0.00           H
La linea anterior indica que 1000 moleculas han sido asignadas, pero al leerla se analiza si es un entero( numero de la molecula) o
si es una cadena (A). Cuando se juntai, se convierte en una cadena y, la forma en que se lee ya no es correcta. Justo para esa parte de la linea
se pasa de leer ("%*s %*d") a ("%*s").
##################### S O L U C I O N E S ##################################################3
SOLUCION 1: Si hay >= 1000 moleculas con mas atomos, entonces se pueden dejar las moleculas con menos atomos al final y debe funcionar.
Por ejemplo, 1000 H2O primero y luego los N iones.
SOLUCION 2: En la linea 35, cambiar <<ReadingChange(Archivo)>> por el numero del ultimo atomo que pertenece a la molecula 999. Por ejemplo, si el atomo 3996 es el ultimo
atomo de la molecula 999, entonces el cambio seria asi:
OneLine = ReadingChange(Archivo);  por OneLine = 3996;
*/
#include "librerias.h"
int main(int argc,char** argv){
  char salidaPDB[100]="";
  //Archivo.pdb a Analizar
  strcpy(Archivo,argv[1]);

  //Lineas de archivo
  LinesFile=LongitudArchivo(Archivo);

  //Configuraciones
  fileParticlesConfigurationPDB(Archivo);

```

```
  /*Los datos se juntan en el PDB, solo debe tener 3 lineas basura al inicio:
        REMARK CRYST1 y MODEL, si no eliminar las sobrantes o poner lineas en blanco.
       Por ejemplo si esta la linea Title de más, no funciona correctamente
      */
  OneLine = ReadingChange(Archivo);

  //Archivo de Salida
  strcat(salidaPDB,Archivo);
  strcat(salidaPDB,"2.xyz"); cout<<"Guardado en: "<<salidaPDB<<endl;

  //input y output
  freopen(salidaPDB,"w",stdout);
  file_pdb=fopen(Archivo,"r");

  //Dos lineas basura, REMARK y CRYT1
    fgets(LineRead,100,file_pdb); fgets(LineRead,100,file_pdb);

    //analiza archivo pdb
    for(Replicas=0;Replicas<Configuration;Replicas++){

    //Las dos primeras lineas de un archivo xyz
        cout<<Particles<<endl<<endl;

        //Asinga valores de cada replica
        ReadFileValuesPDB();

        }
    //Cierra archivo de lectura y de escritura
    fclose(file_pdb);
  fclose(stdout);
  return 0;

}
```

7.9 Librerías de subrutinas de análisis de propiedades

```
/*************************************************
 * Anthoni Alcaraz Torres CIQ, UAEM 2018
 * Maestría en Ciencias
 * anthoni.alcaraztor@uaem.edu.mx
 * toni_surfin@hotmail.com
 *=== Desarrollo de herramientas metodológicas
      de simulación numérica para el estudio
            de interfases acuosas ====
 *************************************************/

#include <iostream>
#include <string>
#include <iomanip>
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <fstream>
using namespace std;

//Datos a Modificar
//la unidad de medida de la caja es en angstroms
#define SideX 19.7056
#define SideY 295.5846
#define SideZ 19.7056
#define MaxConfigur 150000
#define MaxParticles 20000
#define MaxOfBins 200
```

```
#define MasH 1.0079
#define MasO 15.999
#define MasAgua 18.0148
#define TypeAtoms 3
#define AvogNumber 6.020140857e23
#define DeltaDegree 5
#define TotalAngles 360
#define HalfAngles 180
#define ElectronCharge 1.602176565E-19 //C
#define KBJ 1.380648813E-23 //J/K
#define KBeVK 8.617332478E-5  //eV/K
#define eps0 8.8541878176E-12 //C2/Nm2 || F/m

//funciones

void fileParticlesConfiguration(char*);
void fileParticlesConfigurationPDB(char*);
void ReadFileValues();
void ReadFileValuesVelocities();
void SelectCoordinate();
void AssignDensity();
void AssignAngles();
void DensityX();
void DensityY();
void DensityZ();
void AnglesX();
void AnglesY();
void AnglesZ();
void ComputeDelta(int);
void CalcAngles();
void CalcDensity();
void RestartVal();
void PrintProfile();
void PrintProfileAngles();
void PrintDistribution();
void FormatPrint();
void PerfilByAtom(int,int);
void AssignProfile();
void ProfileSelected(int);
void AssignVector(double, double, double, double, double, double, double, double, double,
    double);
void AssignCount();
void AssignRinicial();
void AssignRtemp();
void ComputeDifference();
void ComputeRCMass(int);
void PrintProfileCD();
void PrintProfileDielectricConstant();
void ComputeDipoleMomentSystem();
void ComputeDipoleMoment(int, double []);
void DefineCharges(int);
void ValuesToCero();
void TranslatefullBox();
void CoordinateSelection();
void TranslateBox();
void PrintTip4pToTip3p();
void distance_pbc(double [3],int);
void ProductVectorByScalar(double [3], double);
void ModelSelection();
void DipoleMomentBySliceFunction(double [3],int,double [3]);
void MomentoDipolarModeloFlexible(int);
void ImprimeBines();
void Distances();
void PrintDistance(int,double [2]);
void DistanceToWall(double [2]);
void DivideMolecules();
```

```
94  void ReadFileDataGraph();
95  void HeaderGnuplot(int);
96  void FooterGnuplot(int,int);
97  void GridWall(int);
98  void MapGrid();
99  void FileGnuplotGrid();
100 void ComputeDeltaGrid();
101 void PrintGrid();
102 void PrintGridGnuplot();
103 double DotProduct(double [3],double [3]);
104 double CosineAngle(double [3],double [3]);
105 double AssignBinrCM();
106 double MagnitudeVector(double [3]);
107 int Alerts();
108 int ReadDataAndAssignBlock();
109 int LongitudArchivo(char*);
110 int ReadingChange(char*);
111
112 //files to write something
113
114 ofstream OutPut;
115 ofstream OutPut2;
116 ofstream OutPutBins;
117 ofstream Densidades;
118 ofstream Densidad;
119 ofstream FileSalida;
120
121 #define SQR(x) ((x)*(x))
122 #define CUB(x) ((x)*(x)*(x))
123 #define Deg2Rad(x) (((x)*(M_PI))/180)
124 #define Rad2Deg(x) (((x)*(180))/M_PI)
125 #define AngCua2CmCua(x)  (x*(1.0E-8)*(1.0E-8))
126 #define Ang2cm(x) (x*(1.0E-8))
127 #define Ang2m(x) (x*(1.0E-10))
128 #define Ang32m3(x) (x*((1.0E-10)*(1.0E-10)*(1.0E-10)))
129 #define Ns2s(x) (x*(1.0E-9))
130 #define ps2s(x) (x*(1.0E-12))
131 #define Debye2CM(x) (x*(3.34E-30))
132 #define Debye22Cm2(x) (x*(3.34E-30)*(3.34E-30))
133 #define Cm2Debye(x) (x/(3.34E-30))
134
135 //contadores
136
137 int AtomsByMol=0,OneLine=0,bloque=0,Bini,Binj;
138 int Configuration, LinesFile, Particles, Replicas=0,i=0;
139 int NumAtoms[TypeAtoms], RefBin=0,RefBinAng,atoms[MaxParticles];
140 int valor,Bins=0,NumAtomConsider,distribution[MaxParticles];
141 int contadorReplicas=0,NumberOfReplica=0,ContadorRep[MaxOfBins]={0};
142
143 //Dobles
144
145 //El respDen tiene typatoms +1 porque son los tipos de atomos mas uno donde uno almacena
      la masa total de cada delta
146 double x[MaxParticles], y[MaxParticles], z[MaxParticles], Density[MaxOfBins], Delta,
      DeltaVolumen;
147 double MasTypeAtoms[TypeAtoms+1]={MasO,MasH,MasH,0},AngPMHs[3],AngWall[3],AngH1[3],AngH2
      [3];
148 double vel[MaxParticles],cinetica=0,Cos[TypeAtoms], DeltaAngle=Deg2Rad(DeltaDegree),
      tiempoVar;
149 double acumulador=0, AcumulaConteo[MaxOfBins][TypeAtoms+1];
150 double AcumulaAnglesHist[MaxOfBins][TypeAtoms+1][HalfAngles/DeltaDegree]={0.0};
151 double repH1[MaxOfBins]={0.0},repH2[MaxOfBins]={0.0};
152 double AcumulaAngles[MaxOfBins][TypeAtoms+1]={0.00},RespDen[MaxOfBins][TypeAtoms+1], rep[
      MaxOfBins]={0.00};
153 double r_temp[MaxParticles][3],r_inic[MaxParticles][3],r_dif[MaxParticles][3];
```

```
154 double MSD=0.0,MSDTemp=0.0, timeSimul,NumberOfMolec, rCM[3]={0.0},MSD_comp[3]={0.0},
        MSDTemp_comp[3]={0.0};
155 double MSD_Average = 0.0, MSD_comp_Average[3]={0.0};
156 double Steps, Deltat,MSD_comp_bin[3][MaxOfBins]={0.0},MSD_bin[MaxOfBins]={0.0};
157 double NumberOfMolec_bin[MaxOfBins]={0.0},MSDTemp_bin[MaxOfBins]={0.0},MSDTemp_comp_bin
        [3][MaxOfBins]={0.0};
158 double DipoleMomentByCell[2] = {0.0,0.0}, DipoleMomentBySlice[MaxOfBins][3] = {0.0};
159 double DMomentTotalBySlice[MaxOfBins][2] = {0.0};
160 double MMp=0,SumXZY[3]={0.0,0.0,0.0},Sum2XZY[3]={0.0,0.0,0.0};
161 double charge[3]={0.0,0.0,0.0},MomDipolar[3]={0.0};
162 double Mesh[MaxOfBins][MaxOfBins]={0.0},limite;
163 double ContadorPunto[2][MaxOfBins][MaxOfBins]={0.0},ContadorPuntoTotal[2]={0.0};
164
165 //Caracteres
166
167 FILE* file_xyz;
168 FILE* file_pdb;
169 char LineRead[100], AtomID[MaxParticles][4],WaterModel[12]="";
170 char AtomSearch[TypeAtoms+1][5]={{"O"},{"H"},{"N"},{"all"}};
171 char Archivo[100], Coordinate[4], TypeAtomsConsider[5];
172 char Output1[]="PerfDensidad";
173 char Output[100];
174 char Output2[]="PerAngDipol";
175 char Output3[]="PerDielectricConstant";
176
177 //ESTRUCTURAS
178
179 typedef struct charges{
180 double ChargeO, ChargeH, ChargeM;
181 }modelo;
182
183 modelo iamoeba, amoeba, tip4pfb;
184
185 void PrintGridGnuplot(){
186     //Graficar dos distribuciones en una misma grafica
187     OutPut.open("input.gpl",std::ofstream::out | std::ofstream::trunc);
188     OutPut << "set encoding iso_8859_1" << endl;
189     OutPut << "set terminal pdf enhanced" << endl;
190     OutPut << "set output \"MallaSup.pdf\" " << endl;
191     OutPut << "set lmargin at screen 0.15" << endl;
192     OutPut << "set rmargin at screen 0.45" << endl;
193     OutPut << "set tmargin at screen 0.95" << endl;
194     OutPut << "set multiplot layout 1,2 rowsfirst" << endl;
195     OutPut << "set size square" << endl;
196     OutPut << "set pm3d map" << endl;
197     OutPut << "set cbrange[0.00055:0.0008]" << endl;
198     OutPut << "set palette rgb 33,13,10" << endl;
199     OutPut << "set xlabel \"Eje X [{\\305}] \" " << endl;
200     OutPut << "set ylabel \"Eje Z [{\\305}] \" " << endl;
201     OutPut << "unset colorbox" << endl;
202     OutPut << "splot \"MallaSuperficie1.dat\" notitle" << endl;
203     OutPut << "set lmargin at screen 0.55" << endl;
204     OutPut << "set rmargin at screen 0.85" << endl;
205     OutPut << "set tmargin at screen 0.95" << endl;
206     OutPut << "set colorbox" << endl;
207     OutPut << "splot \"MallaSuperficie2.dat\" notitle" << endl;
208     OutPut.close();
209   }
210
211 void PrintGrid(){
212   //Imprime valores del mapa para las dos superficies
213   OutPut.open("MallaSuperficie1.dat",std::ofstream::out | std::ofstream::trunc);
214   FileSalida.open("MallaSuperficie2.dat",std::ofstream::out | std::ofstream::trunc);
215   FormatPrint();
216   for(int i=0;i<Bini;i++){
217     for(int j=0;j<Binj;j++){
```

```
218       OutPut << Delta*(i+.5) << "\t" << Delta*(j+0.5) << "\t" << ContadorPunto[0][i][j]/
      ContadorPuntoTotal[0] << endl;
219       FileSalida << Delta*(i+.5) << "\t" << Delta*(j+0.5) << "\t" << ContadorPunto[1][i][
      j]/ContadorPuntoTotal[1] << endl;
220       }
221     OutPut << endl;
222     FileSalida << endl;
223     }
224   OutPut.close();
225   FileSalida.close();
226   }
227
228 void FileGnuplotGrid(){
229   //imprime archivos para graficar
230   PrintGrid();
231   //imprime input para gnuplot
232   PrintGridGnuplot();
233   }
234
235 void ComputeDeltaGrid(){
236   switch(valor){
237     //Para X
238     case 1:
239           if(SideY <= SideZ) {
240           Delta = SideY/Bins;
241           Bini = Bins;
242           Binj = int(SideZ/Delta);
243           }
244           else{
245           Delta = SideZ/Bins;
246           Bini = Bins;
247           Binj = int(SideY/Delta);
248           }
249           break;
250     //Para Y
251     case 2:
252           if(SideX <= SideZ) {
253           Delta = SideX/Bins;
254           Bini = Bins;
255           Binj = int(SideZ/Delta);
256           }
257           else{
258           Delta = SideZ/Bins;
259           Bini = Bins;
260           Binj = int(SideX/Delta);
261           }
262           break;
263     //Para Z
264     case 3:
265           if(SideX <= SideY) {
266           Delta=SideX/Bins;
267           Bini = Bins;
268           Binj = int(SideY/Delta);
269           }
270           else{
271           Delta=SideY/Bins;
272           Bini = Bins;
273           Binj = int(SideX/Delta);
274           }
275           break;
276     default:
277           cout << "No se asigno la cara de la celda" << endl;
278           break;
279     }
280   }
281
```

```
282 void GridWall(int i){
283   //Compara si el atomo esta en el intervalo de analisis
284   switch(valor){
285     //Para X
286     case 1:
287          if(x[i] <= limite){
288          //contador de atomos en la region
289          ContadorPuntoTotal[0]++;
290          //Contador de atomo por cuadricula de la malla
291          ContadorPunto[0][int(y[i]/Delta)][int(z[i]/Delta)]++;
292          }
293        if(x[i] >= SideX-limite){
294          //contador de atomos en la region
295          ContadorPuntoTotal[1]++;
296          //Contador de atomo por cuadricula de la malla
297          ContadorPunto[1][int(y[i]/Delta)][int(z[i]/Delta)]++;
298          }
299          break;
300     //Para Y
301     case 2:
302           if(y[i] <= limite){
303          //contador de atomos en la region
304          ContadorPuntoTotal[0]++;
305          //Contador de atomo por cuadricula de la malla
306          ContadorPunto[0][int(x[i]/Delta)][int(z[i]/Delta)]++;
307          }
308        if(y[i] >= SideY-limite){
309          //contador de atomos en la region
310          ContadorPuntoTotal[1]++;
311          //Contador de atomo por cuadricula de la malla
312          ContadorPunto[1][int(x[i]/Delta)][int(z[i]/Delta)]++;
313          }
314          break;
315     //Para Z
316     case 3:
317           if(z[i] <= limite){
318          //contador de atomos en la region
319          ContadorPuntoTotal[0]++;
320          //Contador de atomo por cuadricula de la malla
321          ContadorPunto[0][int(x[i]/Delta)][int(y[i]/Delta)]++;
322          }
323        if(z[i] >= SideZ-limite){
324          //contador de atomos en la region
325          ContadorPuntoTotal[1]++;
326          //Contador de atomo por cuadricula de la malla
327          ContadorPunto[1][int(x[i]/Delta)][int(y[i]/Delta)]++;
328          }
329          break;
330     default:
331          cout << "No se asigno átomo a la malla" << endl;
332          break;
333     }
334   }
335 
336 void MapGrid(){
337   for(int i=0;i<Particles;i++){
338     //Busca el atomo a localizar en la malla
339     if(strcmp(AtomID[i],TypeAtomsConsider)==0){
340       //analiza si el atomo esta en la region de analisis
341       GridWall(i);
342       }
343     }
344   }
345 
346 void HeaderGnuplot(int Bloque){
347   //opciones para gnuplot
```

```
348   OutPut << "set encoding iso_8859_1" << endl;
349   OutPut << "set terminal pdf enhanced" << endl;
350         OutPut << "set output \"Bloque" << Bloque << ".pdf\" " << endl;
351         OutPut << "set xlabel \"tiempo [ns]\" " << endl;
352         OutPut << "set yrange[0:5]" << endl;
353         OutPut << "set ylabel \"Distancia al plano [{\\305}]\" " << endl;
354         OutPut << "set key outside right top vertical Right noreverse enhanced" << endl;
355   }
356
357 void FooterGnuplot(int Bloque,int BT){
358   //Crea instrucción para graficar en gnuplot
359   char imprime[500] = "";
360   char Numero[100] = "";
361   char cadenaAuxiliar[100] = "";
362   char AtomoAuxiliar[100] = "Atomo";
363   char title[100] = "";
364   char Graf[100] = " u ($1/1000):2 w l t ";
365   strcat(imprime,"p ");
366   /*cambia el tamaño de bloque solo la cantidad de elementos
367   * no es suficiente para el número de elementos requeridos
368   * por bloque y que solo haga los elementos disponibles.*/
369   if(Bloque+1 == BT && LinesFile%bloque!=0){
370     bloque = LinesFile%bloque;
371     }
372   //bloque es el tamaño de los elementos del Bloque
373   for(int i=0; i<bloque; i++){
374     sprintf(Numero,"%d",atoms[Bloque*bloque+i]);
375     strcat(cadenaAuxiliar,"\"");
376     strcat(cadenaAuxiliar,AtomoAuxiliar);
377     strcat(cadenaAuxiliar,Numero);
378     strcat(title,"\"O-");
379     sprintf(Numero,"%04d",atoms[Bloque*bloque+i]);
380     strcat(title,Numero);
381     strcat(title,"\"");
382     strcat(cadenaAuxiliar,".dat\"");
383     strcat(imprime,cadenaAuxiliar);
384     strcat(imprime,Graf);
385     strcat(imprime,title);
386     if(i == (bloque-1)){
387       strcat(imprime," ");
388       }
389     else{
390       strcat(imprime,",");
391       }
392
393     //Borra el contenido de la cadena
394     strcpy(cadenaAuxiliar,"");
395     strcpy(title,"");
396     }
397   OutPut << imprime << endl;
398   OutPut << endl;
399
400   }
401
402 int ReadDataAndAssignBlock(){
403   int bloquesTotales=0;
404   //determina el numero de bloques que contiene n átomos
405   bloquesTotales = (int)(LinesFile/bloque);
406
407   if( LinesFile%bloque != 0){
408     bloquesTotales++;
409     }
410   file_xyz=fopen(Archivo,"r");
411   //lee datos de atomos a analizar
412   ReadFileDataGraph();
413   fclose(file_xyz);
```

```
414   return bloquesTotales;
415   }
416
417 void DivideMolecules(){
418   //clasifica oxigenos por secciones de acuerdo a su posicion
419   char OutputBin[100]="";
420         char OutputNumber[]="Conj";
421         int seccion=0,atomo=0;
422   double bin[4];
423   //bin[0] = SideY/4;
424   bin[0] = 5;
425   bin[1] = 2*SideY/4;
426   bin[2] = 3*SideY/4;
427   bin[3] = SideY;
428
429   for(int i=0;i<Particles;i++){
430     //busca oxigenos
431     if(strcmp(AtomID[i],"O")==0){
432       atomo++;
433       //A superficie positiva
434       if(y[i] < bin[1]){
435         if(y[i] < bin[0]){
436           seccion = 1;
437           }
438         else{
439           seccion = 2;
440           }
441         }
442        //A superficie negativa
443           else{
444       if(y[i] < bin[2]){
445         seccion = 3;
446         }
447       else{
448         seccion = 4;
449         }
450       }
451     sprintf(OutputNumber,"%d",seccion);
452                 strcat(OutputBin,"Seccion");
453                 strcat(OutputBin,OutputNumber);
454                 strcat(OutputBin,".dat");
455                 OutPut.open(OutputBin,std::ofstream::out | std::ofstream::app);
456                 OutPut << atomo << endl;
457                 OutPut.close();
458                 strcpy(OutputBin,"");
459         }
460       }
461
462   }
463
464 void PrintDistance(int archivo,double D[2]){
465   //archivo: tiene distancias de cada n átomo
466   char OutputBin[100]="";
467      char OutputNumber[]="Conj";
468
469      sprintf(OutputNumber,"%d",archivo);
470      strcat(OutputBin,"Atomo");
471      strcat(OutputBin,OutputNumber);
472      strcat(OutputBin,".dat");
473      //guarda en el mismo archivo (app)
474      OutPut.open(OutputBin,std::ofstream::out | std::ofstream::app);
475      OutPut << Replicas+1 << "\t" << D[0] << "\t" << D[1] << endl;
476      OutPut.close();
477      strcpy(OutputBin,"");
478   }
479
```

```
480 void Distances(){
481   //calcula distancia solo de oxigenos
482   double distancias[2]={0.0};
483   //contador de oxigenos
484   int Oxigeno = 0;
485   for(int i=0;i<Particles;i++){
486     if(strcmp(AtomID[i],"O")==0){
487       //asigna la coordenada y al elemento 1 del arreglo
488       distancias[0] = y[i];
489       DistanceToWall(distancias);
490       Oxigeno++;
491       PrintDistance(Oxigeno,distancias);
492       }
493     }
494   }
495 
496 void DistanceToWall(double d[2]){
497   //Calcula la distancia a cada superficies
498   //d[0]: superficie izquierda
499   //d[1]: superficie derecha
500   //Guarda la coordenada
501   double y = d[0];
502 
503   //distancia al plano y=0
504   d[0] = y;
505 
506   //distancia al plano y=SideY
507   d[1] = SideY - y;
508 
509   }
510 
511 void ImprimeBines(){
512      //Imprime las moleculas que estan en y menor a 10 solo para una configuracion
513      for(int i=0;i<Particles;i++){
514           if(strcmp(AtomID[i],"O")==0 && y[i]>290.584){
515                cout << AtomID[i-1] << "\t" << x[i-1] << "\t" << y[i-1] << "\t" << z[i-1]
     << endl;
516                cout << AtomID[i] << "\t" << x[i] << "\t" << y[i] << "\t" << z[i] << endl;
517                cout << AtomID[i+1] << "\t" << x[i+1] << "\t" << y[i+1] << "\t" << z[i+1]
     << endl;
518                }
519           }
520      }
521 
522 void DefineCharges(int forcefield){
523      //Define las cargas por modelo
524      switch(forcefield){
525           case 1:
526                  iamoeba.ChargeO = -0.594024*ElectronCharge;
527                  iamoeba.ChargeH =  0.297012*ElectronCharge;
528                  break;
529           case 2:
530                  amoeba.ChargeO = -0.51966*ElectronCharge;
531                  amoeba.ChargeH =  0.25983*ElectronCharge;
532                  break;
533           case 3:
534                  tip4pfb.ChargeO =  0.0;
535                  tip4pfb.ChargeH =  0.527*ElectronCharge;
536                  tip4pfb.ChargeM = -1.054*ElectronCharge;
537                  break;
538           default:
539                  cout << "No se asignaron las cargas, ";
540                  cout << "los modelos disponibles son: ";
541                  cout << "iamoeba, tip4pfb y amoeba" << endl;
542                  break;
543           }
```

```
544
545 }
546
547 void ModelSelection(){
548     int opcion=0;
549
550     if(strcmp(WaterModel,"iamoeba")==0){
551         opcion=1;
552         DefineCharges(opcion);
553         charge[0] = iamoeba.ChargeO;
554         charge[1] = iamoeba.ChargeH;
555         }
556
557     if(strcmp(WaterModel,"amoeba")==0){
558         opcion=2;
559         DefineCharges(opcion);
560         charge[0] = amoeba.ChargeO;
561         charge[1] = amoeba.ChargeH;
562         }
563
564     if(strcmp(WaterModel,"tip4pfb")==0){
565         opcion=3;
566         DefineCharges(opcion);
567         charge[0] = tip4pfb.ChargeO;
568         charge[1] = tip4pfb.ChargeH;
569         charge[2] = tip4pfb.ChargeM;
570         }
571     }
572
573 void ComputeDipoleMomentSystem(){
574
575     ComputeDipoleMoment(valor,charge);
576
577 }
578
579 void ComputeDipoleMoment(int value, double Char[]){
580     /*Direccion de Analisis, x=1,y=2, z=3
581      * Carga del modelo, Char[0] Oxigeno, Char[1] Hidrogeno y Char[2] M
582      * */
583
584         double DipoleMoment[3]={0.0,0.0,0.0};
585         double r[3] = {0.0,0.0,0.0};
586         double r2[3] = {0.0,0.0,0.0};
587         double Magnitude;
588
589         //Suma qr para todos los atomos
590                 for(int i=0;i<Particles;i++){
591                     // coordenadas convertidas a metros
592                     r[0] = Ang2m(x[i]);
593                     r[1] = Ang2m(y[i]);
594                     r[2] = Ang2m(z[i]);
595                     r2[0] = x[i];
596                     r2[1] = y[i];
597                     r2[2] = z[i];
598                     //Producto qr si es oxigeno
599                     if(strcmp(AtomID[i],"O")==0){
600                         ProductVectorByScalar(r,Char[0]);
601                         //DipoleMomentBySliceFunction(r2,value,r);
602                         for(int j=0;j<3;j++){
603                             DipoleMoment[j] = DipoleMoment[j] + r[j];
604                             }
605                         }
606                     //Producto qr si es hidrogeno
607                    if(strcmp(AtomID[i],"H1")==0 ||
608                       strcmp(AtomID[i],"H2")==0 ||
609                       strcmp(AtomID[i],"H")==0){
```

```
610                                 ProductVectorByScalar(r,Char[1]);
611                                 //DipoleMomentBySliceFunction(r2,value,r);
612                                 for(int j=0;j<3;j++){
613                                       DipoleMoment[j] = DipoleMoment[j] + r[j];
614                                       }
615                                 }
616                            //Producto qr si es Sitio Virtual
617                           if(strcmp(AtomID[i],"M")==0){
618                                 ProductVectorByScalar(r,Char[2]);
619                                 //DipoleMomentBySliceFunction(r2,value,r);
620                                 for(int j=0;j<3;j++){
621                                       DipoleMoment[j] = DipoleMoment[j] + r[j];
622                                       }
623                                 }
624                            }
625           //Convierte a Debye
626           Magnitude = Cm2Debye(MagnitudeVector(DipoleMoment));
627
628           /*
629           //Imprime Momento Dipolar Total en Debye
630           cout << Replicas+1 << "\t";
631           Magnitude = Cm2Debye(MagnitudeVector(DipoleMoment));
632           cout << Magnitude << endl;*/
633
634           /**********************/
635           //Imprime Momento Dipolar Total por Bloque en archivos diferentes
636           int numArchivo = 0;
637           char OutputBin[100]="";
638           char OutputNumber[]="Conj";
639           //Bins es el numero de division
640           numArchivo = (Replicas+1)%bloque;
641           //cout << numArchivo << endl;
642           if(numArchivo == 0) numArchivo = bloque;
643           sprintf(OutputNumber,"%d",numArchivo);
644           strcat(OutputBin,"MomDip");
645           strcat(OutputBin,OutputNumber);
646           strcat(OutputBin,".txt");
647           OutPut.open(OutputBin,std::ofstream::out | std::ofstream::app);
648           OutPut << Replicas+1 << "\t" << Magnitude << endl;
649           OutPut.close();
650           strcpy(OutputBin,"");
651
652
653           //Momento dipolar por rebanada
654           /*
655           for(int i=0; i<Bins; i++){
656                 Magnitude = Cm2Debye(MagnitudeVector(DipoleMomentBySlice[i]));
657                 cout << Magnitude << "\t";
658                 }*/
659
660         DipoleMomentByCell[0] = DipoleMomentByCell[0] + Magnitude;
661         DipoleMomentByCell[1] = DipoleMomentByCell[1] + Magnitude*Magnitude;
662
663         //Reinicia los valores para cada bin a cero
664         /*
665         for(int i=0;i<Bins;i++){
666                 for(int j=0;j<3;j++){
667                       DipoleMomentBySlice[i][j] = 0.0;
668                       }
669                 }*/
670
671      }
672
673 void MomentoDipolarModeloFlexible(int lin){
674      /*
675       * Esta funcion lee las componentes del dipolo
```

```
676       * de los modelos flexibles calculados en OpenMM
677       * */
678
679      char linea[100];
680      char OutputBin[100]="";
681      char OutputNumber[]="Conj";
682      double MomDipLocal=0.0;
683      int numArchivo=0;
684      lin--;
685      //Son 7 porque el bloque es de 7 lineas
686      for(int i=0; i<7 ;i++){
687           fgets(linea,100,file_xyz);
688           if(i == lin){
689                sscanf(linea,"%f %f %f",&MomDipolar[0],&MomDipolar[1],&MomDipolar[2]);
690                MomDipLocal = MagnitudeVector(MomDipolar);
691                //printf("%d \t %f \n",Replicas+1,MomDipLocal);
692                }
693           }
694      if(Replicas >= NumberOfReplica) {
695      //En Debye
696      DipoleMomentByCell[0] = DipoleMomentByCell[0] + MomDipLocal;
697      DipoleMomentByCell[1] = DipoleMomentByCell[1] + MomDipLocal*MomDipLocal;
698      }
699
700      strcat(OutputBin,"MomDip.dat");
701      OutPut2.open(OutputBin,std::ofstream::out | std::ofstream::app);
702      OutPut2 << Replicas+1 << "\t" << MomDipLocal << endl;
703      OutPut2.close();
704      strcpy(OutputBin,"");
705      /*char OutputBin[100]=""; //para asignar MD en bloques separados: 1, 19,37, etc
706      char OutputNumber[]="Conj";
707      //Bins es el numero de division
708      numArchivo = (Replicas+1)%bloque;
709      //cout << numArchivo << endl;
710      if(numArchivo == 0) numArchivo = bloque;
711      sprintf(OutputNumber,"%d",numArchivo);
712      strcat(OutputBin,"MomDip");
713      strcat(OutputBin,OutputNumber);
714      strcat(OutputBin,".txt");
715      OutPut.open(OutputBin,std::ofstream::out | std::ofstream::app);
716      OutPut << Replicas+1 << "\t" << MomDipLocal << endl;
717      OutPut.close();
718      strcpy(OutputBin,"");*/
719
720      }
721
722 void PrintProfileDielectricConstant(){
723      double Mmedia, M2media,factorCell,factorBin,e,reptemp=0;
724      //calculo de la fluctuación
725      factorCell = 1/(3*KBJ*298.15*eps0*Ang32m3(SideX*SideY*SideZ));
726
727      if(NumberOfReplica != 0){
728          //para el caso de los flexibles que necesitan equilibrar despues de tip4pfb
729          reptemp = Replicas +1-NumberOfReplica;
730          Mmedia = DipoleMomentByCell[0]/reptemp;
731          M2media = DipoleMomentByCell[1]/reptemp;
732          e =  1 + factorCell*(Debye22Cm2(M2media));
733          OutPut << reptemp << "\t" << e <<endl;
734      }
735      else{
736          Mmedia = DipoleMomentByCell[0]/(Replicas+1);
737          M2media = DipoleMomentByCell[1]/(Replicas+1);
738          e =  1 + factorCell*(Debye22Cm2(M2media));
739          OutPut << Replicas+1 << "\t" << e <<endl;
740      }
741
```

```
742       /*for(int j=0;j<Bins;j++){
743            //Formato para imprimir por columnas ordenadas y con numero de decimales fija
744            FormatPrint();
745            cout << Delta*(j+1)-Delta/2;
746            cout << "  \t" << DMomentTotalBySlice[j][0]/Configuration ;
747            cout << "  \t" << DMomentTotalBySlice[j][1]/Configuration << endl;
748            }*/
749       }
750
751  double DotProduct(double v1[3],double v2[3]){
752       double dotproduct=0;
753       for(int i=0; i<3 ;i++){
754            dotproduct = dotproduct + v1[i]*v2[i];
755            }
756       return dotproduct;
757       }
758
759  double MagnitudeVector(double Vector[3]){
760       double MagnitudeVec = 0.0;
761       for(int i=0;i<3;i++){
762            MagnitudeVec = MagnitudeVec + Vector[i]*Vector[i];
763            }
764       return sqrtl(MagnitudeVec);
765       }
766
767  void DipoleMomentBySliceFunction(double rn[3],int val, double rq[3]){
768       //val corresponde a x,y o z, {1,2,3} y val-1 corresponde a la coordenada
769       //en el arreglo
770       int bin=0;
771
772       //Asigna el producto qr al bin correspondiente
773       bin = (int)(rn[val-1]/Delta);
774       for(int i=0;i<3;i++){
775            DipoleMomentBySlice[bin][i] = DipoleMomentBySlice[bin][i] + rq[i];
776            //cout << DipoleMomentBySlice[bin][i] << "\t";
777            }//cout << endl;
778  }
779
780  void ProductVectorByScalar(double vector[3], double scalar){
781       for(int i =0;i<3;i++){
782            vector[i] = vector[i]*scalar;
783            }
784
785       }
786
787  void PrintProfileCD(){
788       //Desplazamiento cuadratico medio MSD
789       char OutputBin[100]="";
790       char OutputNumber[]="Num";
791       strcat(OutputBin,"CoefDifPorCelda.dat");
792       OutPutBins.open(OutputBin,std::ofstream::out | std::ofstream::app);
793
794       //imprime MSD por componente y total durante el tiempo de simulacion
795       // 6*Pasos*Deltat*NPasos = 6*0.500*0.002*25000
796       OutPutBins << tiempoVar ;
797       for(int j=0;j<3;j++){
798            OutPutBins << " \t" << MSD_comp_Average[j];
799            //OutPutBins << " \t" << MSD_comp_Average[j]/(6*tiempoVar);
800            //OutPutBins << " \t" << AngCua2CmCua(MSD_comp_Average[j])/(6*Ns2s(tiempoVar));
801       }
802       OutPutBins << " \t" << MSD_Average << endl;
803       //OutPutBins << " \t" << MSD_Average/(6*tiempoVar) << endl;
804       //OutPutBins << " \t" << AngCua2CmCua(MSD_Average)/(6*Ns2s(tiempoVar)) << endl;
805       OutPutBins.close();
806       }
807
```

```
808 void ComputeDifference(){
809      double VecDif[3] = {0.0};
810      //ValuesToCero();
811      for(int i=0;i<NumberOfMolec;i++){
812           //Calcula la diferencia con PBC, VecDif entra en cero
813           // y dentro de la funcion se asigna su valor
814           //distance_pbc(VecDif,i);
815           for(int j=0;j<3;j++){
816                VecDif[j] = r_temp[i][j]-r_inic[i][j]; //aqui sin PBC
817
818                //Calcula el producto punto
819                VecDif[j] = VecDif[j]*VecDif[j];
820
821                //Asigna producto punto al MSDTemporal
822                MSDTemp = VecDif[j] + MSDTemp;
823
824                //Asigna producto punto al MSD Temporal por componente (x,y,z)
825                MSDTemp_comp[j] = VecDif[j] + MSDTemp_comp[j];
826                }
827           }
828
829      //Obtiene el MSD promedio en cada punto y lo asigna al MSD global
830      for(int j=0;j<3;j++){
831         //MSD global por componente
832         MSD_comp[j] =  MSDTemp_comp[j]/NumberOfMolec + MSD_comp[j];
833         MSD_comp_Average[j] = MSD_comp[j]/(Replicas+1);
834         //Reinicia
835         MSDTemp_comp[j] = 0.0;
836         }
837
838      //MSD global total
839      MSD = MSDTemp/NumberOfMolec + MSD;
840
841      MSD_Average = MSD/(Replicas+1);
842      MSDTemp = 0.0;
843
844      }
845
846 void ComputeRCMass(int i){
847      double Mtotal;
848
849      //inicializa el vector de centro de masa
850      for(int j=0;j<3;j++){
851           rCM[j] = 0.0;
852           }
853
854      //Calcula masa total molecula de agua
855      Mtotal = 2*MasTypeAtoms[1] + MasTypeAtoms[0];
856
857      //centro de masas por molecula de agua
858      rCM[0] = rCM[0] + MasTypeAtoms[1]*x[i];
859      rCM[1] = rCM[1] + MasTypeAtoms[1]*y[i];
860      rCM[2] = rCM[2] + MasTypeAtoms[1]*z[i];
861
862      rCM[0] = rCM[0] + MasTypeAtoms[0]*x[i+1];
863      rCM[1] = rCM[1] + MasTypeAtoms[0]*y[i+1];
864      rCM[2] = rCM[2] + MasTypeAtoms[0]*z[i+1];
865
866      rCM[0] = rCM[0] + MasTypeAtoms[1]*x[i+2];
867      rCM[1] = rCM[1] + MasTypeAtoms[1]*y[i+2];
868      rCM[2] = rCM[2] + MasTypeAtoms[1]*z[i+2];
869
870      rCM[0] = rCM[0]/Mtotal;
871      rCM[1] = rCM[1]/Mtotal;
872      rCM[2] = rCM[2]/Mtotal;
873
```

```
874       }
875
876 void AssignRtemp(){
877       // asigna arreglo inicial con centros de masa
878       int k = 0;
879       for(int i=0;i<NumberOfMolec;i++){
880             //calcula el centro de masa por molecula
881             ComputeRCMass(k);
882             for(int j=0;j<3;j++){
883                   r_temp[i][j] = rCM[j];
884                   //cout << r_temp[i][j];
885                   }
886                   k=k+3;
887             }
888
889       }
890
891 void distance_pbc(double dif[3],int i){
892       //Calcula vector con PBC
893       double l[3]={SideX,SideY,SideZ};
894             for(int j=0;j<3;j++){
895                   dif[j] = r_temp[i][j]-r_inic[i][j];
896                   dif[j] = dif[j] -l[j]*round(dif[j]/l[j]);
897             }
898 }
899
900 void AssignRinicial(){
901       // asigna arreglo inicial con centros de masa
902       int k = 0;
903       for(int i=0;i<NumberOfMolec;i++){
904             ComputeRCMass(k);
905             for(int j=0;j<3;j++){
906                   r_inic[i][j] = rCM[j];
907                   //cout<< r_inic[i][j];
908                   }
909                   k=k+3;
910             }
911
912       }
913
914 double AssignBinrCM(){
915       int valueR=1000;
916       switch(valor){
917             case 1:
918                   valueR = (int)(rCM[0]/Delta);
919                   break;
920             case 2:
921                   valueR = (int)(rCM[1]/Delta);
922                   break;
923             case 3:
924                   valueR = (int)(rCM[2]/Delta);
925                   break;
926             default:
927                   cout<<"No se asigno la coordenada del CENTRO DE MASA al bin"<<endl;
928                   break;
929       }
930       return valueR;
931 }
932
933 void ValuesToCero(){
934  for(int j=0;j<Bins;j++){
935     NumberOfMolec_bin[j]=0.0;
936      MSDTemp_bin[j]=0.0;
937      for(int i=0;i<3;i++){
938         MSDTemp_comp_bin[i][j]=0.0;
939        }
```

```
940     }
941 }
942
943 void CalcAngles(){
944      switch(NumAtomConsider){
945           case 0: //Hecho para el atomo de oxigeno
946                     for(int j=0;j<Bins;j++){
947                          AcumulaConteo[j][NumAtomConsider] = AcumulaConteo[j][
    NumAtomConsider] + RespDen[j][NumAtomConsider];
948                          }RestartVal();
949                     break;
950           case 1:
951                     //hecho para este atomo
952                     for(int j=0;j<Bins;j++){
953                          AcumulaConteo[j][NumAtomConsider] = AcumulaConteo[j][
    NumAtomConsider] + RespDen[j][NumAtomConsider];
954                          }RestartVal();
955                     break;
956           case 2:
957                     // hecho para este atomo
958                     for(int j=0;j<Bins;j++){
959                          AcumulaConteo[j][NumAtomConsider] = AcumulaConteo[j][
    NumAtomConsider] + RespDen[j][NumAtomConsider];
960                          }RestartVal();
961                     break;
962           case 3:
963                     //Aun no esta hecho para todos los atomos de la molecula
964
965                     break;
966           default:
967                     cout<<"No se asigno el Coseno promedio al bin"<<endl;
968                     break;
969           }
970      }
971
972 void AssignCount(){
973      for(int j=0;j<3;j++){
974           RefBinAng = (int)(acos(Cos[j])/DeltaAngle);
975           //Ese if es para los limites del coseno del angulo, -1 a 1
976           if(Cos[j]>=1){
977                RefBinAng = 0;
978                }
979           if(Cos[j]<=-1){
980                RefBinAng = (int)(M_PI/DeltaAngle);
981                }
982           //Conteo para los tres angulos calculados, por bin, por atomo y por seccion del
     coseno
983           AcumulaAnglesHist[RefBin][NumAtomConsider+j][RefBinAng]++;
984           }
985      }
986
987 void AnglesX(){
988      for(int i=0;i<Particles;i++){
989           RefBin=(int)(x[i]/Delta);
990           if(strcmp(TypeAtomsConsider,AtomID[i])==0){
991                //Contador de incidencias por atomo para los angulos
992                RespDen[RefBin][NumAtomConsider]++;
993                //Si es mayor a la mitad de la caja la orientacion es al otro plano
994                     if(x[i] > SideX/2){
995                          //Se pasan las coordenadas de O, H1, H2 y el plano a analizar
996                          AssignVector(x[i],y[i],z[i],x[i-1],y[i-1],z[i-1],x[i+1],y[i+1],z
    [i+1],SideX);
997                          }
998                     else{
999                          AssignVector(x[i],y[i],z[i],x[i-1],y[i-1],z[i-1],x[i+1],y[i+1],z
    [i+1],0.0);
```

```
1000                        }
1001                    //Calcula el coseno del angulo para el momento dipolar y los dos
    hidrogenos
1002                    Cos[0] = CosineAngle(AngPMHs, AngWall);
1003                    Cos[1] = CosineAngle(AngH1, AngWall);
1004                    Cos[2] = CosineAngle(AngH2, AngWall);
1005                    AssignCount();
1006                    }
1007          }
1008          CalcAngles();
1009     }
1010
1011 void AnglesY(){
1012     for(int i=0;i<Particles;i++){
1013          RefBin=(int)(y[i]/Delta);
1014               if(strcmp(TypeAtomsConsider,AtomID[i])==0){
1015                    //Contador de incidencias por atomo para los angulos
1016                    RespDen[RefBin][NumAtomConsider]++;
1017                    //Si es mayor a la mitad de la caja la orientacion es al otro plano
1018                    if(y[i] < SideY/2){
1019                         //Se pasan las coordenadas de O, H1, H2 y el plano a analizar
1020                         AssignVector(x[i],y[i],z[i],x[i-1],y[i-1],z[i-1],x[i+1],y[i+1],z
    [i+1],SideY);
1021                         }
1022                    else{
1023                         AssignVector(x[i],y[i],z[i],x[i-1],y[i-1],z[i-1],x[i+1],y[i+1],z
    [i+1],0.0);
1024                         }
1025                    if(y[i] != 0){
1026                    //Calcula el coseno del angulo para el momento dipolar y los dos
    hidrogenos
1027                    Cos[0] = CosineAngle(AngPMHs, AngWall);
1028                    Cos[1] = CosineAngle(AngH1, AngWall);
1029                    Cos[2] = CosineAngle(AngH2, AngWall);
1030                    AssignCount();
1031                     }
1032                    }
1033          }
1034          CalcAngles();
1035     }
1036
1037 void AnglesZ(){
1038     for(int i=0;i<Particles;i++){
1039          RefBin=(int)(z[i]/Delta);
1040          if(strcmp(TypeAtomsConsider,AtomID[i])==0){
1041               //Contador de incidencias por atomo para los angulos
1042               RespDen[RefBin][NumAtomConsider]++;
1043               //Si es mayor a la mitad de la caja la orientacion es al otro plano
1044                    if(z[i] > SideZ/2){
1045                         //Se pasan las coordenadas de O, H1, H2 y el plano a analizar
1046                         AssignVector(x[i],y[i],z[i],x[i-1],y[i-1],z[i-1],x[i+1],y[i+1],z
    [i+1],SideZ);
1047                         }
1048                    else{
1049                         AssignVector(x[i],y[i],z[i],x[i-1],y[i-1],z[i-1],x[i+1],y[i+1],z
    [i+1],0.0);
1050                         }
1051                    //Calcula el coseno del angulo para el momento dipolar y los dos
    hidrogenos
1052                    Cos[0] = CosineAngle(AngPMHs, AngWall);
1053                    Cos[1] = CosineAngle(AngH1, AngWall);
1054                    Cos[2] = CosineAngle(AngH2, AngWall);
1055                    AssignCount();
1056                    }
1057          }
1058          CalcAngles();
```

```
1059      }
1060
1061 void AssignVector(double POneX, double POneY,double POneZ,double PTwoX, double PTwoY,
     double PTwoZ, double PThreeX, double PThreeY,double PThreeZ,double L){
1062      //Pone es oxigeno, Ptwo es hidrogeno 1 y Pthree es hidrogeno 3
1063      //L es el valor donde se encuentra el plano, 0 o Ly=295.584
1064      //Vector oxigeno a punto medio de hidrogenos
1065      //Coordenada punto medio en hidrogenos, menos la coordenada del Oxigeno
1066      AngPMHs[0] = ( PTwoX + PThreeX )/2 - POneX;
1067      AngPMHs[1] = ( PTwoY + PThreeY )/2 - POneY;
1068      AngPMHs[2] = ( PTwoZ + PThreeZ )/2 - POneZ;
1069
1070      //Vector oxigeno a hidrogeno 1
1071      AngH1[0] = PTwoX - POneX;
1072      AngH1[1] = PTwoY - POneY;
1073      AngH1[2] = PTwoZ - POneZ;
1074
1075      //Vector oxigeno a hidrogeno 2
1076      AngH2[0] = PThreeX - POneX;
1077      AngH2[1] = PThreeY - POneY;
1078      AngH2[2] = PThreeZ - POneZ;
1079      /*
1080       * A continuacion, se asignan las coordenadas del punto en el MURO,
1081       * dependiendo de la direccion del analisis,
1082       * considerando que el muro esta en el 0,0,0 y en donde termina la caja.
1083       * El +1 es para evitar tener vector nulo, si se hace un analisis para agua pura,
1084       * donde en principio puede tener la coordenada en 0,0,0.
1085       * El vector AngWall va de la superfice al oxigeno
1086       * */
1087      switch(valor){
1088           //Oxigeno menos el plano mas uno
1089           case 1:
1090                     AngWall[0] = POneX -L +1;
1091                     AngWall[1] = 0.0;
1092                     AngWall[2] = 0.0;
1093                    break;
1094           case 2:
1095                     AngWall[0] = 0.0;
1096                     AngWall[1] = POneY - L;
1097                     AngWall[2] = 0.0;
1098                    break;
1099           case 3:
1100                     AngWall[0] = 0.0;
1101                     AngWall[1] = 0.0;
1102                     AngWall[2] = POneZ -L +1;
1103                    break;
1104           default:
1105                     cout<<"No se asigno Vector Correctamente para calcular Angulos"<<endl
     ;
1106                     break;
1107           }
1108
1109      }
1110
1111 double CosineAngle(double v1[3],double v2[3]){
1112      //esta funcion calcula el coseno entre dos vectores
1113    double coseno=0.0,MagnitudV1=0.0,MagnitudV2=0.0;
1114    for(int i=0;i<3;i++){
1115           coseno = coseno + v1[i]*v2[i];
1116           MagnitudV1 = MagnitudV1 + v1[i]*v1[i];
1117           MagnitudV2 = MagnitudV2 + v2[i]*v2[i];
1118      }
1119      return coseno/sqrtl(MagnitudV1*MagnitudV2);
1120     }
1121
1122 void AssignProfile(){
```

```
1123      NumAtomConsider=1000;
1124      for(int j=0;j<=TypeAtoms;j++){
1125           if(strcspn(TypeAtomsConsider,AtomSearch[j])==0){
1126                     NumAtomConsider=j;
1127                cout<<"Entro esta opcion: "<<j<<" "<<AtomSearch[j]<<endl;
1128                }
1129            /*Los valores seleccionados son para oxigeno cero,
1130             * cualquiera de los hidrogenos 2 porque no se distingue entre uno y otro,
1131             * toma 1 para H1 pero pasa a 2 en H2 porque compara las H y 3 para all.
1132            */
1133           }
1134      if(NumAtomConsider==1000){
1135                cout<<"No se pude asignar el tipo de atomo para el perfil, valor de: "<<
     NumAtomConsider;
1136                cout<<" y deberia ser cero, uno, dos o tres"<<endl;
1137                }
1138      }
1139
1140 int Alerts(){
1141      //Alerta si sobrepasa el limite de los arreglos
1142      int condition=1;
1143      if(Bins>MaxOfBins){
1144           cout<<"El numero de bins proporcionado es mayor al maximo definido, cambiar en
     codigo\n"<<endl;
1145           condition=0;
1146           }
1147       else{
1148           condition=2;
1149           }
1150      if(Configuration>MaxConfigur){
1151           cout<<"El numero de configuraciones en el archivo es mayor al maximo definido,
     cambiar en codigo\n"<<endl;
1152           condition=0;
1153           }
1154       else{
1155           condition=2;
1156           }
1157      return condition;
1158      }
1159
1160 void RestartVal(){
1161           for(int j=0;j<Bins;j++){
1162                Density[j]=0.0;
1163                for(int i=0;i<=TypeAtoms;i++){
1164                      RespDen[j][i]=0.0;
1165                      AcumulaAngles[j][i] = 0.0;
1166                }
1167           }
1168      }
1169
1170 void DensityZ(){
1171      /*compara si pertenece al tipo de atomo para contarlos,
1172       * primero identifica el delta y luego el tipo de atomo
1173     */
1174      for(int i=0;i<Particles;i++){
1175           for(int j=0;j<TypeAtoms;j++){
1176                if(strcmp(AtomID[i],AtomSearch[j])==0){
1177                     RespDen[(int)(z[i]/Delta)][j]++;
1178                     }
1179                }
1180           }
1181           CalcDensity();
1182      }
1183
1184 void DensityY(){
1185      /*compara si pertenece al tipo de atomo para contarlos,
```

```
1186        * primero identifica el delta y luego el tipo de atomo
1187      */
1188       for(int i=0;i<Particles;i++){
1189            for(int j=0;j<TypeAtoms;j++){
1190                 if(strcmp(AtomID[i],AtomSearch[j])==0){
1191                      RespDen[(int)(y[i]/Delta)][j]++;
1192                      }
1193                 }
1194            }
1195            CalcDensity();
1196       }
1197
1198  void DensityX(){
1199       /*compara si pertenece al tipo de atomo para contarlos,
1200        * primero identifica el delta y luego el tipo de atomo
1201      */
1202            for(int i=0;i<Particles;i++){
1203                 for(int j=0;j<TypeAtoms;j++){
1204                      if(strcmp(AtomID[i],AtomSearch[j])==0){
1205                           RespDen[(int)(x[i]/Delta)][j]++;
1206                           }
1207                      }
1208            }
1209            CalcDensity();
1210       }
1211
1212  void PrintDistribution(){
1213       freopen("Distribucion.dat","w",stdout);
1214       for(int j=0;j<100;j++){
1215            //Formato para imprimir por columnas ordenadas y con numero de decimales fija
1216            FormatPrint();
1217            cout<<j<<"  \t"<<distribution[j]<<endl;
1218            }
1219       fclose(stdout);
1220       }
1221
1222  void PrintProfileAngles(){
1223       //Perfiles de angulos por bin
1224       char OutputBin[100]="";
1225       char OutputBinConteo[100]="";
1226       char OutputNumber[]="Num";
1227       double DeltaSolid = DeltaAngle/2, SolidAngle = 0;
1228       int inicio = int(DeltaDegree/2), incremento = DeltaDegree;;
1229       for(int k=0;k<Bins;k++){
1230            sprintf(OutputNumber,"%d",k+1);
1231            strcat(OutputBin,"PerfilAnguloBin");
1232            strcat(OutputBin,OutputNumber);
1233            strcat(OutputBin,".txt");
1234            OutPut.open(OutputBin);
1235            for(int l=inicio;l<HalfAngles;l=l+incremento){
1236                 FormatPrint();
1237                 //Angulo Solido
1238                 SolidAngle = 2*M_PI*(cos(Deg2Rad(l)-DeltaSolid)-cos(Deg2Rad(l)+DeltaSolid)
     );
1239                 //SolidAngle = 2*M_PI*(1.0 - cos(Deg2Rad(l)+DeltaAngle));
1240                 OutPut<<cos(Deg2Rad(l))<<" \t"<<AcumulaAnglesHist[k][NumAtomConsider][(int
     )(Deg2Rad(l)/DeltaAngle)]/AcumulaConteo[k][NumAtomConsider];
1241                 OutPut<<" \t"<<AcumulaAnglesHist[k][NumAtomConsider+1][(int)(Deg2Rad(l)/
     DeltaAngle)]/AcumulaConteo[k][NumAtomConsider];
1242                 OutPut<<" \t"<<AcumulaAnglesHist[k][NumAtomConsider+2][(int)(Deg2Rad(l)/
     DeltaAngle)]/AcumulaConteo[k][NumAtomConsider];//<<endl;
1243                 OutPut<<" \t"<<AcumulaAnglesHist[k][NumAtomConsider][(int)(Deg2Rad(l)/
     DeltaAngle)]/AcumulaConteo[k][NumAtomConsider]/SolidAngle;
1244                 //OutPut<<"\t"<<SolidAngle<<" \t"<<AcumulaAnglesHist[k][NumAtomConsider][(
     int)(Deg2Rad(l)/DeltaAngle)]/AcumulaConteo[k][NumAtomConsider];
```

```
1245                OutPut<<" \t"<<AcumulaAnglesHist[k][NumAtomConsider+1][(int)(Deg2Rad(1)/
    DeltaAngle)]/AcumulaConteo[k][NumAtomConsider]/SolidAngle;
1246                OutPut<<" \t"<<AcumulaAnglesHist[k][NumAtomConsider+2][(int)(Deg2Rad(1)/
    DeltaAngle)]/AcumulaConteo[k][NumAtomConsider]/SolidAngle<<endl;
1247                }
1248          OutPut.close();
1249          strcpy(OutputBin,"");
1250          }
1251     }
1252
1253 void PrintProfile(){
1254     //Perfil de densidad promedio de n puntos de la trayectoria
1255     sprintf(Output,"%d",Bins);
1256     strcat(Output,Output1);
1257     strcat(Output,TypeAtomsConsider);
1258     strcat(Output,".txt");
1259     cout<<"Perfil de Densidad Guardado en: "<<Output<<endl;
1260     freopen(Output,"w",stdout);
1261     for(int j=0;j<Bins;j++){
1262          //Formato para imprimir por columnas ordenadas y con numero de decimales fija
1263          FormatPrint();
1264          //cout<<Delta*(j+1)-Delta/2<<"  \t"<<rep[j]/Configuration<<endl;
1265          if(ContadorRep[j]==0) ContadorRep[j]=1;
1266          cout<<Delta*(j+1)-Delta/2<<"  \t"<<rep[j]/ContadorRep[j]<<endl;
1267          }
1268     fclose(stdout);
1269     }
1270
1271 void FormatPrint(){
1272     //notacion fija a punto decimal
1273     cout<<setiosflags(ios::fixed);
1274     //6 numeros despues del punto y lo que no se ocupa de los 12 espacios se llena con
    espacio vacio
1275     cout<<setw(12)<<setprecision(6)<<setfill(' ');
1276     }
1277
1278 void CalcDensity(){
1279     //calcula densidad en unidades g/mL por bin
1280     for(int j=0;j<Bins;j++){
1281          ProfileSelected(j);
1282          Density[j]=(RespDen[j][TypeAtoms]*1.66)/DeltaVolumen;
1283          if(Density[j]!=0){ContadorRep[j]++; }
1284               rep[j]=Density[j]+rep[j];
1285          }
1286          RestartVal();
1287     }
1288
1289 void ProfileSelected(int j){
1290     //Calcula la densidad dependiendo el atomo a considerar O,H1,H2
1291     switch(NumAtomConsider){
1292          case 0: //Solo atomo de oxigeno
1293                  //RespDen[j][TypeAtoms]=RespDen[j][TypeAtoms] + RespDen[j][
    NumAtomConsider]*MasTypeAtoms[NumAtomConsider];
1294                  //Solo atomo de oxigeno pero con masa del agua
1295                  RespDen[j][TypeAtoms]=RespDen[j][TypeAtoms] + RespDen[j][
    NumAtomConsider]*MasAgua;
1296                  break;
1297          case 1: //Todos los atomos de Hidrogeno
1298                  RespDen[j][TypeAtoms]=RespDen[j][TypeAtoms] + RespDen[j][
    NumAtomConsider]*MasTypeAtoms[NumAtomConsider];
1299                  break;
1300          case 2: //Todos los atomos de Hidrogeno
1301                  RespDen[j][TypeAtoms]=RespDen[j][TypeAtoms] + RespDen[j][
    NumAtomConsider]*MasTypeAtoms[NumAtomConsider];
1302                  break;
1303          case 3: //Todos los atomos
```

```
1304                        for(int l=0;l<TypeAtoms;l++){
1305                              RespDen[j][TypeAtoms]=RespDen[j][TypeAtoms] + RespDen[j][l]*
    MasTypeAtoms[l];
1306                              }
1307                        break;
1308              default:
1309                        cout<<"No se calculo densidad ni por atomo ni por toda la molecula"<<
    endl;
1310                        break;
1311              }
1312       }
1313
1314 void AssignDensity(){
1315       //Indica en que direccion se hara el analisis
1316       switch(valor){
1317              case 1:
1318                        DensityX();
1319                       break;
1320              case 2:
1321                        DensityY();
1322                       break;
1323              case 3:
1324                        DensityZ();
1325                       break;
1326              default:
1327                        break;
1328              }
1329       }
1330
1331 void AssignAngles(){
1332       //Indica en que direccion se hara el analisis
1333       switch(valor){
1334              case 1:
1335                        AnglesX();
1336                       break;
1337              case 2:
1338                        AnglesY();
1339                       break;
1340              case 3:
1341                        AnglesZ();
1342                       break;
1343              default:
1344                        break;
1345              }
1346       }
1347
1348 void ComputeDelta(int val){
1349       //Calcula los valores de delta en el eje y de volumen
1350       switch(val){
1351            case 1:
1352                        Delta=SideX/Bins;
1353                        DeltaVolumen=SideY*SideZ*Delta;
1354                       break;
1355              case 2:
1356                        Delta=SideY/Bins;
1357                        DeltaVolumen=SideX*SideZ*Delta;
1358                       break;
1359              case 3:
1360                        Delta=SideZ/Bins;
1361                        DeltaVolumen=SideX*SideY*Delta;
1362                       break;
1363              default:
1364                        break;
1365              }
1366       }
1367
```

```
1368 void SelectCoordinate(){
1369
1370     CoordinateSelection();
1371      //calcula deltas dependiento la direccion del perfil
1372      ComputeDelta(valor);
1373
1374      //inicializar arreglo
1375      for(int j=0;j<Bins;j++){
1376                rep[j] = 0;
1377                repH1[j] = 0.0;
1378                repH2[j] = 0.0;
1379      }
1380      RestartVal();
1381      }
1382
1383 void CoordinateSelection(){
1384      if(strcmp(Coordinate,"x")==0 || strcmp(Coordinate,"X")==0){valor=1;}
1385      if(strcmp(Coordinate,"y")==0 || strcmp(Coordinate,"Y")==0){valor=2;}
1386      if(strcmp(Coordinate,"z")==0 || strcmp(Coordinate,"Z")==0){valor=3;}
1387      }
1388
1389 void fileParticlesConfiguration(char* file){
1390 //lee de un archivo xyz la primer linea y asigna el numero leido a la variable particulas
1391      file_xyz=fopen(file,"r");
1392      fgets(LineRead,100,file_xyz); sscanf(LineRead,"%d",&Particles);
1393      fclose(file_xyz);
1394      Configuration=LinesFile/(Particles+2);
1395 }
1396
1397 void fileParticlesConfigurationPDB(char* file){
1398      //lee de un archivo pdb TER y asigna el # atomos
1399      char TerPDB[20];
1400      //Determina cuantos atomos hay
1401      file_pdb=fopen(file,"r");
1402      if(file_pdb!=NULL){
1403            while(strcmp(TerPDB,"TER")!=0){
1404                  fgets(LineRead,100,file_pdb);
1405                  sscanf(LineRead,"%s %d %*s",TerPDB,&Particles);
1406            }
1407      }
1408      fclose(file_pdb);
1409      Particles--;
1410      // -2 por REMARK y CRYST1 y, +3 por MODEL, TER y ENDMDL
1411      Configuration=(LinesFile-2)/(Particles+3);
1412 }
1413
1414 int ReadingChange(char* file){
1415      int AtomsByMolecule=0,AtomsByMolAux=0;
1416      //Determina cuantos atomos hay por molecula
1417      file_pdb=fopen(file,"r");
1418      if(file_pdb!=NULL){
1419           //lineas no necesarias para análisis
1420            fgets(LineRead,100,file_pdb); //REMARK
1421            fgets(LineRead,100,file_pdb); //CRYST1
1422            fgets(LineRead,100,file_pdb); //MODEL
1423            for(int i=0;i<Particles;i++){
1424                  fgets(LineRead,100,file_pdb);
1425                  sscanf(LineRead,"%*s %*d %*s %*s %*s %d %*s",&AtomsByMolAux);
1426                  if(i==0){
1427                        AtomsByMolecule = AtomsByMolAux;
1428                  }
1429                  if(AtomsByMolecule!=AtomsByMolAux){
1430                        AtomsByMol = i;
1431                        OneLine = AtomsByMol*999;
1432                        break;
1433                  }
```

```
1434            }
1435      }
1436      if(Particles > OneLine){
1437            cout<<"################## A D V E R T E N C I A ##############################"<<
      endl;
1438            cout<<"Átomos por Molécula: "<<AtomsByMol<<" A1000 sucede en el átomo: "<<
      OneLine<<endl;
1439            cout<<"Lee las primeras líneas de la subrutina.cpp"<<endl;
1440            cout<<"################## A D V E R T E N C I A ##############################"<<
      endl;
1441      }
1442      fclose(file_pdb);
1443      return OneLine;
1444 }
1445
1446 void ReadFileDataGraph(){
1447      //lee los datos de la distancia de cada átomo
1448      for(int i=0;i<LinesFile;i++){
1449            fgets(LineRead,100,file_xyz);
1450            sscanf(LineRead,"%d",&atoms[i]);
1451            //printf("%4d \n",atoms[i]);
1452            }
1453      }
1454
1455 void ReadFileValuesVelocities(){
1456      //lineas de datos que no importan para asignar velocidades
1457      fgets(LineRead,100,file_xyz);fgets(LineRead,100,file_xyz);
1458                for(int i=0;i<Particles;i++){
1459                      fgets(LineRead,100,file_xyz);
1460                      sscanf(LineRead,"%f %f %f",&x[i],&y[i],&z[i]);
1461                      //printf("%10.6lf \t%10.6lf \t%10.6lf\n",x[i],y[i],z[i]);
1462                      }
1463      }
1464
1465 void ReadFileValues(){
1466      //lineas de datos que no importan para asignar coordenadas
1467      fgets(LineRead,100,file_xyz);fgets(LineRead,100,file_xyz);
1468                for(int i=0;i<Particles;i++){
1469                      fgets(LineRead,100,file_xyz);
1470                      sscanf(LineRead,"%s %f %f %f",AtomID[i],&x[i],&y[i],&z[i]);
1471                      //printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[i],x[i],y[i],
      z[i]);
1472                      }
1473      }
1474
1475 void ReadFileValuesPDB(){
1476      fgets(LineRead,100,file_pdb);  //Linea MODEL
1477      //Ciclo para asignar coordenadas y tipo de átomo
1478      for(int i=0;i<Particles;i++){
1479            fgets(LineRead,100,file_pdb);
1480            if( i < OneLine ){//Los espacios entre A y el número de moléculas desaparecen,
      A1000
1481                sscanf(LineRead,"%*s %*d %s %*s %*s %*d %f %f %f %*s",AtomID[i],&x[i],&y
      [i],&z[i]);
1482                printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[i],x[i],y[i],z[i]);
1483            }
1484            else{
1485                 if( i < 9999 ){ //Los espacios entre HETATM; y el número de átomo
      desaparecen HETATM10000
1486                      sscanf(LineRead,"%*s %*d %s %*s %*s %f %f %f %*s",AtomID[i],&x[i
      ],&y[i],&z[i]);
1487                      printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[i],x[i],y[i],z[
      i]);
1488                }
1489                else{
```

```
1490                    sscanf(LineRead,"%*s %s %*s %*s %lf %lf %lf %*s",AtomID[i],&x[i],&y[i
    ],&z[i]);
1491                    printf("%-4s  \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[i],x[i],y[i],z
    [i]);
1492               }
1493          }
1494      }
1495      fgets(LineRead,100,file_pdb); //Linea TER
1496      fgets(LineRead,100,file_pdb); //Linea ENDMDL
1497 }
1498
1499 void TranslatefullBox(){
1500      //Traslada la caja completa como la segunda caja en tres replicas
1501      printf("%d \n\n",Particles);
1502      for(int i=0;i<Particles;i++){
1503           if(i%3==1){
1504                printf("H1 \t %10.6lf \t%10.6lf \t%10.6lf\n",x[i-1],y[i-1]+0.5*SideY,z[i-
    1]);
1505                printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[i],x[i],y[i]+0.5*
    SideY,z[i]);
1506                printf("H2 \t %10.6lf \t%10.6lf \t%10.6lf\n",x[i+1],y[i+1]+0.5*SideY,z[i
    +1]);
1507                }
1508
1509           }
1510      }
1511
1512 void TranslateBox(){
1513 /*longitudes de la caja en y para L1,L2,L3,L4,L5
1514  * considerando dos espacios cerca del centro con 10 angstroms de ancho*/
1515   float l[6]={73.896,83.896,231.688,241.688,295.584,315.584},inc=10;
1516   printf("%d \n\n",Particles);
1517      for(int i=0;i<Particles;i++){
1518           //If para el oxigeno y reacomodar los hidrogenos con dos vacios en la caja
1519           if(i%3==1){
1520                //Los extremos de la caja
1521                if(y[i]<=l[0] || y[i]>= l[3]){
1522                     printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[i-1],x[i-1],y[i
    -1],z[i-1]);
1523                     printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[i],x[i],y[i],z[
    i]);
1524                     printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[i+1],x[i+1],y[i
    +1],z[i+1]);
1525                     }
1526                else{
1527                //El centro de la caja, primer vacio l1 y l2 ocupo y -l1 + ly
1528                     if(y[i]>=l[0] && y[i]<=l[1]){
1529                          printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[i-1],x[i-
    1],y[i-1]-l[0]+l[4],z[i-1]);
1530                          printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[i],x[i],y[
    i]-l[0]+l[4],z[i]);
1531                          printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[i+1],x[i
    +1],y[i+1]-l[0]+l[4],z[i+1]);
1532                          }
1533                     else{
1534                          if(y[i]>=l[1] && y[i]<=l[2]){
1535                               printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[i-1],
    x[i-1],y[i-1],z[i-1]);
1536                               printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[i],x[
    i],y[i],z[i]);
1537                               printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[i+1],
    x[i+1],y[i+1],z[i+1]);
1538                               }
1539                          else{
1540                               if(y[i]>=l[2] && y[i]<=l[3]){
```

```
1541                                    printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[
    i-1],x[i-1],y[i-1]-l[2]+l[4]+inc,z[i-1]);
1542                                    printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[
    i],x[i],y[i]-l[2]+l[4]+inc,z[i]);
1543                                    printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[
    i+1],x[i+1],y[i+1]-l[2]+l[4]+inc,z[i+1]);
1544                                    }
1545                                else{
1546                                    printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[
    i-1],x[i-1],y[i-1],z[i-1]);
1547                                    printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[
    i],x[i],y[i],z[i]);
1548                                    printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[
    i+1],x[i+1],y[i+1],z[i+1]);
1549                                    }
1550                                }
1551                            }
1552                        }
1553                }
1554
1555            }
1556
1557      }
1558
1559 void PrintTip4pToTip3p(){
1560      //Imprime la trayectoria sin la particula extra "M"
1561      printf("%d \n\n",Particles*3/4);
1562      for(int i=0;i<Particles;i++){
1563            if(strcmp(AtomID[i],"M")!=0){
1564                printf("%-4s \t %10.6lf \t%10.6lf \t%10.6lf\n",AtomID[i],x[i],y[i],z[i]);
1565                }
1566            }
1567      }
1568
1569 int LongitudArchivo(char* dir){
1570         FILE *f;
1571         //obtiene la cantidad de lineas del archivo
1572         int l=0;
1573         char linea[100];
1574         f=fopen(dir, "r");
1575         if(f!=NULL){
1576                 while(!feof(f)){
1577                         fgets(linea,100,f);
1578                         l++;
1579                 }
1580                 fclose(f);
1581         }
1582         return l-1;
1583 }
```

7.10 Configuración de agua estilo emparedado

Configuración disponible en:
https://www.dropbox.com/sh/wxudc7a7prtns1v/AADGLeecMd7zPfiLszVdJB0ka?dl=0

7.11 Configuración equilibrada de agua durante 100 ns

Configuración disponible en:
https://www.dropbox.com/sh/wxudc7a7prtns1v/AADGLeecMd7zPfiLszVdJB0ka?dl=0

7.12 Configuración equilibrada de agua más superficie hidrofóbica durante 1000 ns

Configuración disponible en:
https://www.dropbox.com/sh/wxudc7a7prtns1v/AADGLeecMd7zPfiLszVdJB0ka?dl=0

7.13 Configuración equilibrada de agua más superficie hidrofílica durante 1000 ns

Configuración disponible en:
https://www.dropbox.com/sh/wxudc7a7prtns1v/AADGLeecMd7zPfiLszVdJB0ka?dl=0
Las configuraciones de las superficies cargadas también estan en esta carpeta.

7.14 Campo de fuerzas del modelo TIP4P-FB (xml)

```
<ForceField>
 <Info>
  <DateGenerated>2014-05-28</DateGenerated>
  <Reference>Lee-Ping Wang, Todd J. Martinez and Vijay S. Pande. Building force fields -
    an automatic, systematic and reproducible approach. Journal of Physical Chemistry
    Letters, 2014, 5, pp 1885-1891. DOI:10.1021/jz500737m</Reference>
 </Info>
 <AtomTypes>
  <Type name="tip4p-fb-O" class="OW" element="O" mass="15.99943"/>
  <Type name="tip4p-fb-H" class="HW" element="H" mass="1.007947"/>
  <Type name="tip4p-fb-M" class="MW" mass="0"/>
 </AtomTypes>
 <Residues>
  <Residue name="HOH">
   <Atom name="O" type="tip4p-fb-O"/>
   <Atom name="H1" type="tip4p-fb-H"/>
   <Atom name="H2" type="tip4p-fb-H"/>
   <Atom name="M" type="tip4p-fb-M"/>
   <VirtualSite type="average3" index="3" atom1="0" atom2="1" atom3="2" weight1="8.
    203146574531e-01" weight2="8.984267127345e-02" weight3="8.984267127345e-02" />
   <Bond from="0" to="1"/>
   <Bond from="0" to="2"/>
  </Residue>
 </Residues>
 <HarmonicBondForce>
  <Bond class1="OW" class2="HW" length="0.09572" k="462750.4"/>
 </HarmonicBondForce>
 <HarmonicAngleForce>
  <Angle class1="HW" class2="OW" class3="HW" angle="1.82421813418" k="836.8"/>
 </HarmonicAngleForce>
 <NonbondedForce coulomb14scale="0.833333" lj14scale="0.5">
  <Atom type="tip4p-fb-O" charge="0" sigma="3.165552430462e-01" epsilon="7.492790213533e-
    01" />
  <Atom type="tip4p-fb-H" charge="5.258681106763e-01" sigma="1" epsilon="0" />
  <Atom type="tip4p-fb-M" charge="-1.0517362213526e+00" sigma="1" epsilon="0" />
 </NonbondedForce>
</ForceField>
```

7.15 Campo de fuerzas del modelo AMOEBA (xml)

```
<ForceField>
 <Info>
  <Source>amoebapro13.prm</Source>
  <DateGenerated>2015-02-19</DateGenerated>
 </Info>
```

```
<AtomTypes>
 <Type name="247" class="43" element="O" mass="15.999"/>
 <Type name="248" class="44" element="H" mass="1.008"/>
</AtomTypes>
<Residues>
 <Residue name="HOH">
  <Atom name="H1" type="248" />
  <Atom name="H2" type="248" />
  <Atom name="O" type="247" />
  <Bond from="0" to="2" />
  <Bond from="1" to="2" />
</Residue>
</Residues>
<AmoebaBondForce bond-cubic="-25.5" bond-quartic="379.3125">
 <Bond class1="43" class2="44" length="0.09572" k="232986.04"/>
</AmoebaBondForce>
<AmoebaAngleForce angle-cubic="-0.014" angle-quartic="5.6e-05" angle-pentic="-7e-07"
   angle-sextic="2.2e-08">
 <Angle class1="44" class2="43" class3="44" k="0.0620690891499" angle1="108.50"  />
</AmoebaAngleForce>
<AmoebaVdwForce type="BUFFERED-14-7" radiusrule="CUBIC-MEAN" radiustype="R-MIN"
   radiussize="DIAMETER" epsilonrule="HHG" vdw-13-scale="0.0" vdw-14-scale="1.0" vdw-15-
   scale="1.0" >
 <Vdw class="43" sigma="0.3405" epsilon="0.46024" reduction="1.0" />
 <Vdw class="44" sigma="0.2655" epsilon="0.056484" reduction="0.910" />
</AmoebaVdwForce>
<AmoebaMultipoleForce  direct11Scale="0.0"  direct12Scale="1.0"  direct13Scale="1.0"
   direct14Scale="1.0"  mpole12Scale="0.0"  mpole13Scale="0.0"  mpole14Scale="0.4"
   mpole15Scale="0.8"  mutual11Scale="1.0"  mutual12Scale="1.0"  mutual13Scale="1.0"
   mutual14Scale="1.0"  polar12Scale="0.0"  polar13Scale="0.0"  polar14Intra="0.5"
   polar14Scale="1.0"  polar15Scale="1.0"  >
 <Multipole type="247" kz="248" kx="-248" c0="-0.51966" d1="0.0" d2="0.0" d3="0.
   00755612136146" q11="0.000354030721139" q21="0.0" q22="-0.000390257077096" q31="0.0"
   q32="0.0" q33="3.62263559571e-05"  />
 <Multipole type="248" kz="247" kx="248" c0="0.25983" d1="-0.00204209484795" d2="0.0" d3
   ="-0.00307875299958" q11="-3.42848248983e-05" q21="0.0" q22="-0.000100240875193" q31=
   "-1.89485963908e-06" q32="0.0" q33="0.000134525700091"  />
 <Polarize type="247" polarizability="0.000837" thole="0.3900" pgrp1="248" />
 <Polarize type="248" polarizability="0.000496" thole="0.3900" pgrp1="247" />
</AmoebaMultipoleForce>
<AmoebaUreyBradleyForce cubic="0.0" quartic="0.0"  >
 <UreyBradley class1="44" class2="43" class3="44" k="-3179.84" d="0.15326" />
</AmoebaUreyBradleyForce>
</ForceField>
```

7.16 Campo de fuerzas del modelo iAMOEBA (xml)

```
<ForceField>
 <AtomTypes>
  <Type name="380" class="73" element="O" mass="15.999"/>
  <Type name="381" class="74" element="H" mass="1.008"/>
 </AtomTypes>
 <Residues>
  <Residue name="HOH">
   <Atom name="H1" type="381"/>
   <Atom name="H2" type="381"/>
   <Atom name="O" type="380"/>
   <Bond from="0" to="2"/>
   <Bond from="1" to="2"/>
 </Residue>
 </Residues>
  <AmoebaBondForce bond-cubic="-25.5" bond-quartic="379.3125">
   <Bond class1="73" class2="74" length="9.584047e-02" k="2.3331232e+05"/>
 </AmoebaBondForce>
```

```
  <AmoebaAngleForce angle-cubic="-0.014" angle-quartic="5.6e-05" angle-pentic="-7e-07" angle-sextic="2.2e-08">
   <Angle class1="74" class2="73" class3="74" k="6.359379296918e-02" angle1="1.064826e+02"/>
 </AmoebaAngleForce>
  <AmoebaOutOfPlaneBendForce type="ALLINGER" opbend-cubic="-0.014" opbend-quartic="5.6e-05" opbend-pentic="-7e-07" opbend-sextic="2.2e-08">
    <!-- LPW: Mark's force field parsing code requires AmoebaOutOfPlaneBendForce in order to read AmoebaAngleForce, even if the clause is empty -->
 </AmoebaOutOfPlaneBendForce>
  <AmoebaVdwForce type="BUFFERED-14-7" radiusrule="CUBIC-MEAN" radiustype="R-MIN" radiussize="DIAMETER" epsilonrule="HHG" vdw-13-scale="0.0" vdw-14-scale="1.0" vdw-15-scale="1.0">
   <Vdw class="73" sigma="3.645297e-01" epsilon="8.2348e-01" reduction="1.0"/>
   <Vdw class="74" sigma="0.0" epsilon="0.0" reduction="1.0"/>
 </AmoebaVdwForce>
  <AmoebaMultipoleForce direct11Scale="0.0" direct12Scale="1.0" direct13Scale="1.0" direct14Scale="1.0" mpole12Scale="0.0" mpole13Scale="0.0" mpole14Scale="0.4" mpole15Scale="0.8" mutual11Scale="1.0" mutual12Scale="1.0" mutual13Scale="1.0" mutual14Scale="1.0" polar12Scale="0.0" polar13Scale="0.0" polar14Intra="0.5" polar14Scale="1.0" polar15Scale="1.0">
   <Multipole type="380" kz="-381" kx="-381" c0="-5.94024e-01" d1="0.0" d2="0.0" d3="4.682021361460e-03" q11="2.111247211390e-04" q21="0.0" q22="-3.009710770960e-04" q31="0.0" q32="0.0" q33="8.984635595700e-05"/>
   <Multipole type="381" kz="380" kx="381" c0="2.97012e-01" d1="-4.969244847950e-03" d2="0.0" d3="-6.646702999580e-03" q11="1.750551751017e-04" q21="0.0" q22="2.029112480700e-05" q31="-3.392685963908e-05" q32="0.0" q33="-1.953462999087e-04"/>
   <Polarize type="380" polarizability="8.063631227791e-04" thole="2.36164e-01" pgrp1="381"/>
   <Polarize type="381" polarizability="5.048434386104e-04" thole="2.36164e-01" pgrp1="380"/>
 </AmoebaMultipoleForce>
  <AmoebaUreyBradleyForce cubic="0.0" quartic="0.0">
   <UreyBradley class1="74" class2="73" class3="74" k="-4.31294e+03" d="1.535676676685e-01"/>
 </AmoebaUreyBradleyForce>
</ForceField>
```

7.17 Script de Python para crear puntos de continuación de DM (xml)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Este script crea un archivo.xml con las posiciones y velocidades de un sistema para simularlo en OpenMM.
Los datos de las posiciones y velocidades se pasan por consola, primero las posiciones y luego las velocidades
la forma de ejecutar el script es la siguiente:
++++++ python script.py posiciones.xyz velocidades.xyz
Los datos a modificar soni:
++++++ NombreArchivoSalida ---> si quieres nombrar el archivo de salida de otra forma
++++++  datos ---> si pasas mas o menos datos (cero para script.py y aumenta por argumento pasado)
++++++ PBC ---> este arreglo tiene los valores de las dimensiones de la celda de simulación
++++++ NPT ---> es una variable logica, cambiar a True si el ensamble es NPT y modificar la linea 88
"""
import sys

""" ##################### Datos a modificar ################################"""

datos=3
NombreArchivoSalida ="Input.xml"
```

```
PBC=[19.705,295.584,19.705]
NPT = False

""" ################################################"""
print "#############################################################################"
print ""
"""El siguiente if es de advertencia por si no se pasó el numero de parámetros necesarios
    """
if len(sys.argv) < datos:
   print "\t \t Error al leer datos por consola"
   print "\t \t Las instrucciones para ejecutar son: python script.py posiciones.xyz
    velocidades.xyz"
   if len(sys.argv) == 1:
        print "\t \r Falta archivo de posiciones y velocidades"
        print ""
        print "#############################################################################"
   if len(sys.argv) == 2:
        print "\t \r Falta archivo de posiciones o velocidades"
        print ""
        print "#############################################################################"
   sys.exit(0)


Archivo1 = open(sys.argv[1],"r")
Archivo2 = open(sys.argv[2],"r")
Archivo3 = open(NombreArchivoSalida,"w")
AtomoID=[]
Coordenadas=[]


def LeeArchivo(Archivo,AtomoID,Coordenadas):
    i = 0
    Atomo = 0
    x=[]
    y=[]
    z=[]
    aid=[]
    for linea in Archivo:
         i=i+1
         if i==1:
            NumeroAtomos=int(linea)
            for k in range(NumeroAtomos):
                AtomoID.append("")
                Coordenadas.append([])
                for l in range(3):
                    Coordenadas[k].append(0.0)
         if i>2 :
              lineaAux = linea.split()
              aid=lineaAux[0:1]
              AtomoID[Atomo]=aid[0]
              x=map(float,lineaAux[1:2])
              y=map(float,lineaAux[2:3])
              z=map(float,lineaAux[3:])
              Coordenadas[Atomo][0]=x[0]
              Coordenadas[Atomo][1]=y[0]
              Coordenadas[Atomo][2]=z[0]
              Atomo = Atomo +1
    return NumeroAtomos


NumeroAtomos=LeeArchivo(Archivo1,AtomoID,Coordenadas)
Archivo3.write('<?xml version="1.0" ?>\n')
Archivo3.write('<State openmmVersion="7.1.1" time="0" type="State" version="1">\n')
Archivo3.write('\t<PeriodicBoxVectors>\n')
Archivo3.write('\t\t<A x="{0:3.6f}" y="0" z="0"/>\n'.format(PBC[0]/10))
Archivo3.write('\t\t<B x="0" y="{0:3.16f}" z="0"/>\n'.format(PBC[1]/10))
```

```
Archivo3.write('\t\t<C x="0" y="0" z="{0:3.6f}"/>\n'.format(PBC[2]/10))
Archivo3.write('\t<PeriodicBoxVectors>\n')

if NPT:
    Archivo3.write('\t<Parameters MonteCarloPressure="1.01325" MonteCarloTemperature="298.15"/>\n')
    print "Escogiste ensamble NPT, la presión es de 1.101325 y la Temperatura del barostato es 298.15 k"
    print "Si la P y T del barostato son diferente, modifica en la linea 88"
###Si el ensamble es NPT
else :
    Archivo3.write('\t<Parameters/>\n')
    print "El XML es para un ensamble NVE o NVT, si quieres un NPT modifica con True la linea 20"
#Para ensamble NVE y NVT

Archivo3.write('\t<Positions>\n')
for i in range(NumeroAtomos):
    Archivo3.write ('\t\t<Position x="{0:3.18f}" y="{1:3.18f}" z="{2:3.18f}"/>\n'.format(Coordenadas[i][0]/10,Coordenadas[i][1]/10,Coordenadas[i][2]/10))
Archivo3.write('\t</Positions>\n')
NumeroAtomos=LeeArchivo(Archivo2,AtomoID,Coordenadas)
Archivo3.write('\t<Velocities>\n')
for i in range(NumeroAtomos):
    Archivo3.write ('\t\t<Velocity x="{0:3.18f}" y="{1:3.18f}" z="{2:3.18f}"/>\n'.format(Coordenadas[i][0]*100,Coordenadas[i][1]*100,Coordenadas[i][2]*100))
Archivo3.write('\t</Velocities>\n')
Archivo3.write('</State>\n')

print "El archivo.xml es:",NombreArchivoSalida
print ""
print "################################################################################"
```

7.18 Script de Python para obtener el momento multipolar de los modelos iAMOEBA/AMOEBA en OpenMM

```
"""
autor: Anthoni Alcaraz Torres
Maestria en Ciencias - UAEM
Funciones para imprimir los valores de los momentos multipolares
calculados en openMM para: iAMOEBA y AMOEBA
"""

def imprimeMultipolos(filename,arrangesize,currentStep,timeStep):
    filename.write("Time:  %f ps  \t Step: %f\n\n"%(currentStep*timeStep,currentStep))
    for i in arrangesize:
        for j in i:
                filename.write( "{:+16.12f}\t".format(j),)
        filename.write("\n ")


#Momento multipolar del sistema
def imprimeMomentoMultipolar(filename,arrangesize,currentStep,timeStep):
    filename.write("Time:  %f ps  \t Step: %f\n\n"%(currentStep*timeStep,currentStep))
    contador=1
    for i in arrangesize:
                if contador == 1:
                    filename.write( "{:+16.12f}\t".format(i),)
                    filename.write("\n ")
                    contador = 2
                else :
                    filename.write( "{:+16.12f}\t".format(i),)
```

```
                    if contador == 4 or contador == 7 or contador == 10 or contador ==
    13:
                        filename.write("\n ")
                    contador = contador + 1
```

7.19 Script de Python para obtener fuerzas y velocidades de OpenMM

```
"""
autor: Anthoni Alcaraz Torres
Maestria en Ciencias - UAEM
clases para imprimir fuerzas y velocidades de la dinamica en openMM
"""
from simtk.openmm.app import *
from simtk.openmm import *
from simtk.unit import *


class ForceReporter(object):
    def __init__(self, file, reportInterval):
        self._out = open(file, 'w')
        self._reportInterval = reportInterval

    def __del__(self):
        self._out.close()

    def describeNextReport(self, simulation):
        steps = self._reportInterval - simulation.currentStep%self._reportInterval
        return (steps, False, False, True,False)

    def report(self, simulation, state):
        forces = state.getForces().value_in_unit(kilojoules/mole/nanometer)
        print >>self._out, "Time:",state.getTime(),"Step:",simulation.currentStep
  print >>self._out,""
        for f in forces:
            print >>self._out,"{:+20.8f}\t{:+20.8f}\t{:+20.8f}".format(f[0], f[1], f[2])

class PosReporter(object):
    def __init__(self, file, reportInterval):
        self._out = open(file, 'w')
        self._reportInterval = reportInterval

    def __del__(self):
        self._out.close()

    def describeNextReport(self, simulation):
        steps = self._reportInterval - simulation.currentStep%self._reportInterval
        return (steps, False, True, False, False)

    def report(self, simulation, state):
        pos = state.getVelocities().value_in_unit(nanometers/picoseconds)
        print >>self._out, "Time:",state.getTime(),"Step:",simulation.currentStep
  print >>self._out,""
        for v in pos:
            print >>self._out,"{:+20.8f}\t{:+20.8f}\t{:+20.8f}".format( v[0],v[1], v[2])

class xyzReporter(object):
    def __init__(self, file, reportInterval,particles):
        self._out = open(file, 'w')
        self._reportInterval = reportInterval

    def __del__(self):
        self._out.close()

    def describeNextReport(self, simulation):
```

```
        steps = self._reportInterval - simulation.currentStep%self._reportInterval
        return (steps, True, False, False,False)

    def report(self, simulation, state):
        forces = state.getPositions().value_in_unit(angstroms)
        print >>self._out, "Time:",state.getTime(),"Step:",simulation.currentStep
  print >>self._out,""
        for f in forces:
            print >>self._out,"{:+20.8f}\t{:+20.8f}\t{:+20.8f}".format(f[0], f[1], f[2])
```

7.20 Script de Python para calcular regresión lineal de una trayectoria

```
import math
import sys
import os
import string

"""
Autor: Anthoni Alcaraz Torres
Maestria en Ciencias - UAEM

Este Script ejecuta el calculo del momento dipolar del sistema de una subrutina en *.cpp
funciona para tip4pfb, porque para iamoeba y amoeba se obtiene el momento dipolar del
sistema de openMM.

Recibe como parametros:
1. trayectoria completa en formato xyz
2. El numero de puntos de la trayectoria para crear un bloque, bloque n = 18, 20, 10, etc
"""
#Modelo de agua para analisis de Momento Dipolar del Sistema
modelo="tip4pfb"
#Lee los puntos y archivo de consola
trayectoria = sys.argv[1]
Archivo1 = open(sys.argv[1],"r")
PuntosDivision = int(sys.argv[2])
#Crea la carpeta con el Analisis
os.system("mkdir Segmento{0}Puntos".format(PuntosDivision))
PuntosConfiguracion = 15362
#El numero corresponde a el numero de lineas por configuracion en el xyz
lineas = PuntosConfiguracion*PuntosDivision
#Compilado de *.cpp para calcular momento dipolar del sistema
ejecuta="~/Dropbox/CODIGOMAESTRIA/COMPILADOS/ConDie116"
#Divide la trayectoria en n archivos
os.system("split -l {0} {1} -d tr".format(lineas,trayectoria))
#Funcion que cuenta las lineas del archivo que contiene la trayectoria
def CuentaDatos(Archivo):
    i = 0
    for linea in Archivo:
         i=i+1

    return i

#Cuenta puntos de trayectoria
Np =  CuentaDatos(Archivo1)
Archivo1.close()
PasosTotales = Np/PuntosConfiguracion
PasosTotales = 25000
puntos = int(math.ceil(PasosTotales/float(sys.argv[2])))

"""
Al dividir la trayectoria asigna tr00 hasta tr89, pero al cambiar a 90 agrega dos ceros y
el archivo se convierte en tr9000, lo mismo sucede al pasar de tr9889, lo escribe como
tr900000. Por esta razon creo los siguientes tres ciclos for para realizar el calculo.
"""
```

```
#primer ciclo para los archivos obtenidos con split 0-89
for j in range(90):
     a=("mv tr{0:02d} trayectoria{0:02d}.xyz".format(j))
     os.system("{0}".format(a))
     b=("{0} trayectoria{1:02d}.xyz {2} y 59 0 > MD{1:02d}.dat".format(ejecuta,j,modelo))
     os.system("{0}".format(b))
     os.system("rm trayectoria{0:02d}.xyz".format(j))


#Segundo ciclo para los archivos obtenidos con split 9000-9899
if puntos > 90:
     contador=9000
     for i in range(900):
          a = ("mv tr{0:02d} trayectoria{0:02d}.xyz".format(contador))
    os.system("{0}".format(a))
    b = ("{0} trayectoria{1:02d}.xyz {2} y 59 0 > MD{1:02d}.dat".format(ejecuta,contador,
    modelo))
    os.system("{0}".format(b))
    os.system("rm trayectoria{0:02d}.xyz".format(contador))
    contador = contador +1

#tercer ciclo para los archivos obtenidos con split 990000-994009
     if puntos > 990:
          contado = 990000
          for i in range(puntos-990):
               a = ("mv tr{0:02d} trayectoria{0:02d}.xyz".format(contado))
          os.system("{0}".format(a))
          b = ("{0} trayectoria{1:02d}.xyz {2} y 59 0 > MD{1:02d}.dat".format(ejecuta,
    contado,modelo))
          os.system("{0}".format(b))
          os.system("rm trayectoria{0:02d}.xyz".format(contado))
          contado = contado +1

     else:
          nada=0

else:
    nada=0

#Calcula epsilon y DevStd por bloques#
os.system("python ~/Dropbox/ScriptsPython/AnalisisDielectrica/MoDiptr.py {0} > conteo.txt
    ".format(puntos))
#Crea un archino con los numeros de los n bloques
NombreArchivo = "num{0}.txt".format(PuntosDivision)
ArchivoNumero = open(NombreArchivo,"w")

for i in range(puntos):
     ArchivoNumero.write("{0}\n".format((i+1)*PuntosDivision))

ArchivoNumero.close()
#Junta los archivos de epsilon y DevStd con su numero de bloque
os.system("paste num{0}.txt conteo.txt > epsilon{1}puntos.txt".format(PuntosDivision,
    puntos))
os.system("mv *.dat conteo.txt num{0}.txt epsilon{1}puntos.txt Segmento{0}Puntos/".format
    (PuntosDivision,puntos))
"""
AQUI DEBE DIVIDIRSE ESTE SCRIPT Y LLAMARLO COMO EN LA LINEA 91 Y GUARDARLO EN ESA RUTA
"""
# coding=utf-8
import os
import sys
import string

"""
Autor: Anthoni Alcaraz Torres
Maestria en Ciencias - UAEM
```

```
114
115 Recibe por consola el numero de bloques: puntos entre tamaño del bloque
116 """
117 Instruccion = "python ~/Dropbox/ScriptsPython/EpsDevStd.py"
118 puntos = int(sys.argv[1])
119
120 for i in range(90):
121      os.system('{0} MD{1:02d}.dat'.format(Instruccion,i))
122
123 if puntos > 90:
124      contador=9000
125      for i in range(puntos-90):
126           os.system('{0} MD{1:02d}.dat'.format(Instruccion,contador))
127           contador=contador+1
128
129      if puntos > 990:
130           contado = 990000
131           for i in range(puntos-990):
132                os.system('{0} MD{1:02d}.dat'.format(Instruccion,contado))
133                contado = contado+1
134
135      else:
136           nada=0
137
138 else:
139     nada=0
```

7.21 Script de Python para calcular regresión lineal de un conjunto de datos

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 """
4 autor: Anthoni Alcaraz Torres
5 UAEM - Maestria en Ciencias MCCC
6 las ecuaciones fueron obtenidas de https://www.uv.es/zuniga/08
      _Ajuste_de_una_recta_por_minimos_cuadrados.pdf
7 Este programa calcula la regresion lineal de un conjunto de datos con formato x y
8 ** Para correr el script necesitas dar el nombre del archivo a analizar
9      python script.py input.txt salida.dat
10 Devuelve la pendiente, la interseccion, la r, la incertidumbre de la pendiente y la
      incertidumbre de la interseccion
11 Las dos lineas iniciales  son para poder imprimir acentos y la libreria matematicas para
      calcular sqrt """
12
13 import math
14 import sys
15 datos=3
16 if len(sys.argv) < datos:
17    print "\t \t Error al leer datos por consola"
18    print "\t \t Las instrucciones para ejecutar son: python script.py datos"
19    if len(sys.argv) == 1:
20         print "\t \r Falta archivo de Datos"
21    sys.exit(0)
22
23 Archivo1 = open(sys.argv[1],"r")
24 Archivo2 = open(sys.argv[1],"r")
25 Archivo3 = open(sys.argv[2],"w")
26 x=[]; y=[]; xmedia=ymedia=r=0
27 Incertidumbreb=Incertidumbrea=0
28 DevStd=chi2=sumaX=sumaY=sumaXY=sumaX2=sumaY2=sumaN=sumaD=0
29
30 def CuentaDatos(Archivo):
31     i = 0
32     for linea in Archivo:
```

```
33          i=i+1
34          NumeroAtomos=i
35
36      return NumeroAtomos
37
38  def AsignaValores(Archivo,val,x,y):
39      x=[i for i in range(val)];xa=[]
40      y=[i for i in range(val)];ya=[]
41      xy=[val]
42      i=0
43      while i<val:
44          for linea in Archivo:
45              lineaAux = linea.split()
46              xa=map(float,lineaAux[0:1])
47              ya=map(float,lineaAux[1:2])
48              x[i]=xa[0]
49              y[i]=ya[0]
50              i=i+1
51
52
53      return x,y
54
55  #Referencie el mismo archivo a dos variables, uno es para contar y otro para asignar
56  n=CuentaDatos(Archivo1)
57  x,y=AsignaValores(Archivo2,n,x,y)
58
59  #Calculo de las sumas
60  for i in range(n):
61       sumaX = sumaX + x[i]
62       sumaY = sumaY + y[i]
63       sumaXY = sumaXY + x[i]*y[i]
64       sumaX2 = sumaX2 + x[i]*x[i]
65       sumaY2 = sumaY2 + y[i]*y[i]
66
67
68  xmedia = sumaX/n
69  ymedia = sumaY/n
70
71  #Esta formula tambien es valida para calcular el valor de b
72  for i in range(n):
73  #       sumaN = sumaN + (x[i]-xmedia)*(y[i]-ymedia)
74  #       sumaD = sumaD + (x[i]-xmedia)**2
75          DevStd = DevStd + (y[i]-ymedia)**2
76  #b = sumaN/sumaD
77
78  #Calculo de la pendiente(b) y la interseccion (a)"""
79  b = (sumaXY-(sumaX*sumaY)/n)/(sumaX2-(sumaX**2)/n)
80  a = sumaY/n -b*sumaX/n
81
82  #Calculo de la incertidumbre de los valores a y b
83  for i in range(n):
84       chi2 = chi2 + (a + b*x[i]-y[i])**2
85
86  Incertidumbreb = math.sqrt((n*chi2)/((n*sumaX2-sumaX**2)*(n-2)))
87  Incertidumbrea = math.sqrt((sumaX2*chi2)/((n*sumaX2-sumaX**2)*(n-2)))
88  r = (n*sumaXY-sumaX*sumaY)/((math.sqrt(n*sumaX2-sumaX**2))*(math.sqrt(n*sumaY2-sumaY**2))
        )
89  Archivo3.write( "#################################################################\n\n")
90  Archivo3.write("El archivo de Datos fue {0}\n\n".format(sys.argv[1]))
91  Archivo3.write ("\tIntersección( b )\tPendiente( m )\n")
92  Archivo3.write ("y  =  {0:5.16f}  {1:+5.16f}*x\n".format(a,b))
93  Archivo3.write( "\t \t *** Incertidumbres ***\n")
94  Archivo3.write( "E(m): {0:3.8f} E(b): {1:3.8f}\n".format(Incertidumbreb,Incertidumbrea))
95  Archivo3.write( "\t *** Coefiente de correlación lineal ***\n")
96  Archivo3.write( " r: {0:3.8f}\n".format(r))
97  Archivo3.write( "\t *** Promedio \t Desviación Estándar ****\n")
```

```
98 Archivo3.write( "X: {0:3.8f} \t \t DevStd: {1:3.8f}\n".format(ymedia,math.sqrt(DevStd/n))
       )
99 Archivo3.write( "\n#################################################################\n")
100 print("{0:+4.8f}  {1:+4.8e}  {2:+4.8e}  {3:+4.8e}  {4:+4.8f}  {5:+4.8f}  {6:+4.8f}".
       format(a,b,Incertidumbrea,Incertidumbreb,r,ymedia,math.sqrt(DevStd/n)))
```

7.22 Script de Python para calcular la constante dieléctrica y el factor de Kirwood

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 """
4 autor: Anthoni Alcaraz Torres
5 UAEM - Maestria en Ciencias MCCC
6 Este programa calcula la constante dielectrica relativa tomando n valores del momento
      dipolar del sistema
7 que se analizo previamente, el formato es x y donde y es el momento multipolar del
      sistema
8 ** Para correr el script necesitas dar el nombre del archivo a analizar
9      python script.py input.txt
10 Unicamente imprime el valor de la constante dielectrica para ese conjunto de datos, esto
      es, tomar n puntos de la
11 trayectoria (por ejemplo, 20 ps, imprimiento cada ps entonces son 20 datos), calcular su
      momento dipolar del
12 sistema por punto e imprimirlo en un archivo (input.txt), ese archivo se alimenta a este
      programa y calcula la
13 constante dielectrica para esos puntos.
14 Si se quiere guardar la salida en un archivo de salida, entonces, cambiar la variable
      datos a 3, quitar el #
15 donde se declara Archivo3 y donde diga printf cambiarlo por Archivo3.write.
16 """
17 import math
18 import sys
19 datos=2
20 if len(sys.argv) < datos:
21    print "\t \t Error al leer datos por consola"
22    print "\t \t Las instrucciones para ejecutar son: python script.py datos"
23    if len(sys.argv) == 1:
24         print "\t \r Falta archivo de Datos"
25    sys.exit(0)
26
27 Archivo1 = open(sys.argv[1],"r")
28 Archivo2 = open(sys.argv[1],"r")
29 #Archivo3 = open(sys.argv[2],"w")
30 #variables para sumar y obtener promedios
31 x=[]; y=[]; xmedia=ymedia=r=x2media=y2media=factor=0
32 Incertidumbreb=Incertidumbrea=0
33 DevStd=chi2=sumaX=sumaY=sumaXY=sumaX2=sumaY2=sumaN=sumaD=0
34
35 #factores para el calculo de la constante dielectrica
36 e0 = 8.8541878176E-12 #Epsilon cero
37 volumen = 19.705*19.705*295.584 # Volumen en angstroms cubicos
38 volumen = volumen*(1E-10)**3 #conversion a metros cubicos
39 Pi = math.pi
40 Temperatura = 298.15 #kelvin
41 KBolt = 1.380648813E-23 #J/K
42
43 factor = 1/(3*volumen*Temperatura*KBolt*e0)
44
45 def CuentaDatos(Archivo):
46     i = 0
47     for linea in Archivo:
48          i=i+1
```

```
            NumeroAtomos=i

    return NumeroAtomos

def AsignaValores(Archivo,val,x,y):
    x=[i for i in range(val)];xa=[]
    y=[i for i in range(val)];ya=[]
    xy=[val]
    i=0
    while i<val:
        for linea in Archivo:
            lineaAux = linea.split()
            xa=map(float,lineaAux[0:1])
            ya=map(float,lineaAux[1:2])
            x[i]=xa[0]
            y[i]=ya[0]
            i=i+1


    return x,y

#Referencie el mismo archivo a dos variables, uno es para contar y otro para asignar
n=CuentaDatos(Archivo1)
x,y=AsignaValores(Archivo2,n,x,y)

#Calculo de las sumas
for i in range(n):
     sumaX = sumaX + x[i]
     sumaY = sumaY + y[i]
     sumaXY = sumaXY + x[i]*y[i]
     sumaX2 = sumaX2 + x[i]*x[i]
     sumaY2 = sumaY2 + y[i]*y[i]


xmedia = sumaX/n
ymedia = sumaY/n
x2media = sumaX2/n
y2media = sumaY2/n

#Calculo de la desviacion estandar de los momentos dipolares
for i in range(n):
      DevStd = DevStd + (y[i]-ymedia)**2

#calculo de la fluctuacion del momento dipolar en Debye
eps = y2media
#conversion a C2m2
eps = eps*(3.34E-30)**2
#Multiplicacion por factor y sumar 1
eps = 1 + eps*factor
print("{0:+4.8f} \t {1:+4.8f}".format(eps,math.sqrt(DevStd/n)))
#print("{:+4.8f} \t {:+4.8f} \t {:+4.8f}".format(y2media,ymedia*ymedia,eps))
```

7.23 Subrutina para clasificar los átomos a lo largo del eje Y de la celda de simulación

```
#include "librerias.h"

//la unidad de medida de la caja es en angstroms
#define SideX 19.7056
#define SideY 295.5846
#define SideZ 19.7056

int main(int argc,char** argv){
```

```
9
10    //Archivo xyz
11    strcpy(Archivo,argv[1]);
12
13    //Calcula lineas de archivo
14    LinesFile=LongitudArchivo(Archivo);
15
16    //Calcula el numero de configuraciones
17    fileParticlesConfiguration(Archivo);
18
19    //Detiene el programa si pasa limites de arreglos
20    if(Alerts()!=2){return 0;}
21
22    //Coordenada de analisis
23    CoordinateSelection();
24
25    //volumen por celda y volumen de la celda
26    ComputeDelta(valor);
27
28    //Analiza los n puntos de la trayectoria
29    file_xyz=fopen(Archivo,"r");
30
31        for(Replicas=0;Replicas<1;Replicas++){
32          //Asigna coordenadas a los arreglos
33          ReadFileValues();
34        //Por secciones y la primer seccion va de 0 a 5 en y
35            DivideMolecules();
36
37        }
38
39    fclose(file_xyz);
40
41     return 0;
42
43 }
```

7.24 Subrutina para calcular las distancias temporales de los átomos cercanos a la superficie

```
1  #include "librerias.h"
2
3  //la unidad de medida de la caja es en angstroms
4  #define SideX 19.7056
5  #define SideY 295.5846
6  #define SideZ 19.7056
7
8  int main(int argc,char** argv){
9
10   //Archivo xyz
11   strcpy(Archivo,argv[1]);
12
13   //Calcula lineas de archivo
14   LinesFile=LongitudArchivo(Archivo);
15
16   //Calcula el numero de configuraciones
17   fileParticlesConfiguration(Archivo);
18
19   //Detiene el programa si pasa limites de arreglos
20   if(Alerts()!=2){return 0;}
21
22   //Coordenada de analisis
23   CoordinateSelection();
24
```

```
  //volumen por celda y volumen de la celda
  ComputeDelta(valor);

  //Analiza los n puntos de la trayectoria
  file_xyz=fopen(Archivo,"r");

       for(Replicas=0;Replicas<Configuration;Replicas++){
         //Asigna coordenadas a los arreglos
         ReadFileValues();
           Distances();
       }

  fclose(file_xyz);

   return 0;

}
```

7.25 Subrutina para calcular funciones de distribución radial

```
/***********************************************
 * calculate radial distribution function g(r)
 * of ensemble configurations in "configuration.xyz"
 * output results in "gr.dat"
 * Calculate rdf only of 2 atoms and you can give its symbol
 * Anthoni Alcaraz Torres CIQ, UAEM 2015
 ***********************************************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
/**
 *Datos que se modifican
 */
#define MAXConfig 150000
#define MAXEqui 30000
#define MAXNumberOfParticles 50000
#define MAXNumberOfBins 10000
#define pi 3.14159265
#define SideY 295.5846
#define SideZ 19.7056
#define SideX 19.7056             //la unidad de medida de la caja es en angstroms

#define SQR(x) ((x)*(x))
#define CUB(x) ((x)*(x)*(x))
//Apuntador a la ruta donde se encuentra el archivo con la trayectoria de la DM con las coordenas en xyz
char dirArchivoDinamica[100];
char *dirArchivogrAB="gAB2.dat";
int getLength(char*);
int main(int argc,char** argv){
  int NumberOfConfig,NumberOfEqui,NumberOfParticles;
  int SampleNumber;
  int snapshot,line;
  double rho,vol;
  double dr;
  double g[MAXNumberOfBins], gAB[MAXNumberOfBins];
  int NumberOfBins, NA, NB;
  int i,j, k,l,NumLineas;
  double dx,dy,dz;
  double rij;
  char AtomID[MAXNumberOfParticles][4], AtomA[4], AtomB[4], lineaUsada[100];
  FILE *fpxyz;
  FILE *fpoutput;
```

```
  FILE *fpoutput2;
  double x[MAXNumberOfParticles],y[MAXNumberOfParticles],z[MAXNumberOfParticles];
  vol = SideX*SideY*SideZ;
  strcpy(dirArchivoDinamica,argv[1]);
  NumberOfEqui = atoi(argv[5]);
  dr = atof(argv[4]);
  NA =atoi(argv[6]);
  NB = atoi(argv[7]);
  strcpy(AtomA,argv[2]);
  strcpy(AtomB,argv[3]);
  NumberOfBins = (int)(15.0/dr);
  int contadorAtomA[NA+1], contadorAtomB[NB+1], asignaA=1, asignaB=1;
  SampleNumber = 0;
  for(i=0;i<NumberOfBins;i++){
   g[i] = 0;
   gAB[i] = 0;
  }
  SampleNumber = 0;
NumLineas=getLength(dirArchivoDinamica);
fpxyz = fopen(dirArchivoDinamica,"r");
fgets(lineaUsada,100,fpxyz);sscanf(lineaUsada,"%d",&NumberOfParticles);
fclose(fpxyz);
rho = NumberOfParticles/vol;
NumberOfConfig=NumLineas/(NumberOfParticles+2);

  for(snapshot=0;snapshot<NumberOfConfig;snapshot++){
  fgets(lineaUsada,100,fpxyz);fgets(lineaUsada,100,fpxyz);
      for(line=0;line<NumberOfParticles;line++){
  fgets(lineaUsada,100,fpxyz);
      sscanf(lineaUsada,"%s %lf %lf %lf",AtomID[line],&x[line],&y[line],&z[line]);
  //printf("%d \t%s %10.6lf %10.6lf %10.6lf\n",line,AtomID[line],x[line],y[line],z[line])
    ;

//Busca en el arreglo donde se encuentra el atomo A para tener coordenadas
if(snapshot == 0){
for(i=0;i<NumberOfParticles;i++){
 if(strstr(AtomID[i],AtomA)!=0){
   if(asignaA<=NA){
  contadorAtomA[asignaA]=i;
  //printf("A %s \t %d \t %d\n",AtomID[i],i, contadorAtomA[asignaA]);
  asignaA++;
    }
  }
 if(strstr(AtomID[i],AtomB)!=0){
    if(asignaB<=NB){
      contadorAtomB[asignaB]=i;
     //printf("B %s \t %d \t %d\n",AtomID[i],i, contadorAtomB[asignaB]);
      asignaB++;
    }
  }
}
}

   if((snapshot+1) > NumberOfEqui){
    SampleNumber++;
      for(k=1;k<NA;k++){
        for(l=k+1;l<=NB;l++){
       i=contadorAtomA[k];
       j=contadorAtomB[l];
       dx = x[i] - x[j];
       dx = dx - SideX*round(dx/SideX);

       dy = y[i] - y[j];
       dy = dy - SideY*round(dy/SideY);

       dz = z[i] - z[j];
```

```
        dz = dz - SideZ*round(dz/SideZ);

        rij = sqrt(SQR(dx)+SQR(dy)+SQR(dz));

        gAB[(int)(rij/dr)] += 2.0;
      }
     }
   }
}


fpoutput2 = fopen(dirArchivogrAB,"w");
  for(i=0;i<NumberOfBins;i++){
   gAB[i] /= SampleNumber;
   gAB[i] /= 4.0*M_PI/3.0*(CUB(i+1)-CUB(i))*CUB(dr);
    fprintf(fpoutput2,"%f\t %f\n",(i+0.5)*dr,gAB[i]*vol/(NA*NB));

  fclose(fpoutput2);
 return 0;
}
int getLength(char* dir){
        FILE *f;
        //obtiene la cantidad de lineas del archivo
        int l=0;
        char linea[100];
        f=fopen(dir, 'r');
        if(f!=NULL){
                while(!feof(f)){
                        fgets(linea,100,f);
                        l++;
                }
                fclose(f);
        }
        return l-1;
}
```

7.26 Subrutina para calcular mapas estructurales

```
/*
 * Este programa divide una cara de la celda en una malla con cuadriculas
 * del mismo tamaño. Por ejemplo la cara x,z cada angstrom.
 * */
#include "librerias.h"

//la unidad de medida de la caja es en angstroms
#define SideX 19.7056
#define SideY 295.5846
#define SideZ 19.7056

int main(int argc,char** argv){
  //Archivo xyz
  strcpy(Archivo,argv[1]);

  //Coordenada que no se analiza, y: analiza x y z
  strcpy(Coordinate,argv[2]);

  //Divisiones en la malla 19x19
  Bins = atof(argv[3]);

  //Atomo a considerar, Oxigeno
  strcpy(TypeAtomsConsider,argv[4]);

  //distancia maxima a la superficie
  limite = atof(argv[5]);
```

```
27
28    //seleccion de la coordenada de analisis
29    CoordinateSelection();
30
31    //Calculo de tamaño de cuadricula
32    ComputeDeltaGrid();
33
34    //Calcula las lineas del archivo
35    LinesFile = LongitudArchivo(Archivo);
36
37    //Calcula el numero de configuraciones
38    fileParticlesConfiguration(Archivo);
39
40    //Verifica que el tamano de los arreglos
41    if(Alerts()!=2){return 0;}
42
43    //Analiza la trayectoria
44          file_xyz=fopen(Archivo,"r");
45
46                  for(Replicas=0;Replicas<Configuration;Replicas++){
47                //Asigna coordenadas a los arreglos
48                      ReadFileValues();
49                      //Asigna los atomos
50                      MapGrid();
51                      }
52
53          //Imprime Archivo para gnuplot
54          FileGnuplotGrid();
55          system("gnuplot input.gpl");
56          //system("gnuplot input2.gpl");
57          fclose(file_xyz);
58
59          return 0;
60
61  }
```

7.27 Subrutina para obtener estructuras de agua hasta 5 Å de la superficie

```
1  #include "librerias.h"
2
3  int main(int argc,char** argv){
4      //Recibe la replica en xyz
5      strcpy(Archivo,argv[1]);
6
7      //lee lineas de archivo
8      LinesFile = LongitudArchivo(Archivo);
9
10     //calcula el numero de configuraciones
11     fileParticlesConfiguration(Archivo);
12
13     freopen("MoleculasBin1y2.xyz","w",stdout);
14     file_xyz=fopen(Archivo,"r");
15     for(Replicas=0;Replicas<Configuration;Replicas++){
16       ReadFileValues();
17       ImprimeBines();
18        }
19     fclose(file_xyz);
20     fclose(stdout);
21
22   return 0;
23
24 }
```