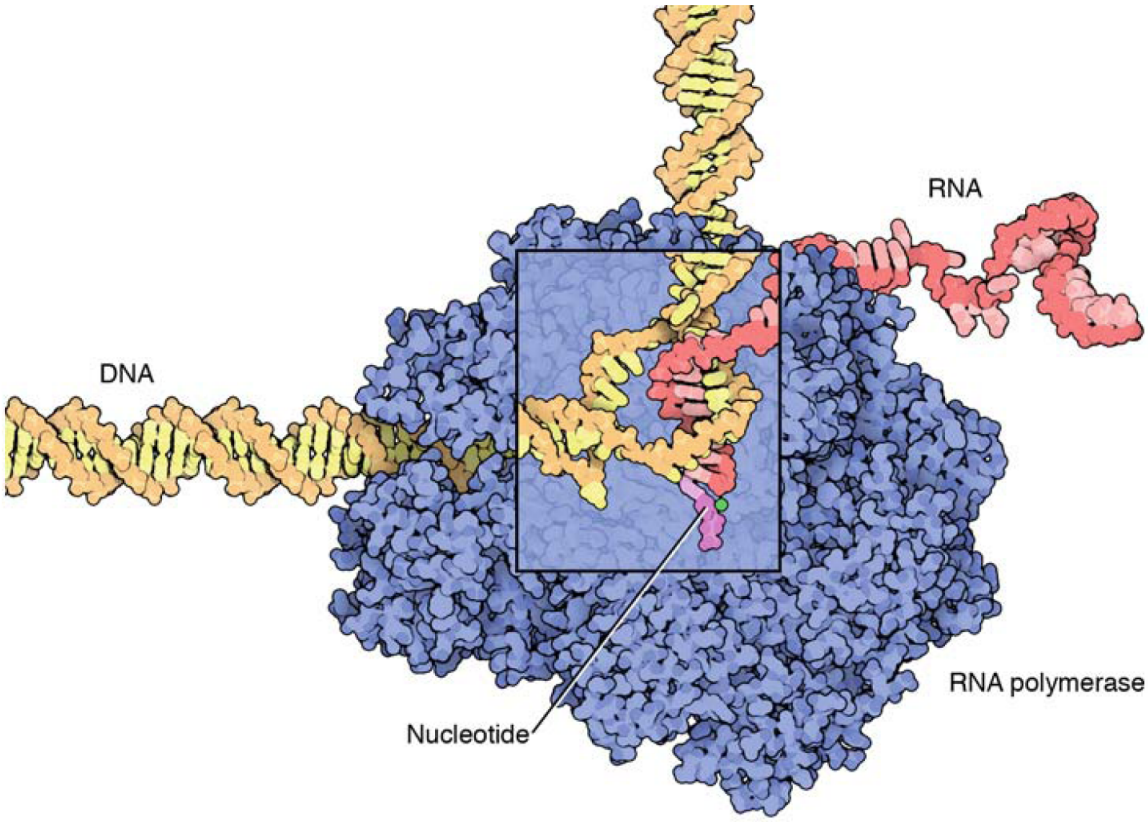
Trabajo de fin de grado

Modelización de motores moleculares: modelo de helicasa

Guillermo Díez Señoráns



Dirigido por Fernando Falo Forniés y Rafael Tapia Rojo.

Departamento de Física de la Materia Condensada

Universidad de Zaragoza

Índice

1. Introducción y objetivos

En un espíritu generalista, un *motor molecular* (o *celular*) se puede definir como un agregado de moléculas discretas capaces de transformar energía química en trabajo mecánico (*i.e* desplazarse en contra de una fuerza externa) *en la escala molecular.* En la práctica, estos dispositivos sólo han resultado eficaces -y masivamente empleados- tras el largo proceso de optimización al que la evolución ha sometido la vida en La Tierra. En este contexto, los motores moleculares son proteínas que presentan acción enzimática sobre la reacción de hidrólisis de alguna biomolécula, que a su vez cumple la función de fuente de energía para el motor; de forma muy mayoritaria en los organismos vivos, esta molécula es *adenosín trifosfato* (en adelante ATP), un *nucleótido* altamente energético que se descompone mediante la reacción $ATP \rightarrow ADP + P_i$ en otro nucleótido (ADP) poco energético y un grupo fosfato libre. En la naturaleza, estos motores llevan a cabo funciones de *transporte activo* (procesos en contra del gradiente de energía libre) dentro de la célula, y pueden encontrarse especializados en tareas dispares como *translación* (transporte de vesículas y orgánulos), *rotación* (movimiento de cilios y flagelos para el desplazamiento bacteriano), *polimerización* (crecimiento de estructuras celulares) y *traslocación* (manipulación del ADN).

Además del interés biológico indudable en comprender la fenomenología de los motores moleculares, es cierto que también existen razones puramente físicas para su estudio [1, 2]: el tiempo de relajación característico de las inhomogeneidades térmicas en la escala molecular ($\sim 10^{-13}s$) es muy inferior a la velocidad típica de un motor celular ($\sim 10^{-3}s$), lo cual implica que un motor de estas características opera en condiciones isotermas, un escenario radicalmente diferente del caso macroscópico, donde el trabajo procede de un sistema que se mueve cíclicamente entre focos a distinta temperatura. A pesar de esto, se trata de un sistema irreversible y muy alejado del equilibrio químico (de hecho, la energía libre neta liberada por la hidrólisis/síntesis de ATP sería nula en equilibrio, llevando a la muerte celular: el desequilibrio es cuidadosamente controlado mediante la inhibición de la acción enzimática frente al exceso de adenosín trifosfato). Como sistema dinámico, los motores moleculares presentan interacciones colectivas a diferentes escalas: por un lado, el funcionamiento de un único motor implica la coordinación de distintas subunidades; a menudo agrupaciones de estas máquinas interaccionan en sistemas dinámicos *cooperativos* o *competitivos*, y también el conjunto de motores actuando en una célula es un sistema ordenado fuera del equilibrio termodinámico (*autoorganizado*). Por último, en la escala molecular el medio celular resulta enormemente disipativo, lo que conduce a una dinámica sobreamortiguada: un proceso sólo será efectivo si no tiene lugar a lo largo de un ciclo simétrico, esto es, debe existir algún mecanismo de ruptura de la simetría que impida al sistema regresar a su estado inicial mediante la reversión de un ciclo.

Teniendo en cuenta estos aspectos de la física en la escala microscópica, se llega de una forma natural a considerar el *movimiento browniano rectificado* mediante *potenciales ratchet* en la acción de los motores celulares [3].

Como se ha visto, para el estudio de los motores moleculares debe renunciarse a algunas potentes técnicas analíticas (termodinámica reversible, física estadística del equilibrio, linealidad), lo cual ha propiciado la aproximación numérica y la simulación computacional como herramientas indispensables en el análisis de estos sistemas; resultado de la búsqueda de un compromiso entre los complejos pero rápidos modelos a gran escala, y los simples pero prohibitivamente lentos modelos de dinámica molecular surgen los *modelos en la mesoescala,* en los que se busca identificar y relacionar los grados de libertad más relevantes de un sistema físico mediante interacciones promedio de fuerzas que varían en escalas de tiempo y espacio microscópicas. En el modelado mesoscópico de motores celulares se emplean escalas espacio-temporales características:

$$l_{meso} \sim \text{nanómetro} \qquad t_{meso} \sim \text{picosegundo}$$

frente a los órdenes típicos de la escala molecular:

$$l_{micro} \sim \text{angstrom (tamaño atómico)}$$

$$t_{micro} \sim \sqrt{\frac{m_H}{3k_bT}} \cdot l_{micro} \sim 10\,\text{femtosegundos (298 [K])}$$

El objetivo de este trabajo será combinar un modelo clásico de dinámica del ADN (modelo de Peyrard-Bishop-Dauxois, en adelante *PBD*) con un modelo de interacción que describa la acción como motor molecular de algunas proteínas enzimáticas relacionadas con la manipulación del material genético (transcripción, replicación, traslocación y apertura de la doble hélice de ADN): *helicasas* y *RNA-polimerasas [4, 5].* Sobre este caso concreto se propondrán dos alternativas de modelos en la mesoescala, cada una de las cuales representativa de distintos mecanismos discutidos actualmente como acción de los motores celulares, a saber [6]:

- *Power stroke:* La hidrólisis de ATP dispara un cambio conformacional en la proteína que arroja el motor impulsivamente a una nueva posición.
- *Térmico:* La hidrólisis de ATP libera la proteína de su anclaje, que puede difundir hasta una nueva posición y fijarse en ella.

El grueso de la información experimental sobre la dinámica de estos sistemas ha procedido, desde finales del siglo XX, del empleo de microscopios de fuerza atómica y pinzas ópticas para llevar a cabo experimentos de *molécula aislada* sobre motores celulares altamente procesivos, *i.e* cuyo movimiento se restringe a un filamento, típicamente ADN o un microtúbulo; en estos sistemas experimentales de alta resolución (actualmente en la escala del ángstrom), el motor se adhiere químicamente a algún material susceptible de ser manipulado por una trampa óptica (sílice o poliestireno), mientras se mide su posición sobre el substrato empleando medios ópticos. La fuerza de carga que experimenta el motor es proporcional a la distancia que lo separa de la pinza, de manera esta puede ser regulada mediante algún mecanismo de realimentación, garantizando así que la carga sobre el motor permanezca constante mientras se registra su movimiento a lo largo del substrato [4, 7]. Mediante esta técnica experimental es habitual medir la velocidad del motor en función de la concentración de 'combustible' disuelto en el medio (con frecuencia ATP), así como la fuerza de carga que son capaces de soportar antes de detenerse, llamada *fuerza de detención («stall force»).*

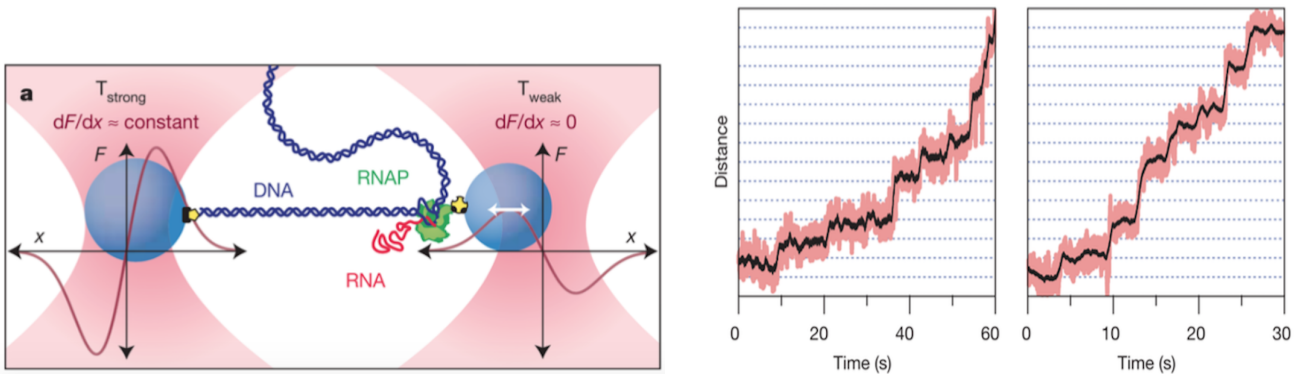


Figura 1: Pinza óptica ultraestable para un experimento de *molécula única* sobre la helicasa (izda) y medición de la posición del motor en dicho experimento (dcha)[4].

2. Modelos *ratchet*

En la escala celular la dinámica está gobernada por las fluctuaciones térmicas, y el movimiento browniano domina cinemáticamente; en este medio no puede existir una máquina térmica capaz de producir trabajo mediante transferencia de calor, tal y como ocurre en los motores macroscópicos: el trabajo máximo extraíble de un sistema tiene lugar cuando este se transforma reversiblemente, *i.e* cuando la entropía «del universo» permanece constante. Dado que la célula en las escalas de tiempo y espacio discutidas es un sistema aislado, el trabajo máximo se extraerá cuando este se transforme de manera reversible, pero al estar en equilibrio térmico su entropía *ya* es máxima, luego sólo puede transformarse isoentrópicamente en sí mismo. Pero entonces no puede haberse extraído trabajo de él (conservación de la energía), de forma que la exergía de este sistema es *nula*: es imposible *en principio* obtener trabajo de un motor molecular *en equilibrio*. Una famosa realización práctica de esta conclusión es *La máquina de Feynman*, un dispositivo teórico consistente en una rueda dentada cuyo movimiento se encuentra limitado por un trinquete (*ratchet*) que sólo le permite rotar en un sentido, unida solidariamente a un juego de palas que recoge los impactos del baño térmico en que se encuentra sumergido el conjunto. A pesar de que el sistema esté en equilibrio (todo el baño está a la misma temperatura), parece claro que el ratchet «seleccionará» sólo los impactos térmicos que permiten mover la rueda en una dirección, extrayendo así trabajo mecánico. En su *Lectures of physics,* Feynman desarrolla cuidadosamente los cálculos del funcionamiento de la máquina, concluyendo que en condiciones isotermas el dispositivo *ratchet* oscilará térmicamente de tal forma que no servirá para impedir el retroceso de la rueda dentada, llevando al fracaso del motor como máquina de movimiento perpetuo y siendo equivalente en su lugar a un motor ideal de Carnot [8] . Respecto de este último resultado, hay que mencionar un artículo posterior de Parrondo y Español con fecha de 1996 donde se corrigen algunos errores de cálculo presentes en el texto original [de Feynman], demostrando que en este dispositivo se dan algunos procesos irreversibles, lo que elimina la posibilidad de un motor máximamente eficaz [9].

La manera general de introducir esta clase de sistemas en la dinámica es sometiendo a las partículas brownianas a *potenciales ratchet* externos. Un potencial ratchet es una función potencial *periódica* y *asimétrica*, que recibe su nombre por la similitud con el ejemplo de Richard Feynman.

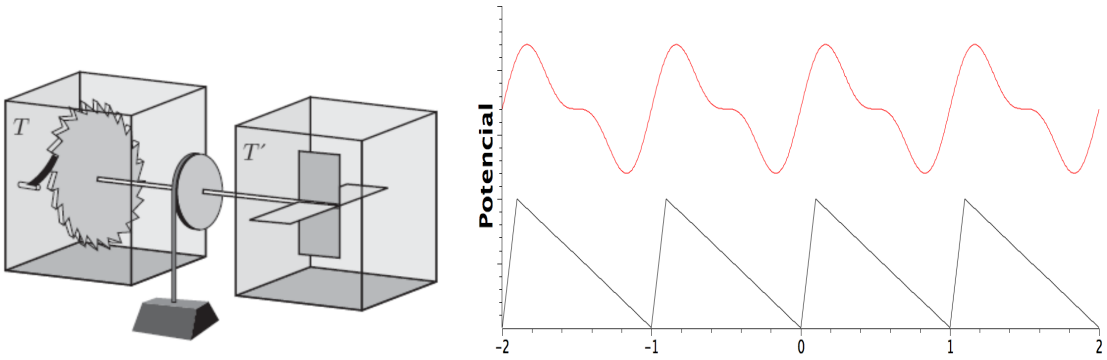


Figura 2: Representación de una máquina de Feynman (izda, extraído de [8]) y ejemplos de potenciales *ratchet* (dcha): en color rojo construido con los dos primeros términos de la función sierra, en negro definido a trozos.

Analíticamente, la imposibilidad de extraer desplazamiento neto de una partícula browniana en equilibrio, a pesar de estar sometida a un potencial asimétrico, se puede deducir a partir de la

fórmula de Kramers para la probabilidad de escape de un pozo de potencial:

$$P_r \propto \exp(-\frac{\Delta V}{k_B T})$$

Esta fórmula establece que la probabilidad de escape es función de la *altura del pozo* (ΔV) y de la temperatura (T); dado que el potencial es periódico, la altura del pozo es la misma en cualquier dirección, luego a igualdad de temperatura la probabilidad de moverse en cualquier sentido será la misma, y no habrá desplazamiento neto.

Con todo lo anterior, se pueden enumerar tres condiciones necesarias para la rectificación del movimiento browniano que conduzca a un motor funcional: *(i)* ruido térmico, *(ii)* asimetría espacial, *(iii)* condición de no-equilibrio.

El ruido térmico es común al funcionamiento de cualquier potencial ratchet, y la asimetría espacial es un parámetro que sólo influye en la eficiencia de un motor dado y no en la manera en que se consigue la rectificación; en general la clasificación de estos potenciales atiende a la manera en que se introduce el no-equilibrio en el sistema, distinguiéndose fundamentalmente dos mecanismos: *rocking ratchet* y *flashing ratchet*.

2.1. *Rocking ratchet*

Las dinámicas *rocking ratchet* se caracterizan porque el desequilibrio (aporte de energía) en el sistema se consigue sometiendo este a la acción de una fuerza periódica [10, 11]. Debido a la pendiente asimétrica del pozo de potencial, la fuerza necesaria para vencer la barrera es diferente en función de la dirección, lo que define un rango de fuerzas en el que la partícula es arrojada en un sentido, pero no en el contrario:

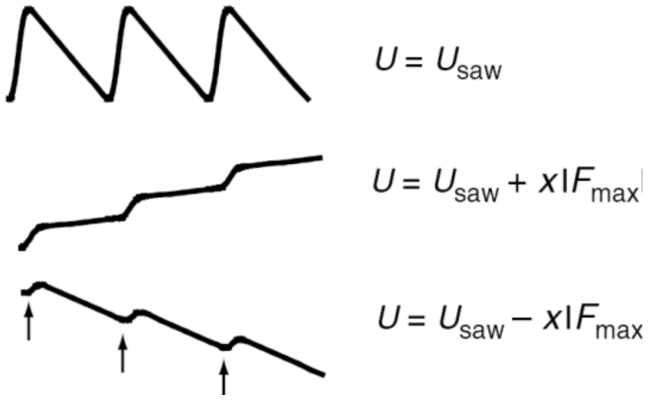


Figura 3: Deformación del potencial ratchet bajo la acción de una fuerza externa periódica: mecanismo *rocking ratchet*. Extraído de [12].

El hecho de que este mecanismo sea determinista a temperatura nula e involucre un cambio en la forma de la función potencial permite asociarlo, en el lenguaje de motores moleculares, con el movimiento *power stroke* mencionado anteriormente, en el que un cambio estructural de la máquina (esto es, un cambio en el potencial de interacción *motor-substrato*) induce el cambio bien definido de posición de la proteína. En estos sistemas la única contribución del ruido térmico es un descenso en la eficiencia del motor, que se maximiza a temperatura nula.

2.2. *Flashing ratchet*

En este mecanismo, el sistema dinámico se aparta del equilibrio mediante la sustitución intermitente y cíclica del potencial ratchet por una función constante, permitiendo que la partícula difunda libremente cuando el potencial asimétrico está desactivado; estadísticamente la partícula en movimiento browniano se extenderá simétricamente durante esta fase, con una densidad de probabilidad que obedece la ecuación de difusión. Sin embargo, debido a la asimetría del ratchet, cuando se introduzca de nuevo el potencial en el sistema un mayor número de eventos tendrá lugar en los que la partícula habrá quedado atrapada por el pozo *siguiente* respecto del *anterior* (entendiendo que *siguiente* se refiere al sentido de la marcha), generando así movimiento neto en una dirección [11]:

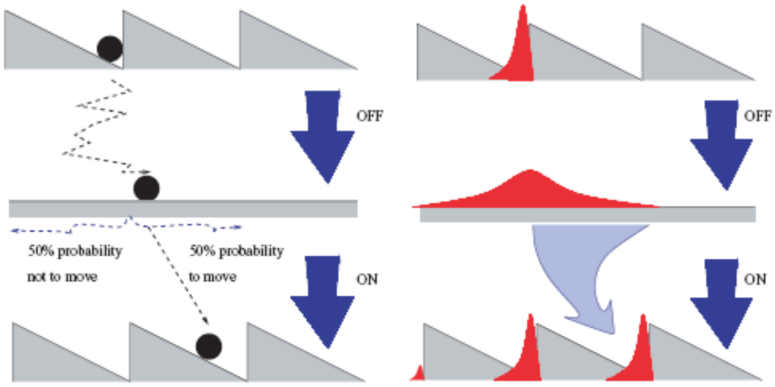


Figura 4: Esquema del mecanismo *flashing ratchet.* De arriba abajo, etapas *ON/OFF/ON.* Extraído de [12].

Este modelo tiene algunas limitaciones claras respecto de la elección de parámetros, en particular la *altura* del potencial ratchet (tiene que ser suficiente para confinar a la partícula a pesar de la agitación térmica), y los tiempos de difusión, que deben ser suficientes para que en la fase difusiva la distribución de probabilidad se extienda apreciablemente sobre la posición *siguiente*, pero no excesivos para evitar que se alcance la posición *anterior*.

En este caso, el ruido térmico es un ingrediente esencial para producir movimiento [3]: el motor tiene eficiencia nula a temperatura cero.

Curva de interpolación teórica del movimiento *flashing ratchet*:

Bajo la suposición de dinámica sobreamortiguada y tiempos largos en cada fase del ciclo *activado/desactivado* (*ON/OFF*), se puede conseguir una expresión teórica de la dinámica bajo un potencial ratchet alternado con movimiento browniano libre. Para ello, se considera que la partícula de masa m se encuentra perfectamente confinada durante la fase ratchet, y difunde libremente durante la fase browniana en un medio de coeficiente de disipación η, de acuerdo con la densidad de probabilidad:

$$P(x,t) = \sqrt{\frac{\eta m}{4\pi k_b T t}}\, e^{-\frac{x^2 \eta m}{4 k_b T t}}$$

En un potencial ratchet de período unidad, con un pozo de potencial contenido entre $a-1 < x < a$ y un tiempo de difusión libre t_{off} constante (y fijado externamente), la distribución de probabilidad

de avanzar n períodos en un ciclo será:

$$P_n(t_{off}) = \sqrt{\frac{\eta m}{4\pi k_b T t_{off}}} \int_{a-1+n}^{a+n} e^{-\frac{x^2 \eta m}{4k_b T t_{off}}} dx$$

Y el promedio de períodos avanzados en un ciclo:

$$< n >= \sum_{n=-\infty}^{\infty} n P_n(t_{off})$$

Por tanto, en un movimiento de período temporal $t_{off} + t_{on}$, la velocidad promedio será:

$$< v >= \frac{< n >}{t_{off} + t_{on}}$$

La validez de esta expresión estará limitada a un rango de parámetros que satisfagan las suposiciones hechas: la densidad de probabilidad difusiva es una solución del movimiento browniano *sobreamortiguado* (coeficiente de disipación alto $\eta \to \infty$); no se tiene en cuenta la posibilidad de que la altura del potencial permita a la partícula escapar térmicamente durante la etapa *ON* ($^{\Delta U}/_{k_b T} \gg 1$), y tampoco se contempla que la partícula no alcance el equilibrio durante esta misma fase ($t_{on} > {}^{\eta m}/_{\Delta V}$) [11].

A pesar de estas limitaciones, disponer de una expresión semejante permite depurar de forma anticipada las técnicas computacionales y numéricas que se emplearán en secciones siguientes, en el contexto de un modelo sencillo que consume pocos recursos de cálculo. Se comparará la curva de interpolación obtenida (evaluada numéricamente para $1 < t_{off} < 10^3$) a temperatura baja ($T = 0{,}01$) y alta ($T = 0{,}1$) con una simulación de la dinámica *flashing ratchet* en un potencial de tipo *sierra* definido a trozos, con parámetros: periodo espacial = 1; factor de asimetría, $a = 0{,}1$; altura del pozo $\Delta U = 30$; $t_{on} = 10^3$:

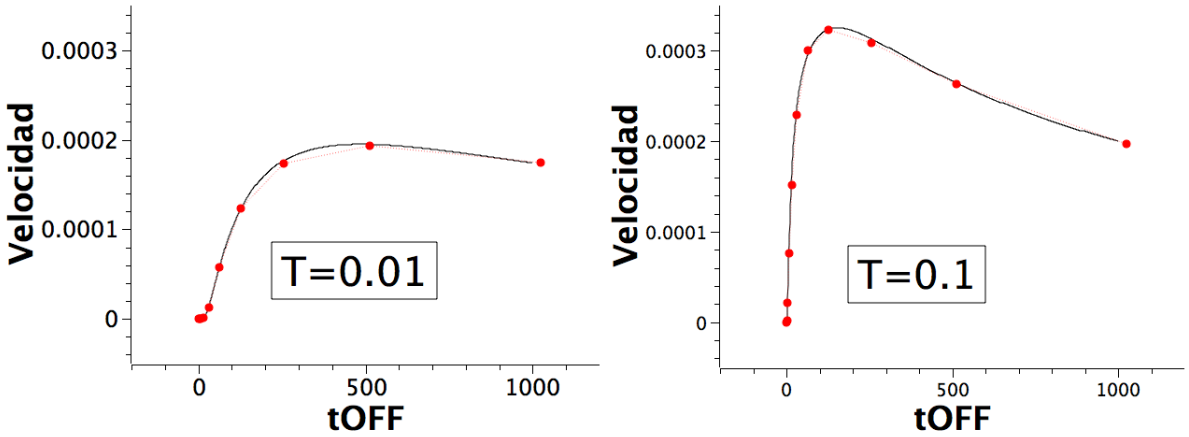


Figura 5: Comparativa de las curva de interpolación (trazo continuo) y simulación (puntos), a temperatura baja (izda) y alta (dcha). Temperatura en unidades arbitrarias.

Con esta elección de parámetros se cumplen los requerimientos de la interpolación y se obtiene un buen acuerdo entre la curva y las simulaciones.

3. Problema biológico - *Helicasa* como motor

El ADN es la macromolécula responsable del almacenamiento de la información hereditaria en una célula. Está compuesto por dos cadenas de polinucleótidos (biomoléculas compuestas por un grupo nitrogenado, un monosacárido y un grupo fosfato), entre los cuales se pueden identificar en función de su grupo proteico (*adenina, timina, guanina* y *citosina*) cuatro subunidades diferentes que se enlazan complementariamente mediante dos (*A-T)* o tres *(G-C*) puentes de hidrógeno. Los residuos de nucleótidos adyacentes perteneciente a una misma hebra se unen de manera asimétrica, formando una suerte de «soporte» para las bases nitrogenadas compuesto alternativamente por azúcar/fosfato. Esta asimetría permite polarizar cada hebra de ADN, distinguiendo (por convenio) los extremos *extremo-3'* y *extremo-5'* .

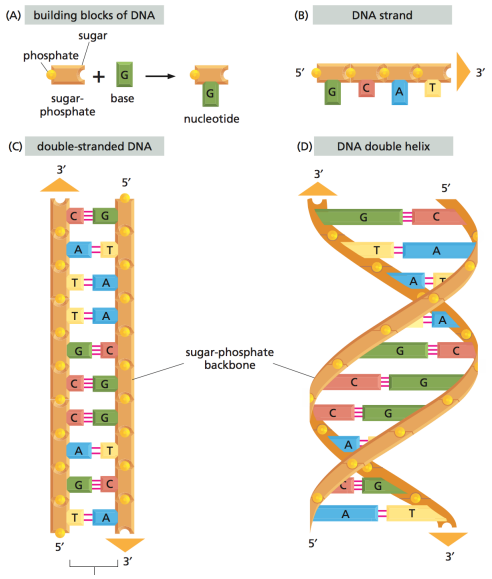


Figura 6: Esquema de la composición y estructura del ADN a distintas escalas. Extraído de [13].

El ADN contiene toda la información biológica (esto es, la composición de todas las proteínas que un espécimen podrá sintetizar) que debe ser copiada y transmitida con precisión a las células hijas durante la división celular; esto plantea dos importantes interrogantes: en primer lugar, de qué manera se almacena la información en el ADN, y en segundo lugar cómo se transmite esta información a lo largo de una estirpe celular. La codificación de información en la cadena tiene lugar mediante la relación unívoca entre un triplete de bases nitrogenadas y un aminoácido (biomoléculas constituyentes de las proteínas), de manera que en el proceso de síntesis proteica media una fase de «descodificación» del ADN (*transcripción*). Por otro lado, la transmisión de la información a la descendencia requiere la duplicación de cada molécula de ADN en la célula (*replicación*). Ambos procesos, esenciales para la vida en La Tierra, dependen en primer lugar de que la célula disponga de recursos para acceder a la secuencia de bases de su material genético, de forma que se pueda utilizar la información que contiene: el acceso a la hebra permite crear una copia «negativa» de la secuencia de un gen, bien directamente en una tercera hélice de ADN durante la replicación, bien en un ácido nucleico intermediario llamado *ARN mensajero* durante la transcripción. El fenómeno de apertura espontánea de la molécula de ADN debido a fluctuaciones térmicas tiene lugar a la *temperatura de desnaturalización* (alrededor de los 60^{o}C), muy por encima de las temperaturas óptimas en biología, de manera que la separación de las hebras a temperaturas biológicas es un proceso catalítico *activo* llevado a cabo por la cooperación coordinada de varias proteínas. Las

funciones básicas de esta *máquina de replicación/transcripción,* a saber la síntesis de nuevo material genético y el desplazamiento del complejo a nuevos lugares de la secuencia, están llevadas a cabo por las proteínas *ADN/ARN-polimerasa* y *helicasa* respectivamente; otras proteínas menores del complejo se encargan de iniciar el proceso a partir de la hebra completamente cerrada, de ejecutar tareas específicas propias de la manera en que se organiza el ADN en un tipo celular concreto, etc. El sujeto de este trabajo es la función de las helicasas como motores moleculares de translación [13].

Las *helicasas* son una familia de proteínas capaces de emplear la energía libre del metabolismo de ATP para catalizar el desenrollamiento de las moléculas de ADN o ARN, así como desplazar relativamente dos hebras de material genético (traslocación de la cadena); de este modo, son motores moleculares en el sentido amplio descrito anteriormente.

Las helicasas se pueden clasificar atendiendo a criterios diversos:

1. Secuencias de aminoácidos compartidas en la estructura proteica, dando lugar a la identificación de subfamilias de helicasas denotadas *SFn*, donde n es un número natural.
2. El tipo de ácido nucleico sobre el que actúa, clasificándose en *DNA-helicasas*, *RNA-helicasas* o *helicasas híbridas.*
3. La dirección en que recorren la cadena de material genético ($5' \to 3'$ o $3' \to 5'$).
4. La especie que las sintetiza (helicasas humanas, vegetales, bacterianas, etc).
5. El número de dominios *ATP-asa* para la hidrólisis de ATP: las helicasas se pueden clasificar en monoméricas o multiméricas (siendo diméricas y hexaméricas los casos más frecuentes).

Las magnitudes cuantitativas más características de las helicasas son el *tamaño del paso* (distancia media recorrida por el centro de masas de la proteína por cada molécula de ATP consumida) y la intensidad del *acoplo a la reacción de ATP*, definida como el número medio de moléculas de 'combustible' consumidas por cada par de bases (en adelante *bp*) desenrolladas; estas magnitudes no se relacionan trivialmente debido a la posibilidad de reacciones fallidas que no produzcan movimiento [1].

Actualmente hay multitud de preguntas abiertas acerca de la dinámica de estas proteínas, cuyas respuestas pertenecen al terreno de la especulación y la hipótesis; alguna de estas incluyen la traslocación del ADN producida impulsivamente (*power stroke*) o como resultado de una búsqueda térmica, la formación de burbujas de manera pasiva (explotando las fluctuaciones térmicas que abren espontáneamente la cadena) o activa (consumiendo ATP), etc.

4. Modelización de la interacción proteína-DNA

Al diseñar un modelo en la mesoescala, debe introducirse *ad hoc* una familia de parámetros efectivos relacionados entre sí de forma que se reproduzca (dentro de un rango) la fenomenología que emerge naturalmente en los modelos microscópicos.

La dinámica del modelo estará descrita por el hamiltoniano total del sistema, *i.e* la interacción de la proteína con la cadena de ADN más la energía de la propia cadena (todavía no se están considerando cambios internos en la proteína).

4.1. Hamiltoniano de la cadena: modelo de Peyrard-Bishop-Dauxois modificado

Este modelo unidimensional describe la cadena de ADN asignando a cada par de bases una función y_i, que expresa su apertura (siendo $y_i = 0$ la posición de equilibrio) y constituye la magnitud relevante para estudiar la formación de burbujas y desnaturalización de esta molécula.

Los términos que contribuyen a la energía potencial de la cadena aislada son el potencial de Morse *(V)* que contiene la interacción característica de los puentes de hidrógeno entre las bases de un mismo par, más un término que modeliza la solvatación de las bases nitrogenadas abiertas en contacto con el medio acuoso (impidiendo la clausura de la cadena y estabilizando las burbujas), y un término de «stacking» (W) que describe la interacción entre pares de bases adyacentes vía sus grupos fosfato; estos elementos constituyen el modelo PBD modificado [14, 15], que está descrito por una función hamiltoniana:

$$H_C = \sum_{i=1}^{N} \left[\frac{p_i^2}{2m} + V(y_i) + W(y_i, y_{i-1}) \right]$$

Donde:

$$V(y) = D(e^{-a\,y} - 1)^2 + G\,e^{-(y-y_0)^2/b}$$

$$W(y_i, y_{i-1}) = \frac{1}{2} K\,(1 + \rho\,e^{-\alpha(y_i + y_{i-1})})\,(y_i - y_{i-1})^2)$$

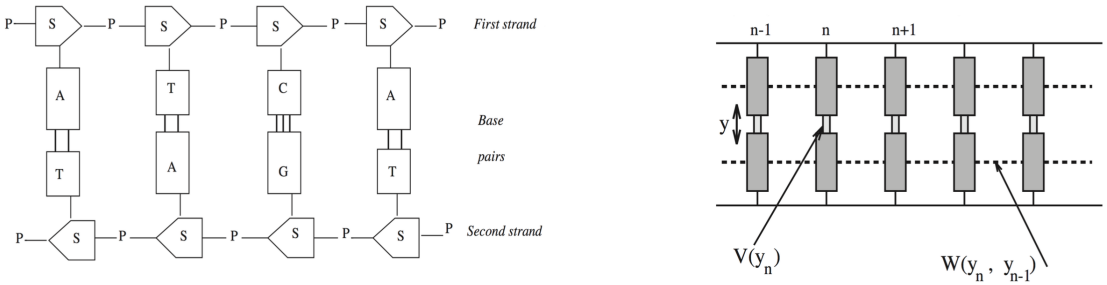


Figura 7: Esquema conceptual del ADN detrás del modelo *PBD*. Extraído de [14].

Este modelo ha sido estudiado en detalle [16], y los valores de los parámetros ajustados al ADN pueden encontrarse en la literatura: $D_{AT} = 0{,}043\,eV$ (modelo PBD sin barrera de solvatación), $D_{AT} = 0{,}052\,eV$ (con barrera de solvatación), $D_{CG} = 1{,}5 D_{AT}$, $a_{AT} = 4\,\text{Å}^{-1}$, $a_{CG} = 1{,}5\,\alpha_{AT}$, $G = 3D_{AT}$, $y_0 = {}^2/_{\alpha}$, $b = {}^1/_{2\alpha^2}$, $K = 0{,}03\,eV/\text{Å}^2$, $\rho = 3$, $\alpha = 0{,}8\,\text{Å}^{-1}$, $m = 150\,u.m.a$ (masa reducida de un par de bases)[16, 15].

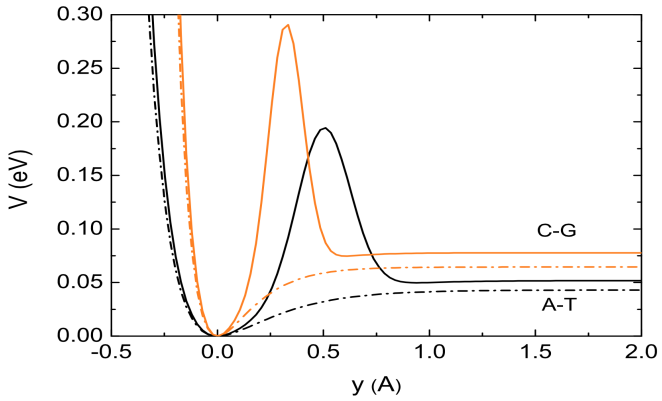


Figura 8: Potencial de Morse con (línea sólida) y sin (línea discontinua) barrera para A-T (línea negra) y C-G (línea naranja). Extraído de [16].

4.2. Interacción de la proteína con la cadena

Restringidos a la interacción helicasa/ARN polimerasa-DNA, se propone un modelo de interacción que contemple los siguientes aspectos:

1. Estas moléculas son motores con una alta *procesividad*, esto es, actúan durante largos períodos de tiempo sin desligarse de la hebra de ADN. En virtud de esta característica nos limitaremos a un modelo unidimensional donde la variables espacial corresponda a *posición en la cadena.*

2. El ADN nativo cumple una función meramente estructural, y cualquier etapa de la vida celular que requiera de información genética implica la apertura parcial de esta molécula. Una parte esencial de la actividad biológica de los enzimas helicasa y ARN polimerasa es la apertura de *burbujas* (horquillas) de fase desnaturalizada en regiones limitadas de la doble hélice; físicamente: la interacción entre los motores y la cadena será estable (*i.e* presentará un *mínimo de energía*) cuando estos se sitúen sobre regiones abiertas de la hebra.

3. La acción de los motores sobre el substrato es *local*: la interacción debe limitarse a la extensión espacial de la proteína.

Se propone una interacción dependiente de tres parámetros:

- $\sigma \equiv$ Anchura efectiva de la proteína, $[\sigma] = L$
- $B \equiv$ Intensidad efectiva de la interacción, $[B] = E{\cdot}L$
- $\gamma^{-1} \equiv$ Amplitud de apertura de la burbuja, $[\gamma] = L^{-1}$

Relacionados mediante el potencial:

$$V_{int}(X_p, \{y\}) = -\frac{B}{\sigma\sqrt{\pi}} \sum_i \tanh(\gamma\, y_i)\, e^{-\frac{(X_p - i\cdot\, \mathrm{d})^2}{\sigma^2}}$$

Donde:

$X_p \equiv$ Posición de la proteína sobre la hélice de DNA

$y_i \equiv$ Apertura de un par de bases

$d \equiv$ Distancia entre dos pares de bases consecutivas

La suma se extiende sobre todos los pares de bases de la cadena.

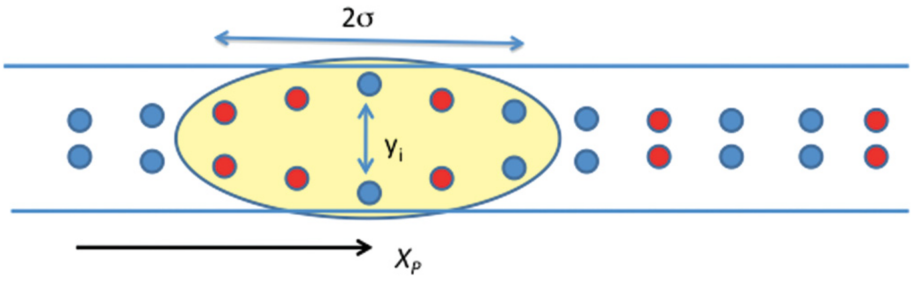


Figura 9: Ilustración esquemática del modelo de interacción proteína-DNA. Extraído de [15].

Teniendo en cuenta la interacción descrita arriba, la expresión del hamiltoniano total del sistema (independiente del tiempo, *i.e* termalizable) será:

$$H = H_C + H_P = \sum_{i=1}^{N} \left[\frac{p_i^2}{2m} + V(y_i) + W(y_i, y_{i-1}) \right] + \frac{p_p^2}{2m_p} + V_{int}(X_p, \{y\})$$

Donde la masa de la proteína [15]: $m_p = 2250\,[u.m.a] = 15\,u_m$.

Con el fin de garantizar la estabilidad numérica de las simulaciones, se emplearán las siguientes unidades adecuadas a este problema: distancia, $u_L = d = 3{,}3\,\text{Å}$; energía, $u_E = D_{AT} = 0{,}043\,eV$; masa, $u_m = m = 150\,u.m.a$; tiempo, $u_t = \sqrt{\frac{md^2}{D_{AT}}} = 1{,}99\,ps$; fuerza, $u_F = \frac{D_{AT}}{d} = 20{,}87\,pN$; la temperatura se expresará en unidades de energía $\tilde{T} = {}^{k_b T}/_{D}$, pero seguiré utilizando el símbolo T ($\tilde{T} \equiv T$) en adelante. Salvo que se indique lo contrario, las magnitudes adimensionadas se referirán a este sistema de unidades.

5. Modelo modificado: movimiento direccional

El modelo descrito captura adecuadamente la interacción de las proteínas con la apertura de la cadena; sin embargo, no contiene ningún mecanismo para introducir energía en el sistema, y en contacto con un baño térmico terminará por alcanzar el equilibrio: un modelo semejante no puede reproducir el movimiento de un motor molecular, tal y como se discutió en las secciones introductorias; para solucionar esto, los motores moleculares emplean la energía química liberada del metabolismo de ATP, y los experimentos muestran una clara correlación 1:1 entre el número de moléculas consumidas y el avance del motor en unidades de d [7]. Inspirados por este resultado, en este trabajo se ha modificado el modelo anterior definiendo un segundo potencial de interacción asimétrico e introduciendo dos estados diferenciados en el motor, de tal manera que cada uno de ellos se asocie con un hamiltoniano diferente. La transición entre uno u otro estará mediada conceptualmente por la adsorción-de-ATP/liberación-de-ADP , y operativamente mediante unos tiempos característicos (que se llamarán en adelante t_{on}, t_{off}) y que indicarán el intervalo de actuación de cada potencial. Se despreciará el tiempo de transición entre estados en este modelo.

De esta forma, las ecuaciones a resolver para la dinámica serán:

$$\begin{cases} m\ddot{y}_i = -\frac{\partial(V+W)}{\partial y_i} - \frac{\partial V_{int}}{\partial y_i} - \eta m\dot{y}_i + \xi_i(t) \\ \\ m_p\ddot{X}_p = -\frac{\partial V_{int}}{\partial X_p} - m_p\eta_p\dot{X}_p + \xi_p(t) \end{cases}$$

donde las cantidades ξ son variables aleatorias que representan la influencia del ruido térmico en el sistema (ver apéndice A).

Se tomarán los siguientes valores para las constantes de disipación [15]: $\eta = 500\,GHz = 1\,u_t^{-1}$; $\eta_p = 10\,THz = 20\,u_t^{-1}$.

Se proponen dos modelos diferentes de potencial asimétrico, que actuarán alternándose con el potencial (simétrico) definido en la sección previa; en estos, el potencial *on* corresponde a la configuración nativa del motor, cuya estructura química interactúa asimétricamente con la cadena. La energía proveniente de la hidrólisis de ATP debilitará la unión ADN-proteína, permitiendo que esta se mueva con cierta libertad, confinada en un pozo simétrico de potencial (estado *off*). Una vez terminada la reacción y liberados los productos ($ADP + P_i$), la partícula vuelve a estar confinada asimétricamente.

5.1. Potencial asimétrico con carácter *power stroke*: *skew gaussian*

El primer modelo que se propone como potencial asimétrico es una modificación del potencial de interacción ya descrito, sustituyendo la función gaussiana por una función *gaussiana asimétrica* (o *skew gaussian*), definida como:

$$f(x) = \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}} \left[1 + \mathrm{fer}(\frac{\alpha x}{\sqrt{2}})\right]$$

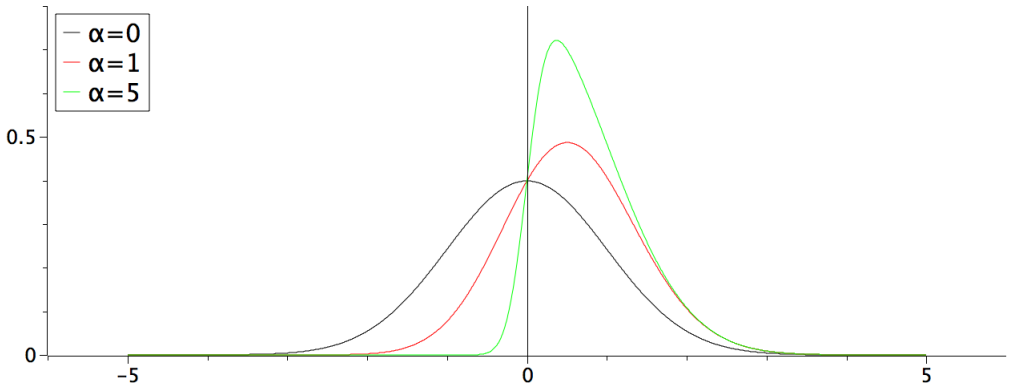


Figura 10: Curvas gausianas asimétricas para varios valores de asimetría (a).

donde a es el *factor de asimetría* de la función.

El el potencial de interacción asimétrico (en adelante V_{on} para distinguirlo del simétrico, que se denotará V_{off}) será:

$$V_{on}^{int} = -\frac{B}{\sigma\sqrt{\pi}} \sum_i \left\{ \tanh(\gamma\, y_i)\, e^{-\frac{(X_p - i)^2}{\sigma^2}} \left[1 + \mathrm{fer}\left(\frac{\alpha(X_p - i)}{\sigma}\right)\right]\right\}$$

En la siguiente figura se observa un diagrama de potencial *OFF/ON* para este modelo, correspondiente a una burbuja centrada en $i = 5$ para una pequeña cadena de $10\,bp$.

También se presenta un ejemplo de trayectoria de la proteína a diferentes temperaturas; el comportamiento *power stroke* queda en relieve en la simulación a temperatura nula, donde se consigue la máxima eficacia del motor (un paso por molécula de ATP consumida):

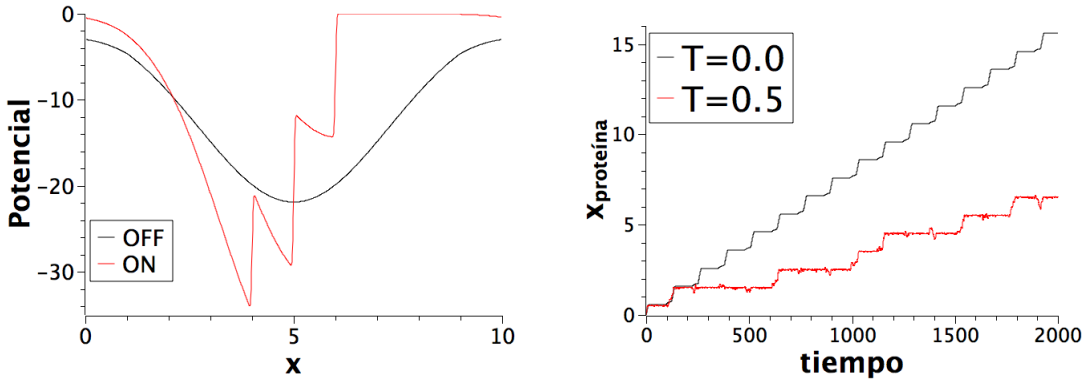


Figura 11: Potenciales *ON* / *OFF* para el modelo *Skew Gaussian* (izquierda). Ejemplo de trayectoria a temperatura baja/alta; se aprecia el carácter *power stroke* de este motor.

5.2. Potencial de *búsqueda térmica:* modelo *ratchet*

Se introduce un segundo modelo de potencial asimétrico, que dirigirá el movimiento direccional de la partícula sobre la cadena de ADN vía una dinámica de *búsqueda térmica* (ver sección primera). En este caso, el ruido térmico será un requisito indispensable para el funcionamiento del motor; la forma funcional de la interacción propuesta es:

$$V_{on}^{int}(X_p, \{y\}) = V_{off}^{int} \cdot V^{ratchet}(X_p)$$

Donde $V^{ratchet}$:

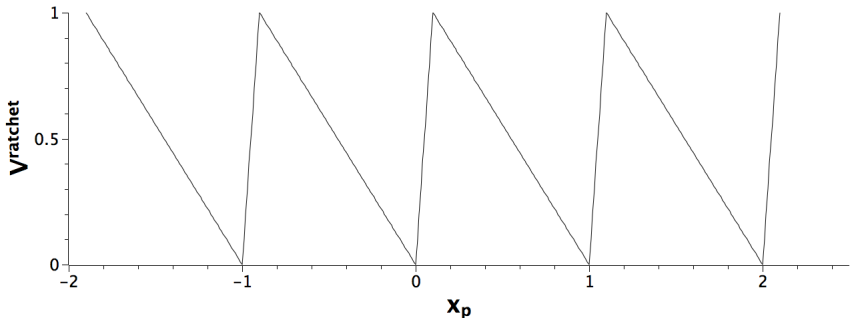


Figura 12: Función $V^{ratchet}$ definida a trozos, con un factor de asimetría $\alpha = 0{,}1$.

Como en el caso anterior, se incluye el diagrama de potencial para una pequeña cadena con una burbuja centrada en $i = 5$, así como un ejemplo de trayectoria de la partícula a temperatura alta y nula. En este caso se evidencia la pérdida de eficacia con el descenso de la temperatura:

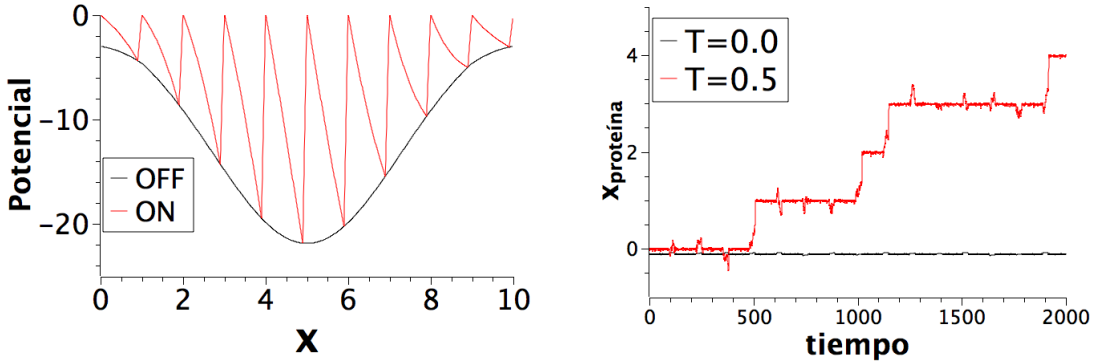


Figura 13: Comparativa de potenciales *ON/OFF* en el modelo *ratchet* (izquierda). Ejemplo de trayectoria a temperaturas baja/alta, donde se observa la necesidad del ruido térmico (derecha).

Estos mecanismos representan las dos alternativas mayoritariamente discutidas actualmente como realización microscópica del paso en los motores celulares. Si bien en la escala espacial de la helicasa la resolución instrumental es insuficiente, experimentos de *molécula única* en motores con tamaño de paso mayor (*e.g* kinesinas encargadas de desplazar cargas a través de la célula) han permitido determinar una combinación de ambas dinámicas como origen del movimiento [17](ver apartado *7,«Discusión y conclusiones»*).

6. Exploración de parámetros y resultados

La exploración del espacio de parámetros tiene por objetivo ajustar el modelo para que este sea verdaderamente representativo del proceso biológico; idealmente esto supondría explorar cada combinación de parámetros para cada magnitud susceptible de ser comparada con medidas experimentales, pero dada la inviabilidad computacional de simular ciegamente, se debe buscar algún criterio racional que restrinja la exploración: en este problema dicho criterio será la maximización de alguna magnitud adecuada que dependa de la configuración paramétrica, y que se puede justificar por que el proceso de *evolución biológica* es de hecho, en la escala celular, un proceso de optimización energética. A la hora de seleccionar una magnitud respecto de la cual optimizar la elección de parámetros, la opción más obvia es la *velocidad promedio* de la partícula, $< v > (t_{on}, t_{off})$:

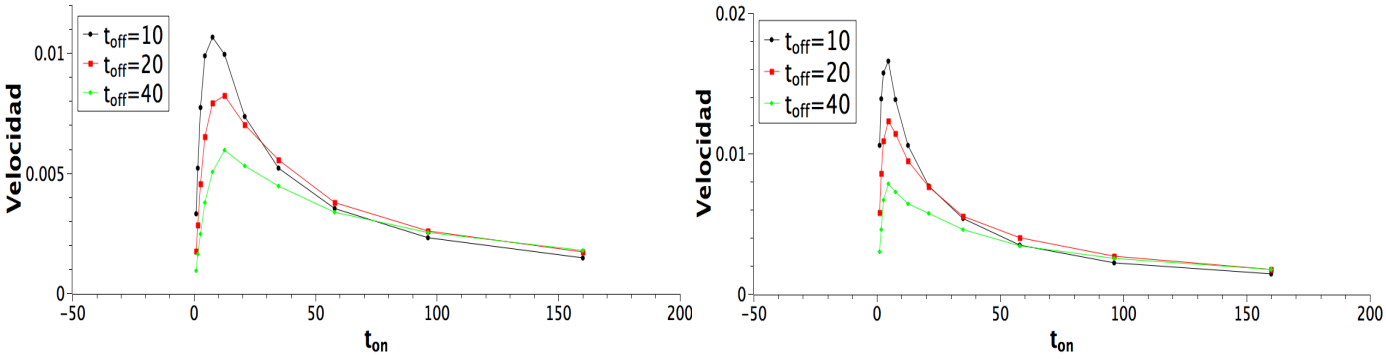


Figura 14: Velocidad media en función de t_{on}, para distintos t_{off}. Modelos *power stroke* (izda), *ratchet* (dcha).

(ver apéndice B para una justificación de los rangos t_{on}, t_{off} estudiados).

Sin embargo, la biología de la helicasa/RNA-polimerasa incluye procesos (*e.g* síntesis de nuevo material genético) que involucran tiempos característicos mayores que la catálisis de ATP, resultando muy difícil abstraer su función como motor. En su lugar, resulta mucho más conveniente utilizar como magnitud directora la *eficiencia* (ϵ), definida como el número de *bp* avanzadas por ciclo. En las unidades de trabajo, esta definición coincide con el avance por ciclo:

$$\epsilon =< v > \cdot(t_{on} + t_{off})$$

A pesar de que físicamente t_{on} es una variable aleatoria (en tanto que representa el tiempo de espera del motor entre el fin de un ciclo y la adsorción de una nueva molécula de ATP), se mantendrá fijo durante la exploración paramétrica con el fin de no camuflar su influencia en el sistema.

La desnaturalización térmica del ADN tiene lugar a una temperatura que depende de la longitud de la cadena, composición, concentración de sal en el medio acuoso y otros parámetros difícilles de controlar; para evitar que la dinámica del motor se vea condicionada de un modo no deseable, se realizarán las simulaciones a una temperatura superior a los 0º C, pero moderada: $T = 6{,}5$ºC.

Por último, los parámetros $\sigma = 3{,}0$ y $\gamma = 0{,}2$ están ajustados atendiendo a criterios biológicos [16, 15], de forma que su valor se mantendrá constante en todas las simulaciones.

A continuación se introduce un ejemplo de exploración del espacio de parámetros para una monocadena A-T de *100 bp*, $T = 0{,}56$, sin barrera de solvatación (ver *«Discusión y conclusiones»*):

6.1. Modelo *Skew Gaussian*

Para optimizar la eficiencia del motor respecto de los parámetros de tiempo se procede de la siguiente manera:

A partir de varias curvas $\epsilon(t_{on})$, $v(t_{on})$ para t_{off} fijo, se obtiene la curva de la eficiencia y velocidad máxima en función de t_{off}:

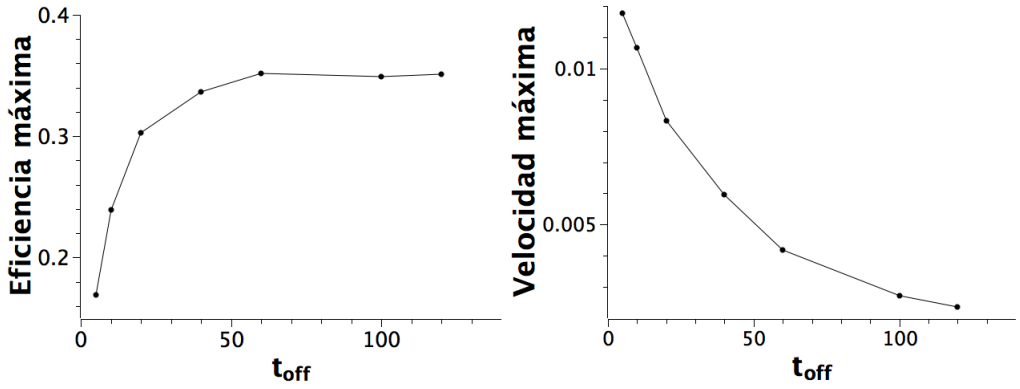


Figura 15: Eficiencia de saturación de la función $\epsilon(t_{on})$ frente t_{off} (izda), y velocidad máxima de la función v(t_{on}) frente t_{off} (dcha); modelo *Skew Gaussian.*

De la figura de la izquierda se puede extraer una cota mínima de t_{off} para el modelo, tomando el valor de tiempo para el cual la eficiencia satura (lo cual se debe al confinamiento de la proteína en el pozo gaussiano de potencial); dada esta curva concreta, se obtiene $t_{off} > 60$. A continuación, para seleccionar un único valor del conjunto, se maximiza la velocidad (curva de la derecha) compatiblemente con la restricción anterior; dado que la curva es monótonamente decreciente (en este rango), el valor que se utilizará en este modelo será $t_{off} = 60$.

Seguidamente se repite el proceso para t_{on}; las curvas de eficiencia y velocidad con el parámetro $t_{off} = 60 = \text{cte}$:

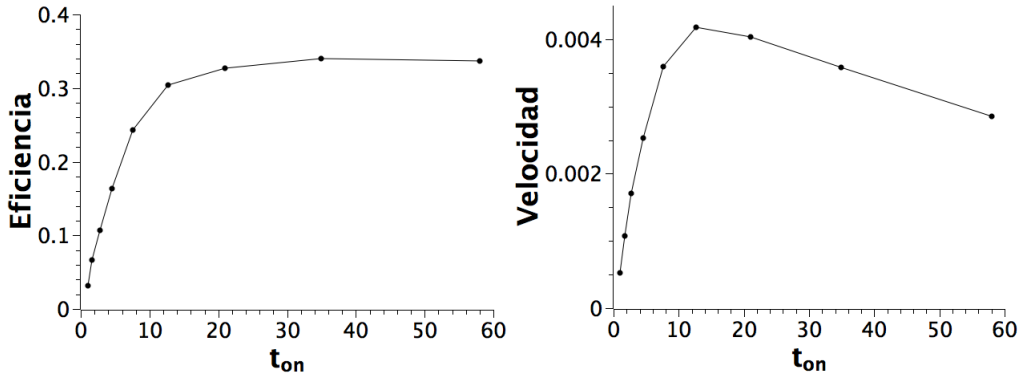


Figura 16: Curvas de eficiencia (izda.) y velocidad (dcha.) frente t_{on} del modelo *Skew* Gaussian, para $t_{off} = 60$.

De la curva de la izquierda se obtiene la restricción $t_{on} > 35$, que proporciona una velocidad máxima (figura de la derecha) para $t_{on} = 35$

Para estudiar la eficiencia del motor en función del acoplo proteína-cadena, se obtiene la curva

$\epsilon(B)$ a partir de varias simulaciones donde se ha variado el parámetro de interacción, manteniendo constantes los tiempos obtenidos arriba. Para $10 < B < 100$:

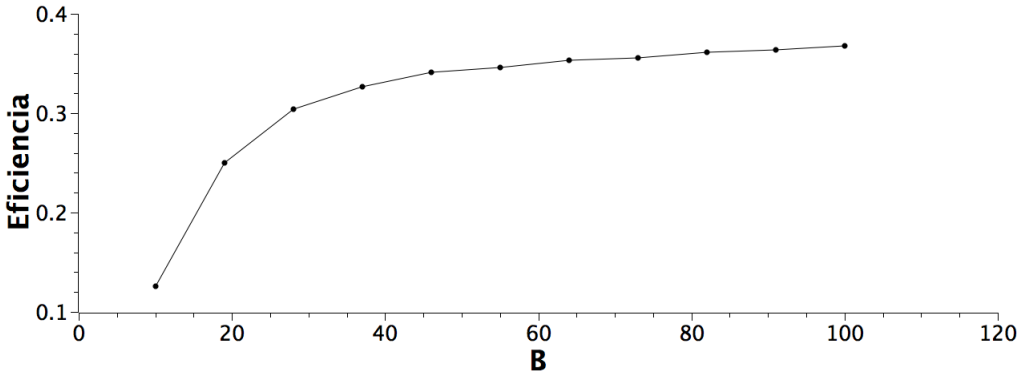


Figura 17: Curva de eficiencia frente B del modelo *Skew Gaussian,* para $t_{off} = 60$, $t_{on} = 35$.

La intensidad del acoplo produce un movimiento máximamente eficaz para $B = 100$, de forma que en adelante se utilizará este valor (que proporciona interacciones del orden de la energía de disociación de Morse en cada par de bases [15]) para la dinámica *Skew Gaussian.*

6.2. Modelo *Ratchet*

Siguiendo los pasos y la metodología anterior, se busca en primer lugar optimizar la eficiencia del modelo respecto de los tiempos característicos *ON/OFF.*

A partir de las curvas $\epsilon(t_{on})$ a $t_{off} = \text{cte}$ se construye la curva de eficiencia máxima $\epsilon_{max}(t_{off})$, de la cual se obtiene un conjunto de potenciales valores t_{off}, que finalmente se discriminan buscando que maximicen la velocidad máxima, $v_{max}(t_{off})$:

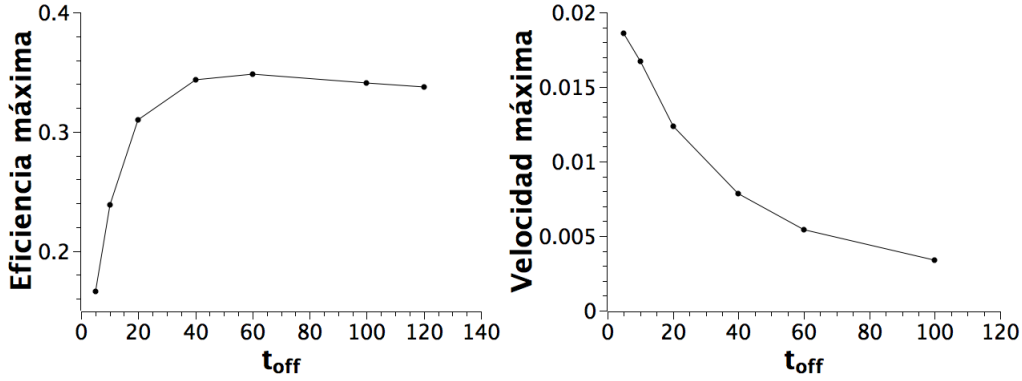


Figura 18: Eficiencia de saturación de la función $\epsilon(t_{on})$ frente t_{off} (izda), y velocidad máxima de la función v(t_{on}) frente t_{off} (dcha); modelo *Ratchet.*

Como en el caso anterior, este procedimiento lleva a tomar $t_{off} = 60$.

Maximizando ahora sucesivamente la eficiencia y velocidad para $t_{off} = 60 = \text{cte}$:

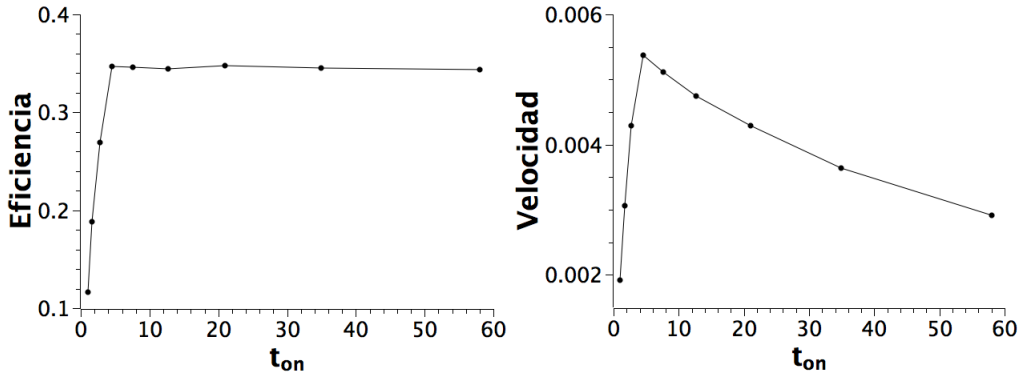


Figura 19: Curvas de eficiencia (izda.) y velocidad (dcha.) frente t_{on} del modelo *Ratchet,* para $t_{off} = 60$.

De estas curvas se obtiene $t_{on} = 4{,}6$

Manteniendo constantes estos tiempos, se construye la curva $\epsilon(B)$ lanzando una serie de simulaciones para distintos valores del parámetro B:

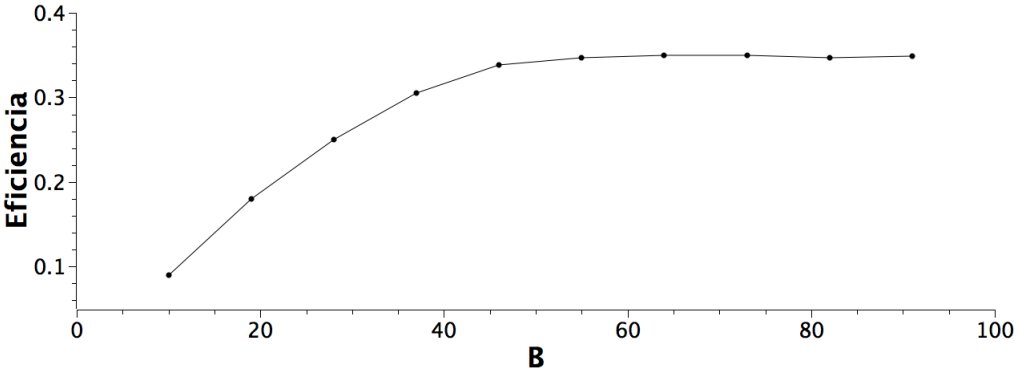


Figura 20: Curva de eficiencia frente B del modelo *Ratchet,* para $t_{off} = 60$, $t_{on} = 4{,}6$.

En este caso, esta curva alcanza la saturación para $B \approx 55$, de manera que en lo que sigue las simulaciones con el modelo *Ratchet* se llevarán a cabo tomando $B = 55$.

6.3. Simulación de ATP disuelto en el medio celular: velocidad y fuerza de detención

El potencial V^{off} actuará durante un tiempo breve y definido (representando el tiempo de reacción del ATP), mientras el intervalo t_{on}, que representa el tiempo de espera del motor para la adsorción de una molécula de 'combustible' (disuelto homogéneamente en el medio celular) entre ciclos, se distribuirá aleatoriamente:

Si la adsorción es un fenómeno descorrelacionado, la probabilidad de adsorber n moléculas en el

intervalo Δt seguirá una distribución poissoniana:

$$\mathcal{P}(n, \Delta t) = \frac{e^{-\nu \Delta t} (\nu \Delta t)^n}{n!}$$

Así mismo, la probabilidad de adsorber *alguna* molécula en Δt:

$$\sum_{n=1}^{\infty} \mathcal{P}(n, \Delta t) = 1 - e^{-\nu \Delta t} \equiv \mathcal{P}(\Delta t)$$

Tomando $\Delta t = t_{on}$, la probabilidad de haber registrado algún evento en dicho intervalo será precisamente:

$$\mathcal{P}(t_{on}) = \int_0^{t_{on}} P(t'_{on}) dt'_{on}$$

De manera que:

$$\frac{d\mathcal{P}(t_{on})}{dt_{on}} = P(t_{on})$$

$$P(t_{on}) = \nu \, e^{-\nu t_{on}}$$

Siendo ν (tasa media de adsorción) una magnitud relacionada con la concentración de ATP.

Cabe esperar que en cada ciclo, el motor requiera de un tiempo mínimo para realizar cambios internos en escalas espaciales inferiores a la resolución de un modelo mesoscópico; para modelizar esto se introduce un *tiempo muerto* (t_m) determinado en el intervalo estocástico: $t_{on}^{total} = t_{on}^{\text{estocástico}} + t_m$, donde $t_{on}^{\text{estocástico}}$ sigue la distribución $P(t_{on})$.

Experimentalmente se ha identificado el comportamiento de los motores celulares frente a la concentración de ATP con una *cinética de Michaelis-Menten [7]*:

$$v = v_{max} \frac{[ATP]}{[ATP] + K_m} \rightarrow v_{max} \frac{\nu}{\nu + K'_m}$$

En el estudio expuesto a continuación se pone de relieve cómo la incorporación del mecanismo descrito para simular la concentración de ATP conduce a la obtención de una ley de Michaelis-Menten para la velocidad del motor.

También se obtendrán distintas curvas de velocidad frente a la fuerza de carga F_{ext}, introducida en el modelo mediante una fuerza externa constante opuesta al sentido de la marcha de la proteína. En particular, por ser una magnitud representativa de la fuerza que es capaz de ejercer el motor (y habitualmente medida en los experimentos), se estimará la *fuerza de detención* definida como la fuerza a la que la velocidad de la partícula se anula, F_s.

Skew Gaussian

Se lanzan varias simulaciones para distintos valores del parámetro ν (0,001 $< \nu <$ 1). Al aumentar el valor del tiempo muerto, las curvas $v(\nu)$ se aproximan a una ley de Michaelis-Menten; si esto es efectivamente cierto, las magnitudes inversas se relacionarán de forma lineal:

$$v^{-1} = v_{max}^{-1} \frac{\nu + K'}{\nu} = \nu^{-1} \alpha + \beta$$

Luego para verificar cuantitativamente esta observación, se ajusta la curva $v^{-1}(\nu^{-1})$ a una recta:

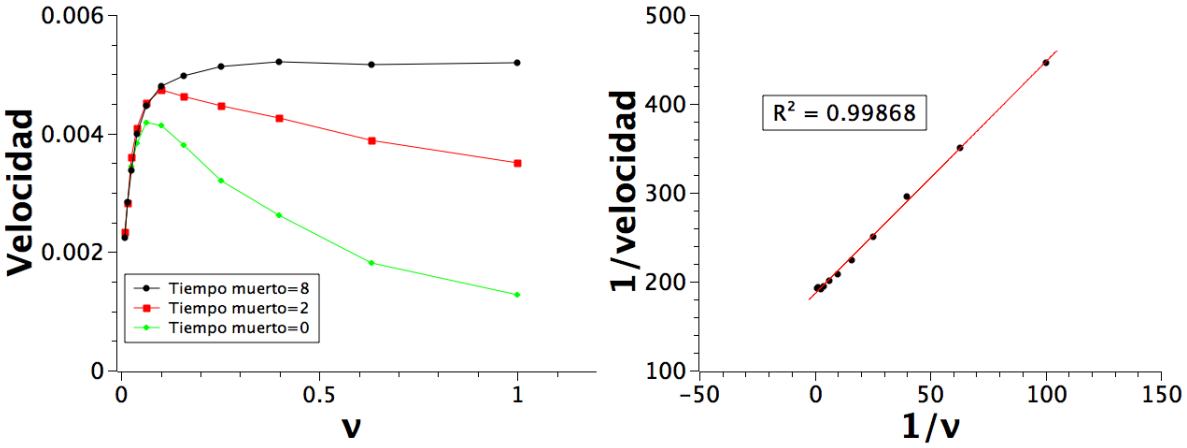


Figura 21: Comparativa para distinto *tiempo muerto* (izda). El buen ajuste a una recta de la magnitud $1/v$ frente a ν^{-1} para $t_m = 8$ (dcha) permite verificar la ley de Michaelis-Menten.

Para este modelo, el ajuste óptimo encontrado tiene lugar para $t_m = 8$; a pesar de que las curvas de velocidad simuladas sin tiempo muerto están adecuadamente descritas por una ley de Michaelis-Menten para valores pequeños del parámetro ν, cabe destacar que la introducción de un tiempo de espera mínimo en la dinámica resulta necesario para obtener este comportamiento en todo el rango estudiado de la tasa de adsorción.

Manteniendo ahora constante el parámetro ν, se obtienen curvas $v(F_{ext})$ a partir de diversas simulaciones para distintos valores F_{ext}:

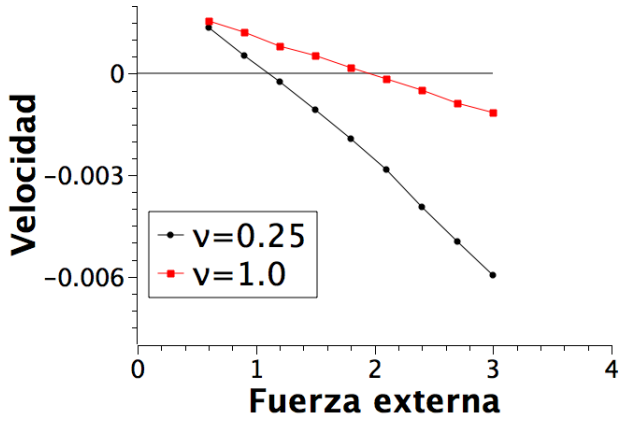


Figura 22: Velocidad frente a la fuerza de carga. Se ha marcado la línea $v = 0$.

La fuerza de detención hallada para este modelo (ν máximo), $F_s(\nu = 1{,}0) = 1{,}9 = 39{,}7\,pN$; experimentalmente se ha reportado para la *ARN-polimerasa* $F_s^{exp} \approxeq 35\,pN$[18] (ver *«Discusión y conclusiones»*).

Ratchet

Se obtienen una serie de curvas $v(\nu)$ para distintos valores del tiempo muerto, y se estudia su adecuación a una ley de Michaelis-Menten mediante el ajuste a una recta de las magnitudes inversas:

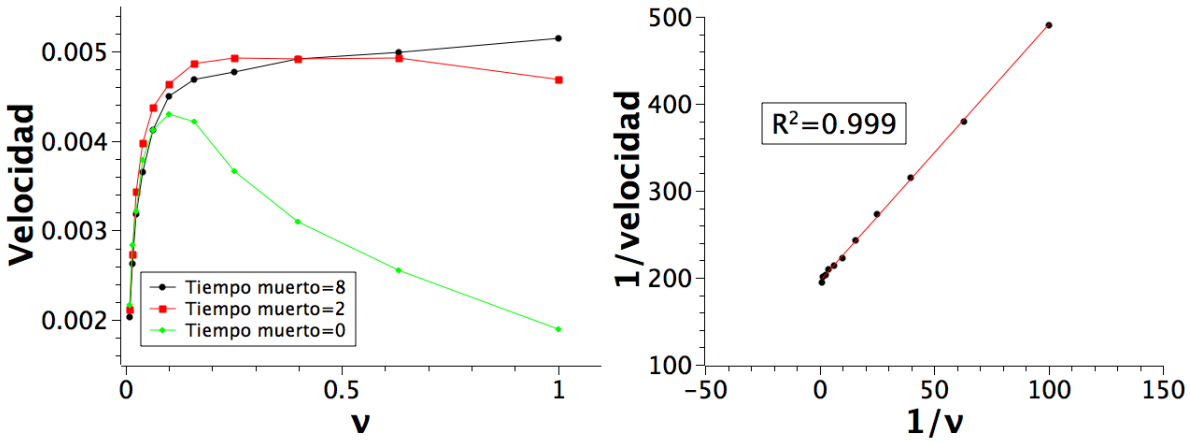


Figura 23: Dinámica para distinto *tiempo muerto* en el modelo Ratchet, y ley de Michaelis-Menten.

Como en el caso anterior, el modelo *Ratchet* se ajusta bien a una curva de Michaelis-Menten para $t_m = 8$

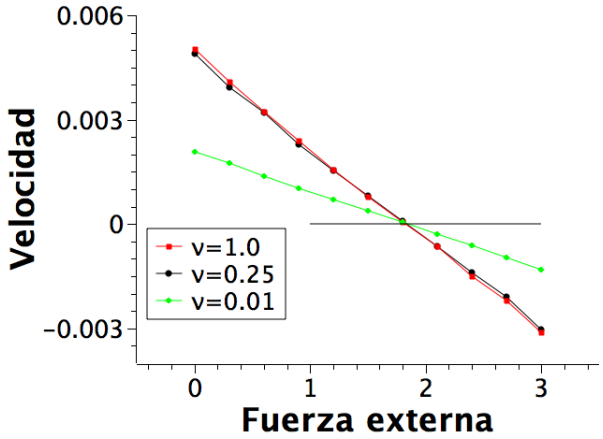


Figura 24: Velocidad frente a carga en el modelo *Ratchet*.

En este caso la fuerza de detención $F_d(\nu = 1{,}0) \approx 1{,}8 = 37{,}6\,pN$, coincide aproximadamente con las obtenidas en el modelo *Skew Gaussian*, si bien en este caso resulta destacable la convergencia de F_d para distintos valores del parámetro ν. Esta circunstancia, junto con la pequeña variación relativa de velocidades frente a ν en ambos modelos, apoya la hipótesis de una dinámica en -o cerca de- las concentraciones de saturación del motor; así mismo este hecho motivará en parte la posterior discusión sobre la escala de tiempo *biológica* frente a *simulada*.

6.4. Composición de la cadena

En este apartado se estudiará la influencia que la composición (A-T/C-G) tiene sobre el movimiento del motor, en particular sobre su velocidad (equivalentemente eficiencia); en estas simulacio-

nes se emplearán los valores óptimos de los parámetros hallados en **6.1** y **6.2**, variando únicamente el substrato sobre el que se desplaza la proteína.

En primer lugar se presenta una muestra de las trayectorias halladas para una monocadena de *adenina-timina* (curvas en color negro), *guanina-citosina* (curvas en color rojo), y una cadena mixta construida mediante la alternancia de *A-T/G-C* en grupos de *10 bp* (color verde). También se incluye la forma de una burbuja típica para cada una de estas composiciones en una cadena de *100 bp* (la configuración de la cadena ha sido promediada en el tiempo para eliminar fluctuaciones térmicas); para la dinámica *Skew Gaussian:*

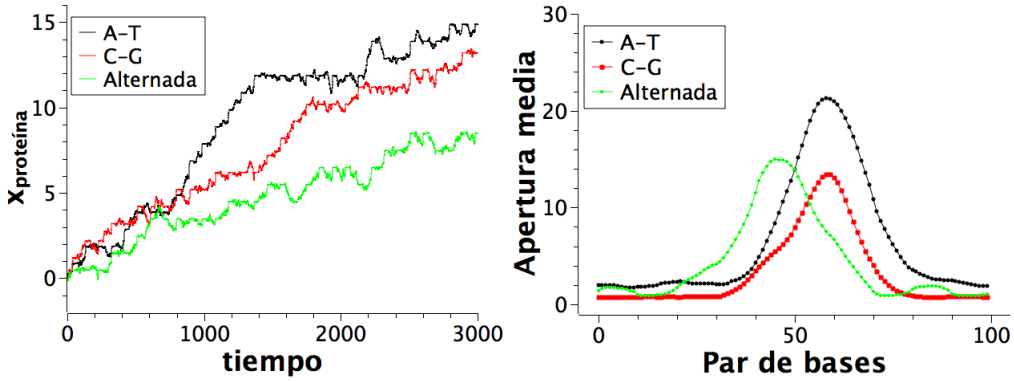


Figura 25: Ejemplo de trayectoria y formación de burbujas para distintas cadenas de ADN en el modelo *Skew Gaussian.*

Y en el caso *Ratchet:*

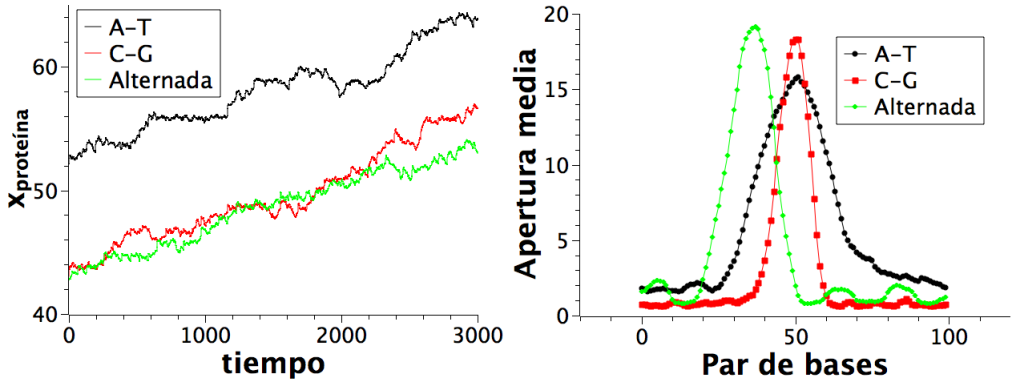


Figura 26: Ejemplo de trayectoria y formación de burbujas para distintas cadenas de ADN en el modelo *Ratchet.*

En ambos modelos se observa el comportamiento esperado dada la diferencia química de las bases: las bases *A-T*, enlazadas mediante dos puentes de hidrógeno, tiene más facilidad para abrirse y dar lugar a burbujas más anchas, lo cual aumenta el acoplo con el motor e incrementa su velocidad respecto de la cadena *G-C*, con bases enlazadas por tres puentes de hidrógeno.

También se llevan a cabo algunas simulaciones sobre una cadena de *605 bp*, codificada con el gen *han A* (relacionado con la diferenciación celular en cianobacterias). La siguiente figura muestra la configuración instantánea de sus pares de bases durante una simulación

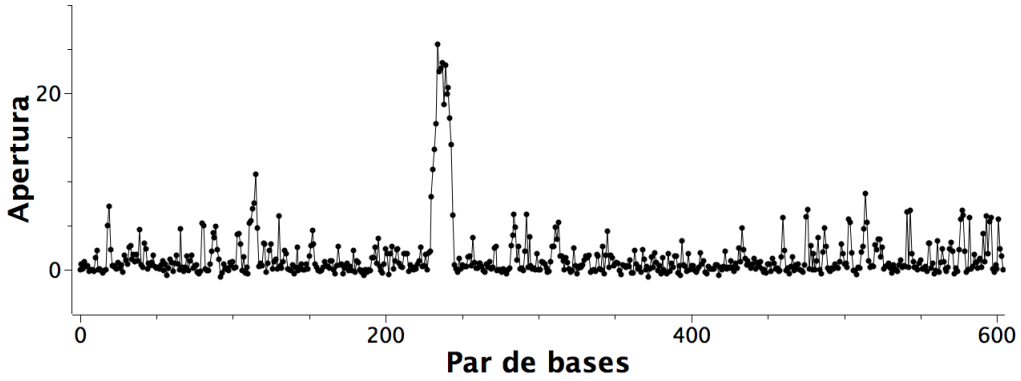


Figura 27: Diagrama de apertura de bases del gen *han A* durante el paso de la proteína.

donde destaca la diferencia de apertura entre las burbujas térmicas y la creada por la proteína.

Finalmente, el siguiente cuadro resume las diferencias de velocidad (eficiencia) obtenidas para cada modelo optimizado y composición; la estimación del error se ha obtenido a partir de una serie de varias simulaciones:

	Velocidad$\times 10^{-5}$ {relativa a A-T}	
	Skew gaussian	*Ratchet*
Monocadena de adenina (A)	$385 \pm 3\,\{1\}$	$531 \pm 9\,\{1\}$
Monocadena de guanina (G)	$365 \pm 3\,\{0{,}948\}$	$433 \pm 2\{0{,}815\}$
Cadena alterna A/G (cada 10 b-p)	$363 \pm 4\,\{0{,}943\}$	$470 \pm 7\{0{,}885\}$
Gen *han A* (58 % A-T)	$359 \pm 5\,\{0{,}932\}$	$487 \pm 3\{0{,}917\}$

Cuadro 1: Comparativa de la velocidad media para varias composiciones de la cadena. Los parámetros de cada modelo son los calculados como óptimos anteriormente.

Se destaca la tendencia a una mayor velocidad en las regiones ricas en A-T, tal como cabía esperar

7. Discusión y conclusiones

A lo largo de las secciones anteriores se han expuesto algunas de las características principales de los motores brownianos, su ubicuidad en la biología celular y, en particular su conveniencia como modelo dinámico para algunas proteínas encargadas de la manipulación del material genético; ha sido un objetivo primordial mostrar las prometedoras capacidades del modelado en la mesoescala para reproducir satisfactoriamente el comportamiento cualitativo de la física biomolecular. Sin embargo, la escala temporal de los procesos biológicos estudiados ($\sim$ segundo) es todavía muy superior a los rangos accesibles computacionalmente (ver figura 1), de manera que los procesos dependientes del tiempo resultan imposibles de comparar directamente con resultados experimentales. No obstante esto, el buen acuerdo entre la eficiencia de las simulaciones y los experimentos abre la posibilidad de explotar los dos mecanismos incluidos en el modelo para introducir una escala temporal superior: la dependencia $\nu([ATP])$ (esto es, la frecuencia con la que una molécula de ATP colisiona con -y es capturada por- la proteína), y el *tiempo muerto*, que puede incluir cualesquiera procesos biológicos que no interaccionen con la dinámica del motor (*e.g* síntesis de material genético).

Resulta llamativa la aproximación a los valores experimentales de las fuerzas de detención encontradas para ambos modelos (Figuras 22 y 24), especialmente en un contexto de poca fiabilidad

respecto de los procesos dependientes del tiempo, tal y como se acaba de discutir; actualmente el mecanismo preciso por el que la fuerza externa influye en la dinámica del motor es desconocida, pero entre las alternativas se encuentran el descenso de la tasa de adsorción de ATP (incremento de t_{on}), y la reducción de la tasa de salto por molécula de «combustible» consumida (descenso de la *eficiencia*). Si bien existen argumentos experimentales para considerar la combinación de ambos fenómenos [7], estas simulaciones apoyan (al menos en el sentido de «suficiente» para la helicasa/ARN-polimerasa) el segundo.

Otro aspecto limitante de las simulaciones ha sido la velocidad de relajación de la cadena en relación a su longitud para el modelo PBD modificado (*i.e* incluyendo una barrera de potencial que capture la tendencia de las bases nitrogenadas del ADN a saturar sus enlaces de hidrógeno libres con el solvente); las simulaciones llevadas a cabo con esta modificación mostraban una desnaturalización completa de la cadena. En este sentido, se propone reducir el tiempo de exposición a la proteína de burbujas previamente abiertas mediante el aumento de longitud de la cadena, y la reducción de velocidad del motor: para una cadena $100\,bp$, la influencia de la proteína (ver diagramas de la burbuja en las figuras 25 y 26) se extiende a lo largo de $\sim 50\,bp$. De este modo, y dado que las simulaciones se llevan a cabo con *condiciones periódicas de contorno* a fin de poder simular (en principio) dinámicas arbitrariamente largas, se espera que las regiones de la cadena abiertas hayan termalizado en el tiempo que le cuesta al motor recorrer $\sim 50\,bp$. Para velocidades típicas $\sim 0{,}005 - 0{,}01$ esto implica tiempos $\sim 10^4 - 5{\cdot}10^3\,u_t$. En la siguiente figura se ha representado el área bajo una burbuja frente al tiempo con barrera de solvatación (izquierda) y sin ella (derecha);

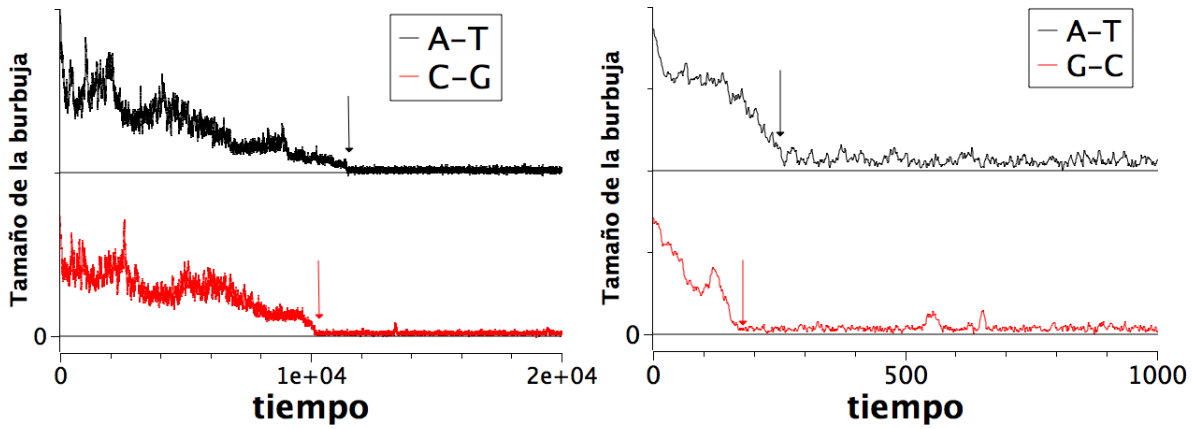


Figura 28: Termalización de una burbuja en un sistema con barrera de solvatación, $G = 3$ (izda) y sin ella, $G = 0$ (dcha).

Como se puede observar para el caso con barrera, el tiempo de termalización de una única burbuja aislada (sin un motor aledaño que tienda a abrir la cadena) resulta $\gtrsim$ que la estimación descrita arriba, de forma que la cadena todavía «sentirá» la influencia de la proteína cuando esta regrese cíclicamente; esto aumenta sucesivamente la apertura de la cadena, que resulta rápidamente desnaturalizada. A pesar de que esta es la razón por la que las simulaciones se han llevado a cabo sin introducir la barrera de solvatación en el modelo PBD, no debe descartarse como «sinsentido» antes de examinar detenidamente el fenómeno, ya que en particular la función de replicación del ADN supone la apertura de grandes burbujas.

Como conclusión final, cabe destacar el valor de los modelos en la escala mesoscópica como punto de partida a la hora de aproximase a nuevos problemas de los que se espera un alto grado de emergencia, en la medida en la que proporcionan una buena relación velocidad/precisión; sin

embargo, su uso en la búsqueda de fenómenos a escalas inferiores se encuentra restringido debido a la supresión de grados de libertad microscópicos en la construcción de variables y parámetros efectivos. Un último apunte a propósito del problema biológico a este respecto es la imposibilidad de distinguir el patrón de movimiento de las dinámicas *power stroke* y *ratchet* cuando estas actúan a temperatura finita, a pesar de que los mecanismos microscópicos que conducen a una u otra sean radicalmente diferentes.

Referencias

[1] Debashish Chowdury, *Stochastic mechano-chemical kinetics of molecular motor: a multidisciplinary enterprise from a physicist's perspective*, Phys. Rep. **529**, 1-197, (2013)

[2] Frank Jülicher, *Statistical physics of active processes in cells*, Physica A **369**, 187, (2006)

[3] R. Dean Astumian y Peter Hänggi, *Brownian Motors*, Physics Today, 34, (Noviembre de 2002)

[4] Elio A. Abbondanzieri, William J. Greenleaf, Joshua W. Shaevitz, Robert Landick y Steven M.Block, *Direct observation of base-pair stepping by ARN polymerase*, Nature **438**, 461, (2005)

[5] José A. Morin, Francisco J. Cao, José M. Lázaro, J. Ricardo Arias-Gonzalez, José M. Valpuesta, José L. Carrascosa, Margarita Salas Y Borja Ibarra, *Active ADN unwinding dynamics during processive ADN replication*, PNAS **109**, (2012)

[6] Jason A. Wagoner y Ken A. Dill, *Molecular Motors: Power Strokes Outperform Brownian Ratchets*, J. Phys. Chem B, (2016); DOI: 10.102/acs.jpcb.6b02776

[7] Koen Visscher, Mark J. Schnitzer y Steven M. Block, *Single kinesin studied with a molecular force clamp*, Nature **400**, 185, (1999)

[8] Richard P. Feynman, Robert B. Leighton, Matthew Sands, *Lectures on Physics volume I,* versión en español: *Física Volumen I: Mecánica, radiación y calor,* Addison-Wesley Longman, 46-2, (1998)

[9] Juan M.R Parrondo, Pep Español, *Criticism of Feynman's analysis of the ratchet as an engine*, American Journal of Physics **64**, 1125, (1996)

[10] Marcelo O. Magnasco, *Forced Thermal Ratchets*, Phys. Rev. Lett **71**, 1477-1481, (1993)

[11] R. Dean Astumian, *Thermodynamics and Kinetics of a Brownian Motor*, Science **276**, 917-922, (1997)

[12] Kim Sneppen y Givanni Zocchi, *Physics in molecular biology*, Cambridge University Press, (2005)

[13] Alberts, Bray, Hopkin, Johnson, Lewis, Raff, Roberts, Walter, *Essential cell biology*, Garland Science, (2010)

[14] Michel Peyrard, *Nonlinear dynamics and statistical physic of ADN*, Nonlinearity **17**, R1-R14, (2004)

[15] R. Tapia-Rojo, D. Prada-Gracia, J.J Mazo y F. Falo, *Mesoscopic model for free-energy-landscape analysis of ADN sequences*, Phys. Rev. E **86**, 021908, (2012)

[16] R. Tapia-Rojo, J.J. Mazo y F. Falo, *Thermal and mechanical properties of a ADN model with solvation barrier*, Phys. Rev. E **82**, 031916, (2010)

[17] N.J. Carter y R.A. Cross, *Mechanics of the kinesin step*, Nature **435**, 308, (2005)

[18] Aleksandr Noy, *Handbook of molecular force spectroscopy,* Springer, (2008)

[19] William H. Press, Saul A. Teukolsky, William T. Vetterling, y Brian P. Flannery, *Numerical recipes in C*, Cambridge University Press, (2002)

[20] K.F. Riley, M.P. Hobson, S.J. Bence, *Mathematical methods for physics and engineering*, Cambridge University Press, (2006)

[21] H.S Greenside y E. Helfand, *Numerical Integration of Stochastic Differential Equations-II*, The Bell System Technical Journal **60**, 1927-1940, (1981)

Apéndices

A. Ecuación de Langevin para el movimiento browniano

El movimiento browniano fue descrito por primera vez por el médico neerlandés Jan Ingenhousz en 1785 al percatarse del movimiento de partículas de carbón suspensas en alcohol; posteriormente fue redescubierto (y popularizado) por el biólogo escocés Robert Brown en 1827, al estudiar el movimiento aleatorio de diminutas partículas de polen en equilibrio térmico con un baño de agua. Fue descrito teóricamente por Albert Einstein y Paul Langevin a comienzos del siglo XX a través de dos aproximaciones radicalmente diferentes: estadísticamente el primero, dinámicamente el segundo. El estudio de Langevin culminó con una ecuación diferencial estocástica de segundo orden, la «Ecuación de Langevin», que describe el movimiento browniano:

$$m\ddot{\mathbf{r}} = -\overrightarrow{\nabla} V - m\eta\dot{\mathbf{r}} + \xi(t)$$

En esta ecuación se reconoce la Segunda Ley de Newton (primer término de la derecha), un término disipativo directamente proporcional a la velocidad e inversamente al tiempo característico de relajación de las fluctuaciones térmicas, $\tau = 1/\eta$; y un término estocástico representado mediante la variable aleatoria ξ, correlacionada en el tiempo:

$$< \xi(t)\xi(t') >= \xi_0^2 \cdot \frac{1}{\sqrt{\pi}\,\tau} e^{-\frac{(t-t')^2}{\tau^2}}$$

La velocidad de relajación térmica es muchos órdenes de magnitud superior a la velocidad de la partícula, por lo que podemos tomar $\tau \to 0$, y escribir:

$$< \xi(t)\xi(t') >= \xi_0^2 \cdot \delta(t - t')$$

Cabe mencionar que en el equilibrio térmico, una pequeña fluctuación mecánica es indistinguible de la agitación térmica, *i.e* para garantizar el segundo principio de la termodinámica (que el baño térmico no aumente la energía cinética de las partículas que contiene *en equilibrio*), debe haber una relación entre el acoplo de la agitación térmica ξ_0, y la constante de disipación η. Efectivamente, puede demostrarse el *teorema de fluctuación-disipación*:

$$\xi_0^2 = 2m\eta k_B T$$

B. Estimación del orden de los parámetros t_{on} y t_{off}

Dado el alto número de parámetros en el modelo de interacción y su amplio dominio, conviene calcular algunas cotas antes de iniciar la exploración del espacio de parámetros.

Para ello se asume en particular que el movimiento browniano de la fase difusiva es libre (*i.e* $V_{off} \approx cte$):

$$V_{off}^{lineal} = \lim_{x_p \to 0} V_{off}(x_p) = -\frac{B}{\sigma\sqrt{\pi}}$$

Teniendo en cuenta que la periodicidad espacial del problema es la unidad, factores de asimetría $\tilde{}$ *0.1* dividirán el intervalo espacial en regiones $L = 1 = 0{,}1 + 0{,}9$; por lo tanto, se busca el error cometido al linealizar el potencial en distancias $x \sim 0{,}9$:

$$\Delta = \frac{V_{off}(x = 0{,}9) - V_{off}^{lineal}}{V_{off}^{lineal}} = 1 - e^{\frac{-0{,}9}{\sigma^2}}$$

Y para un error $\Delta \lesssim 10\,\%$:

$$1 - e^{\frac{-0{,}9}{\sigma^2}} \lesssim 0{,}1 \rightarrow \sigma \gtrsim 2{,}9$$

Una proteína de tamaño $\tilde{}$ 3 *bp* es razonable, de forma que se justifica la aproximación.

Tiempo $t_{\mathbf{off}}$

El tiempo de difusión de la partícula debe ser tal que le permita explorar el entorno del siguiente par de bases, pero no regresar a la posición previa. Por tanto, teniendo en cuenta:

$$< x^2 >= 2D\, t_{off} = 2\frac{k_B T}{m_p \eta_p}\, t_{off} \quad \text{(donde se ha utilizado la relación de Einstein)}$$

Para temperatura ambiente $T = 25^{\text{o}}\text{C}$, $k_b T = 0{,}6 \rightarrow t_{off} \backsimeq 250 < x^2 >$ (la masa de la proteína y su constante de disipación han sido definidas previamente).

Luego:

$$0{,}1 < x < 0{,}9 \rightarrow 25 < t_{off} < 250$$

Tiempo $t_{\mathbf{on}}$

La única limitación en el intervalo temporal de la fase asimétrica es que debe ser suficientemente largo como para permitir a la partícula alcanzar el pozo de potencial antes de iniciar un nuevo ciclo. Por tanto, la situación más desfavorable (*i.e* que requiere de un t_{On} mayor) es aquella en la que la partícula se encuentra en una posición $x_{max} = 0$ de potencial máximo, $V_{max}^{on} = 0$, y debe transitar hacia otra $x_{min} = 1$ de potencial mínimo $V_{min}^{on} = -\frac{B}{\sigma\sqrt{\pi}}$; la fuerza media que sufrirá la partícula será:

$$\bar{F} = \frac{-\int_0^1 \frac{dV_{on}}{dx} dx}{\int_0^1 dx} = \frac{-\Delta V_{on}}{1} = \frac{B}{\sigma\sqrt{\pi}}$$

Y suponiendo que en el régimen sobreamortiguado la velocidad de la proteína será constante e igual a la velocidad límite $v_{limite} = \frac{F}{m_p \eta_p}$:

$$t_{on} < \frac{1}{v_{limite}} = \sqrt{\pi}\,\frac{m_p \eta_p \sigma}{B} \simeq 531{,}7\,\frac{\sigma}{B}$$

Para valores de la anchura $\sigma \sim 3$ y constantes $B > 10$, será suficiente con explorar $t_{on} < 160$

C. Algoritmo para la integración de ecuaciones diferenciales estocásticas

La simulación de los modelos expuestos en el texto implican la resolución de grandes sistemas de *ecuaciones diferenciales estocásticas*, donde destaca la presencia de un término aleatorio (físicamente justificado por el ruido térmico). La manera de interpretar la «solución» de tales ecuaciones es la producción de curvas que sean estadísticamente significativas a orden k, esto es trayectorias x_t tales que todos los momentos de la distribución se puedan aproximar arbitrariamente:

$$< x_t^q >=< x(t)^q_{exacto} > +O(\Delta t^k)$$

La forma general de los sistemas de ecuaciones a resolver en este trabajo han sido:

$$\begin{cases} \frac{dy_i}{dt} = \frac{p_i}{m} \mid_{i=1,...N} \\ \frac{dp_i}{dt} = -\frac{\partial(V(\{y_j\})+W(\{y_j,y_{j-1}\}))}{\partial y_i} - \frac{\partial V_{int}(\{y_j\},x_p)}{\partial y_i} - \eta p_i - \xi(t) \mid_{i=1,...N} \\ \frac{dx_p}{dt} = \frac{p_p}{m_p} \\ \frac{dp_p}{dt} = -\frac{\partial V_{int}(\{y_j\},x_p)}{\partial x_p} - \eta_p p_p - \xi_p(t) \end{cases}$$

Siendo N el número de pares de base de una cadena.

El algoritmo empleado para resolver estos sistemas ha sido el *método de Runge-Kutta* de tercer orden, que requiere de cuatro pasos y la generación de dos números aleatorios para integrar una variable en un intervalo δt:

Se define la *matriz Y*

$$Y_{ik} = \sum_{j=1}^{2} \lambda_{ij} Z^{jk}$$

donde Z es un número aleatorio distribuido normalmente con media uno y varianza unidad, el índice i toma tantos valores como pasos utiliza el algoritmo (en este caso cuatro), j toma tantos valores como números aleatorios calcula el algoritmo (en este caso, dos) y k toma tantos valores como variables estocásticas es necesario integrar (en este caso, dado que las ecuaciones de las posiciones son deterministas, este índice tomará *N+1* valores, siendo N el número de bases de la cadena).

Una vez construida esta cantidad, el algoritmo para integrar la trayectoria de una partícula genérica sometida a fluctuaciones térmicas y un potencial externo de la forma

$$\begin{cases} \dot{x_k} = \frac{p_k}{m} \\ \dot{p_k} = -\frac{\partial V_k(x_k)}{\partial x_k} - \eta p_k + \xi_k(t) \qquad \text{donde } < \xi_k(t)\xi_k(t') >= \xi_0^2 \delta(t-t') \end{cases}$$

será [21]:

$$x_k(t) = x_k(t_0) + \delta t(A_1 g_{1k}^{(x)} + A_2 g_{2k}^{(x)} + A_3 g_{3k}^{(x)} + A_4 g_{4k}^{(x)})$$

$$p_k(t) = p_k(t_0) + \delta t(A_1 g_{1k}^{(p)} + A_2 g_{2k}^{(p)} + A_3 g_{3k}^{(p)} + A_4 g_{4k}^{(p)}) + \sqrt{\delta t \xi_0} Y_{0k}$$

donde:

$$g_{lk}^{(x)} = \frac{1}{m}\left(p_k(t_0) + \delta t \sum_{i=1}^{l-1} \beta_l^i g_{ik}^{(p)} + \sqrt{\delta t \xi_0} Y_{lk}\right) \qquad i = 1, ..., l-1$$

$$g_{lk}^{(p)} = -\frac{\partial V_k(x_k(t_0) + \delta t \sum_{i=1}^{l-1} \beta_l^i g_{ik}^{(x)})}{\partial x_k} - \eta \cdot \left(p_k(t_0) + \delta t \sum_{i=1}^{l-1} \beta_l^i g_{ik}^{(p)} + \sqrt{\delta t \xi_0} Y_{lk} \right) \qquad i = 1, ..., l-1$$

Las cantidades λ, A, y β para este algoritmo concreto están recogidas en la siguiente tabla:

Table II—Parameters for a $3_O 4_S 2_G$ algorithm appropriate to vector SDEs

A_1	0.0	A_2	0.644468
A_3	0.194450	A_4	0.161082
β_{21}	0.516719	β_{31}	−0.397300
β_{32}	0.427690	β_{41}	−1.587731
β_{42}	1.417263	β_{43}	1.170469
λ_{01}	1.0	λ_{02}	0.0
λ_{11}	0.0	λ_{12}	0.271608
or			
λ_{11}	−0.567253	λ_{12}	0.0
λ_{21}	0.516719	λ_{22}	0.499720
λ_{31}	0.030390	λ_{32}	−0.171658
λ_{41}	1.0	λ_{42}	0.0

D. Códigos

Programas escritos en lenguaje C, utilizados para la realización de este trabajo. Además de los códigos, algunos programan utilizan datos provenientes de archivos externos; si es el caso, un superíndice junto al nombre del programa indica la necesidad de un fichero de parámetros (P), secuencia de bases nitrogenadas (S) o algún otro (*). Los programas aparecen enumerados de acuerdo con el orden en el que han sido necesarios en el texto:

1. ***FlashingRatchet_interpol*** este programa resuelve la expresión de interpolación presentada para la dinámica *flashing ratchet* en la sección *2*
2. ***FlashingRatchet_sim*** $^{(P)}$ simulación de la dinámica *flashing ratchet*.
3. ***DebugPotencial*** este programa calcula y dibuja los potenciales *ON/OFF* sobre una monocadena *A-T* de *10 bp,* permitiendo ajustar la apertura de cada base individualmente, o cambiar los parámetros de interacción con la proteína.
4. ***HelicasaMotor_SkwGaus*** $^{(P,\ S)}$ simulación de la dinámica *Skew Gaussian* con el parámetro tiempo t_{on} determinado.
5. ***HelicasaMotor_Ratchet*** $^{(P,\ S)}$simulación de la dinámica *Ratchet* con el parámetro t_{on} determinado.
6. ***HelicasaMotor_SkwGaus_ATP*** $^{(P,\ S)}$simulación de la dinámica *Skew Gaussian* con el parámetro t_{on} distribuido estocásticamente.
7. ***HelicasaMotor_Ratchet_ATP*** $^{(P,\ S)}$simulación de la dinámica *Ratchet* con el parámetro t_{on} distribuido estocásticamente.
8. ***PintaCinematica*** $^{(*)}$ crea una animación de la cinemática de la proteína y la cadena de ADN sobre la que se mueve. Requiere datos que se generan (opcionalmente) por los programas *(4)*, *(5)*, *(6)*, o *(7).*
9. ***PintaPotencial*** $^{(*)}$ crea una animación de la energía potencial instantánea de la proteína sobre la cadena de ADN. Requiere datos que se generan (opcionalmente) por los programas *(4)*, *(5)*, *(6)*, o *(7).*

Los programas *(4)*, *(5)*, *(6)*, y *(7)* son una adaptación, ampliación y traducción a lenguaje C del programa original escrito en *Fortran* por F. Falo y R.Tapia-Rojo; el código de todos ellos es similar salvo pequeñas modificaciones, de forma que por motivos de espacio únicamente se incluye *(4)*

(1)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.14159
double Prob_n(double D,double t_off,double a,int n);

int main()
{
///Definicion de variables operativas
int i,k,N,N_integral;
char name[256];
FILE *punte;



///Definicion de constantes del procedimiento
double T,eps,masa,asimetria,D,t_on,t_off_ini,t_off_final,delta_t_off;



///Definicion de variables del procedimiento
double t_off,avance_medio,velocidad;



///Asignacion de constantes
    T=0.1;
    eps=20.0;
    masa=15.0;
    asimetria=0.1;
    D=T/(eps*masa); //Relacion de Einstein: D=kT/(mu*masa)
    t_on=50;
    t_off_ini=1;
    t_off_final=1000;
    delta_t_off=1;



///Inicializacion de variables operativas
    N=(int)((t_off_final-t_off_ini)/delta_t_off);
    N_integral=10;
    sprintf(name,"InterpolacionFR_T=%f_tON=%f",T,t_on);
    punte=fopen(name,"w");
    fprintf(punte,"tOFF\t<v>\n");



///Inicializacion de variables del procedimiento
    t_off=t_off_ini;
    avance_medio=0;




///Procedimiento de interpolacion pot. Flashing Ratchet:
    for(k=0;k<N;k++){
        avance_medio=0;

        for(i=-N_integral;i<N_integral;i++)
        avance_medio+=i*Prob_n(D,t_off,asimetria,i);

        velocidad=avance_medio/(t_on+t_off);
        fprintf(punte,"%f\t%f\n",t_off,velocidad);

        t_off+=delta_t_off; }

```

```
67
68  ///Fin del programa y cierre de archivos
69  printf("\n\nPROGRAMA FINALIZADO\n");
70      fclose(punte);
71      return 0;
72  }
73
74
75
76  double Prob_n(double D,double t_off,double a,int n){
77      double out=0;
78      double factor=1/sqrt(4*D*t_off);
79      out=0.5*(erf(factor*(a+n))-erf(factor*(a-1+n)));
80      return out; }
```

(2)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#define PI 3.1415926536
#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)



double Calc_fuerza(double x_prot,double d2,double altura,int estadoATP,double fuerza_externa);
void Calc_Output(double p_prot,double x_prot,double x_prot_previo,double m_prot, double T, double TON,double TOFF,double *Energia_cinetica,double *velocidad_media,double t_term, double t_final, double dtau,int N,int Nterm,int indice_simulacion,FILE *Output,double f_externa,int n);

float ran1(long *idum);
void Calcula_EP(double x_prot,double d2, double altura,int puntos,int estadoATP, FILE *Epotencial);
void CalculaTrayectoria_prot(double x_prot,double p_prot,double m_prot, double t, int estadoATP, FILE *Trayectoria_prot);
double dV_ratchett(double X, double d2,double altura,double epsilon);
double V_ratchett(double X,double d2, double altura,double epsilon,double suelo);




int main()
{

//Definicion de constantes y variables operativas:
    FILE *datosp;
    unsigned int i,n,indTemp,N,Nterm,N_on,N_off,contadorON,contadorOFF,ind_TON,ind_TOFF,
        ind_Fexterna,indice_simulacion;
    int selector_EP,selector_Output,selector_Tray,contador_muestra;
    long idum;
    char basura[256],etiqueta[256],name_leyenda[256],base;

//Inicializacion del generador de numeros aleatorios
    idum=-(long)time(NULL);

//Definicion de constantes del algoritmo:
    const double lambda11=-0.567253;
    const double lambda21=0.516719;
    const double lambda22=0.499720;
    const double lambda31=0.030390;
    const double lambda32=-0.171658;

    const double beta21=0.516719;
    const double beta31=-0.397300;
    const double beta32=0.427690;
    const double beta41=-1.587731;
    const double beta42=1.417263;
    const double beta43=1.170469;

    const double A2=0.644468;
    const double A3=0.194450;
    const double A4=0.161082;
```

```
60
61  //Definicion de variables del algoritmo
62      double Y[5];
63      double gx[4];
64      double gp[4];
65      double random1;
66      double random2;
67
68  //Definicion de constantes globales del modelo:
69      double dtau,T,T0,incrementoT,t_termalizacion,t_final,TON0,TOFF0,TON,TOFF,incremento_TON,
            incremento_TOFF,asimetria,altura,fuerza_externa0,incr_fuerza_externa,fuerza_externa;
70      int muestra,PuntosT,Puntos_TON,Puntos_TOFF,PuntosFuerza_externa;
71
72  //Definicion de constantes locales del modelo
73      double eps_prot,m_prot;
74
75  //Definicion de variables del modelo:
76      double x_prot,p_prot,t,aux_prot; //[x,p,t];
77      double x_prot_abs,p_prot_abs,t_abs;
78      double fuerza_prot;
79      int estadoATP; //0-> no ATP / 1-> si ATP
80
81  //Definicion de observables:
82      double Energia_cinetica,*config_promedio,velocidad_media,x_prot_previo;
83      int eficacia;
84
85      FILE *EP_prot;
86      char name_ep[256];
87
88      FILE *Trayectoria_prot;
89      char name_tray[256];
90
91      FILE *Output;
92      char name_out[256];
93
94      FILE *Randomness;
95      char name_randomness[256];
96
97
98   //Lectura de dtos :
99      datosp=fopen("datospG.inp","r");
100
101     fscanf(datosp,"%␣%s\n",etiqueta,basura);
102
103     fscanf(datosp,"%␣%s\n",basura,basura);
104     fscanf(datosp,"%f␣%s\n",&dtau,basura);
105     fscanf(datosp,"%f␣%s␣%f␣%␣%s\n",&T0,basura,&incrementoT,basura,&PuntosT,basura);
106     fscanf(datosp,"%f␣%s␣%f␣%s\n",&t_termalizacion,basura,&t_final,basura);
107     fscanf(datosp,"%␣%s\n",&muestra,basura);
108
109     fscanf(datosp,"%s␣%s␣%s␣%s\n",basura,basura,basura,basura);
110     fscanf(datosp,"%s␣%s␣%s␣%s\n",basura,basura,basura,basura);
111     fscanf(datosp,"%s␣%s␣%s␣%s\n",basura,basura,basura,basura);
112     fscanf(datosp,"%s␣%s\n",basura,basura); //0= no; 1=si
113     fscanf(datosp,"%f␣%s␣%f␣%s␣%␣%s\n",&TOFF0,basura,&incremento_TOFF,basura,&Puntos_TOFF,
            basura);
114     fscanf(datosp,"%f␣%s␣%f␣%s␣%␣%s\n",&fuerza_externa0,basura,&incr_fuerza_externa,basura
            ,&PuntosFuerza_externa,basura);
115
116     fscanf(datosp,"%␣%s\n",&selector_Output,basura); //0= no; 1=si
117     fscanf(datosp,"%s␣%s\n",basura,basura); //0= no; 1=si
118     fscanf(datosp,"%␣%s\n",&selector_EP,basura); //0= no; 1=si
119     fscanf(datosp,"%s␣%s\n",basura,basura); //0= no; 1=si
120     fscanf(datosp,"%␣%s\n",&selector_Tray,basura); //0= no; 1=si
121     fscanf(datosp,"%s␣%s\n",basura,basura);
122
123     fscanf(datosp,"%f␣%s␣%f␣%s␣%␣%s\n",&TON0,basura,&incremento_TON,basura,&Puntos_TON,
            basura);
```

```
124       fscanf(datosp,"%s %s\n",basura,basura);
125       fscanf(datosp,"%lf %s\n",&asimetria,basura);
126       fscanf(datosp,"%lf %s\n",&altura,basura);
127       fscanf(datosp,"%s %s %s %s\n",basura,basura,basura,basura);
128       fscanf(datosp,"%s %s\n",basura,basura);
129
130
131       fclose(datosp);
132
133
134    //Asignacion de constantes locales:
135       m_prot=15.0;
136       eps_prot=20.0;
137
138
139
140    //Bucle de parametros.
141           if(selector_Output==1){
142               sprintf(name_out,"Output_%s",etiqueta);
143               Output=fopen(name_out,"w");
144               fprintf(Output,"IndiceSim\tTemperatura\ttON\ttOFF\tF_externa\tEnergiaCinetica\
                      tdof*kT/2\tEC_normalizada\t<Eficacia>\t<Velocidad>\n");
145           }
146
147    T=T0;
148    TON=TON0;
149    TOFF=TOFF0;
150    fuerza_externa=fuerza_externa0;
151     for(indTemp=0;indTemp<PuntosT;indTemp++){
152                         for(ind_TON=0;ind_TON<Puntos_TON;ind_TON++){
153                                                 for(ind_TOFF=0;ind_TOFF<Puntos_TOFF;ind_TOFF++){
154                                                         for(ind_Fexterna=0;ind_Fexterna<
                                                              PuntosFuerza_externa;ind_Fexterna++)
                                                              {
155
156     //Inicializacion de variables y observables
157     indice_simulacion=indTemp*Puntos_TON*Puntos_TOFF*PuntosFuerza_externa+ind_TON*Puntos_TOFF*
          PuntosFuerza_externa+ind_TOFF*PuntosFuerza_externa+ind_Fexterna;
158           Energia_cinetica=0;
159           velocidad_media=0;
160
161           t=0;
162           x_prot=0;
163           p_prot=0;
164           x_prot_previo=x_prot;
165           aux_prot=sqrt(m_prot*dtau*2*T*eps_prot);
166
167
168           if(selector_EP==1){
169               sprintf(name_ep,"EP_%s_%i",etiqueta,indice_simulacion);
170               EP_prot=fopen(name_ep,"w");
171               fprintf(EP_prot,"#t_term %lf\n#t_fnal %lf\n#dt %lf\n#Ncadena %i\n#muestra %i\n",
                      t_termalizacion,t_final,dtau,10,muestra);
172           }
173
174
175
176           if(selector_Tray==1){
177               sprintf(name_tray,"Trayectoria_%s_%i",etiqueta,indice_simulacion);
178               Trayectoria_prot=fopen(name_tray,"w");
179               fprintf(Trayectoria_prot,"Tiempo\tx_prot\tv_prot\testadoATP\n");
180           }
181
182
183           N=(int)(t_final/dtau);
184           Nterm=(int)(t_termalizacion/dtau);
185           N_on=(int)(TON/dtau);
186           contadorON=0;
```

```
187         N_off=(int)(TOFF/dtau);
188         contadorOFF=0;
189         estadoATP=0;
190         if(N_off==0)
191             estadoATP=1;
192         contador_muestra=0;
193
194
195 //TERMALIZACION Y DINAMICA
196         for(n=0;n<N;n++){
197             if(n%(N/50)==0) //Porcentaje!
198                 printf("%f\n",100.0*n/N);
199
200
201 //Comienza integracion en dtau:
202                 random1=ran1(&idum);
203                 random2=ran1(&idum);
204                     Y[0]=-sqrt(-2*log(random1))*cos(2*PI*random2);
205                     Y[1]=-lambda11*sqrt(-2*log(random1))*cos(2*PI*random2);
206                     Y[2]=-lambda21*sqrt(-2*log(random1))*cos(2*PI*random2)-lambda22*sqrt(-2*
                            log(random1))*sin(2*PI*random2);
207                     Y[3]=-lambda31*sqrt(-2*log(random1))*cos(2*PI*random2)-lambda32*sqrt(-2*
                            log(random1))*sin(2*PI*random2);
208                     Y[4]=-sqrt(-2*log(random1))*cos(2*PI*random2);
209                 //Produccion de la matriz Y
210
211
212                 x_prot_abs=x_prot;
213                 p_prot_abs=p_prot+aux_prot*Y[1];
214                 //t_abs=t;
215
216                 fuerza_prot=Calc_fuerza(x_prot_abs,asimetria,altura,estadoATP,fuerza_externa
                        );
217
218
219                 gx[0]=p_prot_abs/m_prot;
220                 gp[0]=fuerza_prot-eps_prot*p_prot_abs;
221                 //    g[0][2]=1;
222                 //PASO 1
223
224
225                 x_prot_abs=x_prot+dtau*beta21*gx[0];
226                 p_prot_abs=p_prot+dtau*beta21*gp[0]+aux_prot*Y[2];
227                 //t_abs=t+dtau*0.516719;
228
229                 fuerza_prot=Calc_fuerza(x_prot_abs,asimetria,altura,estadoATP,fuerza_externa
                        );
230
231                 gx[1]=p_prot_abs/m_prot;
232                 gp[1]=fuerza_prot-eps_prot*p_prot_abs;
233                 //    g[1][2]=1;
234                 //PASO 2
235
236
237                 x_prot_abs=x_prot+dtau*(beta31*gx[0]+beta32*gx[1]);
238                 p_prot_abs=p_prot+dtau*(beta31*gp[0]+beta32*gp[1])+aux_prot*Y[3];
239                 //t_abs=t+dtau*(-0.397300+0.427690);
240
241                 fuerza_prot=Calc_fuerza(x_prot_abs,asimetria,altura,estadoATP,fuerza_externa
                        );
242
243                 gx[2]=p_prot_abs/m_prot;
244                 gp[2]=fuerza_prot-eps_prot*p_prot_abs;
245                 //g[2][2]=1;
246                 //PASO 3
247
248
249                 x_prot_abs=x_prot+dtau*(beta41*gx[0]+beta42*gx[1]+beta43*gx[2]);
```

```
250                 p_prot_abs=p_prot+dtau*(beta41*gp[0]+beta42*gp[1]+beta43*gp[2])+aux_prot*Y
                        [4];
251                 //t_abs=t+dtau*(-1.587731+1.417263+1.170469);
252
253                 fuerza_prot=Calc_fuerza(x_prot_abs,asimetria,altura,estadoATP,fuerza_externa
                        );
254
255                 gx[3]=p_prot_abs/m_prot;
256                 gp[3]=fuerza_prot-eps_prot*p_prot_abs;
257                 //    g[3][2]=1;
258                 //PASO 4
259
260
261                 x_prot_previo=x_prot; //Memoria de x_prot
262                 x_prot=x_prot+dtau*(A2*gx[1]+A3*gx[2]+A4*gx[3]);
263                 p_prot=p_prot+dtau*(A2*gp[1]+A3*gp[2]+A4*gp[3])+aux_prot*Y[0];
264                 t+=dtau; //*(0.644468*g[1][2]+0.194450*g[2][2]+0.161082*g[3][2]);
265
266
267 ////////////////////////////////////////////////Aqui acaba la integracion en dtau.
268
269 //toma de medidas:
270             if(n>=Nterm){ //Termalizacion!
271                 contador_muestra++;
272
273                 if(selector_Output==1)
274                     Calc_Output(p_prot,x_prot,x_prot_previo,m_prot,T,TON,TOFF,&
                            Energia_cinetica,&velocidad_media,t_termalizacion,t_final,dtau,N,
                            Nterm,indice_simulacion,Output,fuerza_externa,n); //Observables
                            promediados no tienen sentido muestreados!
275
276                 if(contador_muestra==muestra){
277
278                     if(selector_Tray==1)
279                         CalculaTrayectoria_prot(x_prot,p_prot,m_prot,t,estadoATP,
                                Trayectoria_prot);
280
281                     if(selector_EP==1)
282                         Calcula_EP(x_prot,asimetria,altura,1000,estadoATP,EP_prot);
283
284                     contador_muestra=0;
285
286                 }
287
288             }
289
290
291             if(estadoATP==0)
292                 contadorOFF++;
293
294             if(estadoATP==1)
295                 contadorON++;
296
297             if(contadorON==N_on){
298                 estadoATP=0;
299                 contadorON=0;
300             }
301             if(contadorOFF==N_off){
302                 estadoATP=1;
303                 contadorOFF=0;
304             }
305         }//Fin del experimento l-esimo
306
307          if(selector_EP==1)
308             fclose(EP_prot);
309
310         if(selector_Tray==1)
311             fclose(Trayectoria_prot);
```

```
312
313
314             printf("%/%i\n\n",indice_simulacion+1,PuntosT*Puntos_TON*Puntos_TOFF*
                    PuntosFuerza_externa);
315
316                                                             fuerza_externa=fuerza_externa+
                                                                    incr_fuerza_externa;}
317
318                                                         fuerza_externa=fuerza_externa0;
319                                                         TOFF=TOFF*incremento_TOFF;}
320
321                                                 fuerza_externa=fuerza_externa0;
322                                                 TOFF=TOFF0;
323                                                 TON=TON*incremento_TON;}
324
325
326                         fuerza_externa=fuerza_externa0;
327                         TOFF=TOFF0;
328                         TON=TON0;
329                         T=T+incrementoT;}
330
331
332
333     //fin del programa.
334     printf("\n\n PROGRAMA FINALIZADO\n\n");
335
336     if(selector_Output==1)
337         fclose(Output);
338
339         return 0;
340     }
341
342
343     double Calc_fuerza(double x_prot,double d2,double altura,int estadoATP,double fuerza_externa
            ){
344         double fuerza_prot,suma=0;
345         int i,j;
346
347         if(estadoATP==1)
348             fuerza_prot=-dV_ratchett(x_prot,d2,altura,0.0);
349
350
351         if(estadoATP==0)
352             fuerza_prot=0;
353
354
355         return fuerza_prot+fuerza_externa;
356
357     }
358
359
360     void Calc_Output(double p_prot,double x_prot,double x_prot_previo,double m_prot, double T,
            double TON,double TOFF,double *Energia_cinetica,double *velocidad_media,double t_term,
            double t_final, double dtau,int N,int Nterm,int indice_simulacion,FILE *Output,double
            f_externa,int n){
361         int i;
362         double time;
363
364         if(n==Nterm){
365             *Energia_cinetica=0;
366             *velocidad_media=0;
367         }
368
369
370         *Energia_cinetica+=0.5*p_prot*p_prot/m_prot;
371         *velocidad_media+=(x_prot-x_prot_previo);
372
373
```

```
374
375     if(n==(N-1)){
376         time=dtau/(t_final-t_term);
377         fprintf(Output,"%i\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\n",indice_simulacion
                ,T,TON,TOFF,f_externa,*Energia_cinetica*time,0.5*T,*Energia_cinetica*time/(0.5*T
                ),*velocidad_media/(t_final-t_term)*(TON+TOFF),*velocidad_media/(t_final-t_term)
                );
378             }
379
380 }
381
382
383 float ran1(long *idum){
384     int j;
385     long k;
386     static long iy=0;
387     static long iv[NTAB];
388     float temp;
389     if (*idum <= 0 || !iy) {
390     if (-(*idum) < 1) *idum=1;
391     else *idum = -(*idum);
392
393     for (j=NTAB+7;j>=0;j--) {
394         k=(*idum)/IQ;
395         *idum=IA*(*idum-k*IQ)-IR*k;
396          if (*idum < 0)
397             *idum += IM;
398          if (j < NTAB)
399             iv[j] = *idum;
400         }
401         iy=iv[0];
402     }
403     k=(*idum)/IQ;
404     *idum=IA*(*idum-k*IQ)-IR*k;
405     if (*idum < 0) *idum += IM;
406     j=iy/NDIV;
407     iy=iv[j];
408     iv[j] = *idum;
409     if ((temp=AM*iy) > RNMX) return RNMX;
410     else return temp;
411 }
412
413
414
415 void Calcula_EP(double x_prot,double d2, double altura,int puntos,int estadoATP, FILE *
        Epotencial){
416     double XP=-5;
417     double dx=(double)10.0/puntos;
418     double V=0;
419     int i,j;
420
421
422     if(estadoATP==0){
423         for(i=0;i<puntos;i++){
424             fprintf(Epotencial,"%f\t%f\n",XP,0.0);
425             XP+=dx;
426
427         }//Aqui acaba de pintar EPcadena. Indice PAR.
428         fprintf(Epotencial,"\n\n");
429
430
431         fprintf(Epotencial,"%f\t%f\n",x_prot,0.0);
432
433         fprintf(Epotencial,"\n\n");
434     }
435
436
437     V=0;
```

```
438     XP=-5;
439     if(estadoATP==1){
440         for(i=0;i<puntos;i++){
441                 V=V_ratchett(XP,d2,altura,0.0,0.0);
442
443             fprintf(Epotencial,"%f\t%f\n",XP,V);
444             XP+=dx;
445         }//Aqui acaba de pintar EPcadena. Indice PAR.
446         fprintf(Epotencial,"\n\n");
447
448     V=0;
449
450         V=V_ratchett(x_prot,d2,altura,0.0,0.0);
451
452         fprintf(Epotencial,"%f\t%f\n",x_prot,V);
453
454         fprintf(Epotencial,"\n\n");
455     }
456
457 }
458
459
460
461 void CalculaTrayectoria_prot(double x_prot,double p_prot,double m_prot, double t, int
        estadoATP, FILE *Trayectoria_prot){
462     fprintf(Trayectoria_prot,"%f\t%f\t%f\t%i\n",t,x_prot,p_prot/m_prot,estadoATP);
463 }
464
465
466 double dV_ratchett(double X, double d2,double altura,double epsilon){
467
468     if(((int)X)<=0)
469         X=2+X-((int)X);
470
471
472     if((fabs(X)-abs((int)X))<epsilon)
473         return 0;
474
475     if(((int)X)==((int)(X-1+d2)))
476         return (-altura/d2);
477
478     if(((int)X)>((int)(X-1+d2)))
479         return (altura/(1-d2));
480
481
482 }
483 double V_ratchett(double X,double d2, double altura,double epsilon,double suelo){
484     int i;
485
486     if(((int)X)<=0)
487         X=2+X-((int)X);
488
489     if(dV_ratchett(X,d2,altura,epsilon)>0)
490         return ((X-(int)X)*dV_ratchett(X,d2,altura,epsilon))-suelo;
491
492     if(dV_ratchett(X,d2,altura,epsilon)<0)
493         return ((X-(int)(X+d2))*dV_ratchett(X,d2,altura,epsilon))-suelo;
494
495 }
```

(3)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.14159
#define Ncadena 10
#define constante 3


double distanciaProt_Base(double x_prot,double i);
void PintaPotencialDebug(double *x, double gamma, double B, double anch,int k,FILE *debug,
    FILE *debug_pipe);
double dV_ratchett(double X, double d2,double altura,double epsilon);
double V_ratchett(double X,double d2, double altura,double epsilon,double suelo);
double Exponencial_asim(double x_prot,double i,double anch,double anchON);
double dExponencial_asim(double x_prot,double i,double anch,double anchON);
double funcion_asim(double exp3_i,double dist_i,double anch,double alfa,double B);



int main()
{
    printf("\n\n\nESTUDIO␣DEL␣POTENCIAL:␣gausiano\nSeleccionar␣parametro␣e␣introducir␣su␣
        valor\nIntroducir␣el␣valor␣'-2'␣para␣salir␣del␣programa\n\n");
    int i,j,k,parametro,contador_iteracion;
    char basura;
    FILE *debug,*debug_pipe;
    double x_prot,x[Ncadena],Von[Ncadena],Voff[Ncadena],Vton,Vtoff,B,gamma,anch;

 //INICIALIZACION:
 debug_pipe=popen("gnuplot","w");
 debug=fopen("PotencialDebug_Prot","w");
 parametro=-1;
 contador_iteracion=0;
 for(i=0;i<Ncadena;i++)
    x[i]=0;
 x_prot=0.5*Ncadena;
 B=50;
 gamma=0.2;
 anch=3;

for(k=0;k<1;k++){
 for(i=0;i<1;i++){

//EXPOSICION DE PARAMETROS:
    for(j=0;j<Ncadena;j++)
        printf("(%i)␣x[%i]=␣%f\n",j,j,x[j]);

    printf("(%i)␣B=␣%f\n",Ncadena,B);
    printf("(%i)␣gamma=␣%f\n",Ncadena+1,gamma);
    printf("(%i)␣anch=␣%f\n",Ncadena+2,anch);
    printf("\n(%i)␣x[i]␣aleatorio\n",Ncadena+3);
    printf("(%i)␣x[i]␣infinito\n",Ncadena+4);
    printf("\n");

//SELECCION DE PARAMETROS:
    setbuf(stdin, NULL);
    scanf("%i%c",&parametro,&basura);

    if(parametro==-2) //PARA SALIR DEL PROGRAMA!
        k=1;

    if((parametro>=0) && (parametro<Ncadena)){
        setbuf(stdin, NULL);
        scanf("%f",&x[parametro]);
        i--;
    }
```

```
65
66      if(parametro>=Ncadena){
67          switch(parametro){
68              case Ncadena:
69                  setbuf(stdin, NULL);
70                  scanf("%lf",&B);
71                  break;
72              case Ncadena+1:
73                  setbuf(stdin, NULL);
74                  scanf("%lf",&gamma);
75                  break;
76              case Ncadena+2:
77                  setbuf(stdin, NULL);
78                  scanf("%lf",&anch);
79                  break;
80              case Ncadena+3:
81                  for(j=0;j<Ncadena;j++)
82                      x[j]=rand()/(1.0*RAND_MAX)*10-5;
83                  break;
84              case Ncadena+4:
85                  for(j=0;j<Ncadena;j++)
86                      x[j]=15;
87                  break;
88          }
89          i--;
90      }
91      parametro=-1;
92    }
93
94
95  //CALCUIO DEL POTENCIAL
96
97     PintaPotencialDebug(x,gamma,B,anch,contador_iteracion,debug,debug_pipe);
98     contador_iteracion++;
99     k--;
100
101  }
102
103     fclose(debug);
104     pclose(debug_pipe);
105
106     return 0;
107
108 }
109
110 double distanciaProt_Base(double x_prot,double i){//OJO! d= (XP-i) Los signos no son
        arbitrarios!
111     double dx=0.000001,distancia1,distancia2,XP;  //Precision en la posicion de x_prot.
            ARBITRARIO! (hablar con Fernando).
112     int vueltas=(int)(x_prot/Ncadena);
113     XP=x_prot-vueltas*Ncadena;
114     distancia1=fabs(XP-i);
115     distancia2=Ncadena-distancia1;
116     if(distancia1<distancia2)
117      return distancia1*(int)((XP-i)/distancia1);
118
119     else{
120      return distancia2*(int)((i-XP)/distancia1);
121     }
122  }
123
124 void PintaPotencialDebug(double *x, double gamma, double B, double anch,int
        contador_iteracion,FILE *debug, FILE *debug_pipe){
125     double Vton=0;
126     double Von[Ncadena];
127     double Vtoff=0;
128     double Voff[Ncadena];
129     double x_prot=0;
```

```
130     double elongacion=2;
131     elongacion=elongacion*2/(anch*anch);
132     int i,j,k;
133     double dist[Ncadena],exp3[Ncadena],tanhip[Ncadena];
134
135
136     for(i=0;i<1000;i++){
137             for(k=0;k<Ncadena;k++){
138                 dist[k]=distanciaProt_Base(x_prot,(double)k);
139                 exp3[k]=exp(-dist[k]*dist[k]/(anch*anch));
140                 tanhip[k]=tanh(gamma*x[k]);
141             }
142         for(j=0;j<Ncadena;j++){
143                 Voff[j]=-B/(sqrt(PI)*anch)*funcion_asim(exp3[j],dist[j],anch,0,B)*tanhip[j];
144                 Von[j]=-B/(sqrt(PI)*anch)*V_ratchett(x_prot,0.1,1.0,0.0,0.0)*tanhip[j]*
                        funcion_asim(exp3[j],dist[j],anch,0,B);
145                 Vton+=Von[j];
146                 Vtoff+=Voff[j];
147         }
148
149         fprintf(debug,"%lf\t%lf\t%lf\n",x_prot,Vtoff,Vton);
150         fflush(debug);
151         x_prot+=0.01;
152         Vton=0;
153         Vtoff=0;
154     }
155     fprintf(debug,"\n\n");
156     fprintf(debug_pipe,"plot␣'PotencialDebug_Prot'␣index␣%i␣using␣1:2␣w␣l,␣'
            PotencialDebug_Prot'␣index␣%i␣using␣1:3␣w␣l␣\n",contador_iteracion,
            contador_iteracion);
157     fflush(debug_pipe);
158 }
159
160 double dV_ratchett(double X, double d2,double altura,double epsilon){
161
162     if(((int)X)<=0)
163         X=2+X-((int)X);
164
165
166     if((fabs(X)-abs((int)X))<epsilon)
167         return 0;
168
169     if(((int)X)==((int)(X-1+d2)))
170         return (-altura/d2);
171
172     if(((int)X)>((int)(X-1+d2)))
173         return (altura/(1-d2));
174
175
176 }
177 double V_ratchett(double X,double d2, double altura,double epsilon,double suelo){
178     int i;
179
180     if(((int)X)<=0)
181         X=2+X-((int)X);
182
183     if(dV_ratchett(X,d2,altura,epsilon)>0)
184         return ((X-(int)X)*dV_ratchett(X,d2,altura,epsilon))-suelo;
185
186     if(dV_ratchett(X,d2,altura,epsilon)<0)
187         return ((X-(int)(X+d2))*dV_ratchett(X,d2,altura,epsilon))-suelo;
188
189 }
190
191 double funcion_asim(double exp3_i,double dist_i,double anch,double alfa,double B){
192     return (exp3_i*(1+erf(alfa/anch*dist_i)));
193
194 }
```

(4)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#define PI 3.1415926536
#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)


double Calc_fuerza(double x[],double x_prot, double a[], double D[], double alfa[],double S
    [], double G, double C, double xo, double ro,double gamma,double anch,double B,double
    asimetria, int Ncadena,int estadoATP, double fuerza[],double fuerza_externa, double
    auxiliar_STK[], double auxiliar_Morse[],double auxiliar_Prot,double uno[], double dos[],
    double tres[], double cuatro[], double exp0[], double exp1[],double exp2[],double exp3
    [],double dist[],double tanhip[]);
void Calc_Output(double p[],double p_prot,double x_prot,double x_prot_previo,double m_prot,
    double T,double gamma,double B,double anch,double TON,double TOFF,double *
    Energia_cinetica,double *velocidad_media,double t_term,double t_final, double dtau,int
    Ncadena,int n,int N,int Nterm,int indice_simulacion,int selectorCPC, FILE *Output,double
     fuerza_externa,double G,double asimetria);
float ran1(long *idum);
void Calcula_EP(double x[],double x_prot,double B,double gamma,double anch,double asimetria,
    int Ncadena,int puntos,int estadoATP, FILE *Epotencial);
double distanciaProt_Base(double x_prot,double i, int Ncadena);
void CalculaCinematica(double *x, double x_prot,double t, int Ncadena,FILE *Cinematica);
void CalculaTrayectoria_prot(double x_prot,double p_prot,double m_prot, double t, int
    estadoATP, FILE *Trayectoria_prot);
void Calcula_configPromedio(double x[], double promedio[],int Ncadena, int j, int N,int
    Nterm, FILE *config_final);
double funcion_asim(double exp3_i,double dist_i,double anch,double asimetria,double B);
double derivada_funcion_asim(double exp3_i,double dist_i,double anch,double asimetria,double
     B);


int main()
{

//Definicion de constantes y variables operativas:
    FILE *datosp,*secuencia;
    unsigned int i,n,indTemp,indGamma,indB,indAnch,ind_TON,ind_TOFF,N,Nterm,N_on,N_off,
        contadorON,contadorOFF,contador_muestra,indice_simulacion,ind_Fexterna;
    int selector_EP,selector_Output,selector_Cin,selector_Xprom,selector_Tray,selectorCPC;
    long idum;
    char basura[256],etiqueta[256],name_leyenda[256],base;

//Inicializacion del generador de numeros aleatorios
    idum=-(long)time(NULL);

//Definicion de constantes del algoritmo:
    const double lambda11=-0.567253;
    const double lambda21=0.516719;
    const double lambda22=0.499720;
    const double lambda31=0.030390;
    const double lambda32=-0.171658;

    const double beta21=0.516719;
    const double beta31=-0.397300;
    const double beta32=0.427690;
    const double beta41=-1.587731;
```

```
53      const double beta42=1.417263;
54      const double beta43=1.170469;
55
56      const double A2=0.644468;
57      const double A3=0.194450;
58      const double A4=0.161082;
59
60   //Definicion de variables del algoritmo
61      double *Y[5];
62      double *gx[4];
63      double *gp[4];
64      double random1;
65      double random2;
66
67  //Definicion de constantes globales del modelo:
68      double B,B0,incrementoB,G,anch,anch0,incrementoAnch,C,gamma,gamma0,incrementoGamma,dtau,
            T,T0,incrementoT,t_termalizacion,t_final,TON0,TOFF0,TON,TOFF,incremento_TON,
            incremento_TOFF,xo,ro,asimetria,fuerza_externa0,fuerza_externa,incr_fuerza_externa;
69      int Ncadena,muestra,PuntosT,PuntosB,PuntosAnch,PuntosGamma,Puntos_TON,Puntos_TOFF,
            PuntosFuerza_externa;
70
71  //Definicion de constantes locales del modelo
72      double *a,*alfa,*epsilon,*S,*D,eps_prot,m_prot;
73
74  //Definicion de variables del modelo:
75      double *x,*p,x_prot,p_prot,t,aux,aux_prot; //[x,p,t];
76      double *x_abs,*p_abs,x_prot_abs,p_prot_abs,t_abs;
77      double *fuerza,fuerza_prot;
78      int estadoATP; //0-> no ATP / 1-> si ATP
79
80  //Definicion de observables:
81      double Energia_cinetica,*config_promedio,velocidad_inst,velocidad_media,x_prot_previo;
82
83      FILE *EP_prot;
84      char name_ep[256];
85
86      FILE *Cinematica;
87      char name_cin[256];
88
89      FILE *Trayectoria_prot;
90      char name_tray[256];
91
92      FILE *config_final;
93      char name_config[256];
94
95      FILE *Output;
96      char name_out[256];
97
98
99  //Definicion de constantes auxiliares para ahorrar productos en las ecuaciones:
100     double *auxiliar_STK,*auxiliar_Morse,auxiliar_Prot;
101
102
103 //Definicion de variables auxiliares para la funcion FUERZA:
104     double *uno,*dos,*tres,*cuatro,*exp0,*exp1,*exp2,*exp3,*dist,*tanhip;
105
106
107  //Lectura de dtos :
108     datosp=fopen("datosASIM.inp","r");
109
110     fscanf(datosp,"%s␣%s\n",etiqueta,basura);
111
112     fscanf(datosp,"%i␣%s\n",&Ncadena,basura);
113     fscanf(datosp,"%lf␣%s\n",&dtau,basura);
114     fscanf(datosp,"%lf␣%s␣%lf␣%s␣%s\n",&T0,basura,&incrementoT,basura,&PuntosT,basura);
115     fscanf(datosp,"%lf␣%s␣%lf␣%s\n",&t_termalizacion,basura,&t_final,basura);
116     fscanf(datosp,"%i␣%s\n",&muestra,basura);
117
```

```
118     fscanf(datosp,"%f %s %f %s %i %s\n",&gamma0,basura,&incrementoGamma,basura,&PuntosGamma,
            basura);
119     fscanf(datosp,"%f %s %f %s %i %s\n",&B0,basura,&incrementoB,basura,&PuntosB,basura);
120     fscanf(datosp,"%f %s %f %s %i %s\n",&anch0,basura,&incrementoAnch,basura,&PuntosAnch,
            basura);
121     fscanf(datosp,"%f %s\n",&G,basura); //0= no; 1=si
122     fscanf(datosp,"%f %s %f %s %i %s\n",&TOFF0,basura,&incremento_TOFF,basura,&Puntos_TOFF,
            basura);
123     fscanf(datosp,"%f %s %f %s %i %s\n",&fuerza_externa0,basura,&incr_fuerza_externa,basura
            ,&PuntosFuerza_externa,basura);
124
125     fscanf(datosp,"%i %s\n",&selector_Output,basura); //0= no; 1=si
126     fscanf(datosp,"%i %s\n",&selector_Xprom,basura); //0= no; 1=si
127     fscanf(datosp,"%i %s\n",&selector_EP,basura); //0= no; 1=si
128     fscanf(datosp,"%i %s\n",&selector_Cin,basura); //0= no; 1=si
129     fscanf(datosp,"%i %s\n",&selector_Tray,basura); //0= no; 1=si
130     fscanf(datosp,"%i %s\n",&selectorCPC,basura);
131
132     fscanf(datosp,"%f %s %f %s %i %s\n",&TON0,basura,&incremento_TON,basura,&Puntos_TON,
            basura);
133     fscanf(datosp,"%f %s\n",&asimetria,basura);//0= no; 1=si
134     fscanf(datosp,"%s %s\n",basura,basura);
135     fscanf(datosp,"%s %s\n",basura,basura);
136     fscanf(datosp,"%s %s %s %s\n",basura,basura,basura,basura);
137     fscanf(datosp,"%s %s\n",basura,basura);
138
139     fclose(datosp);
140
141 //Asignacion de constantes del algoritmo:
142     for(i=0;i<5;i++)
143         Y[i]=malloc((Ncadena+1)*sizeof(double));
144     for(i=0;i<4;i++)
145         gx[i]=malloc((Ncadena+1)*sizeof(double));
146     for(i=0;i<4;i++)
147         gp[i]=malloc((Ncadena+1)*sizeof(double));
148
149 //Asignacion de cte globales del modelo
150     C=0.5;
151     xo=2;
152     ro=3;
153
154
155
156  //Lectura de la secuencia y asignacion de constantes locales:
157     a=malloc(Ncadena*sizeof(double));
158     alfa=malloc(Ncadena*sizeof(double));
159     epsilon=malloc(Ncadena*sizeof(double));
160     S=malloc(Ncadena*sizeof(double));
161     D=malloc(Ncadena*sizeof(double));
162     auxiliar_STK=malloc(Ncadena*sizeof(double));
163     auxiliar_Morse=malloc(Ncadena*sizeof(double));
164
165     m_prot=20;
166     eps_prot=15;
167
168     secuencia=fopen("secuencia.inp","r");
169     for(i=0;i<Ncadena;i++){
170         fscanf(secuencia,"%c\n",&base);
171
172         if(base=='A'){
173             S[i]=0.03616;
174             D[i]=1.0;
175             a[i]=1.0;
176             alfa[i]=0.20;
177             epsilon[i]=1.0;
178             auxiliar_STK[i]=0.5*S[i]*ro;
179             auxiliar_Morse[i]=2*D[i]*a[i];
180         }
```

```
181         if(base=='T'){
182             S[i]=0.03616;
183             D[i]=1.0;
184             a[i]=1.0;
185             alfa[i]=0.20;
186             epsilon[i]=1.0;
187             auxiliar_STK[i]=0.5*S[i]*ro;
188             auxiliar_Morse[i]=2*D[i]*a[i];
189         }
190         if(base=='C'){
191             S[i]=0.03616;
192             D[i]=1.5;
193             a[i]=1.5;
194             alfa[i]=0.20;
195             epsilon[i]=1.0;
196             auxiliar_STK[i]=0.5*S[i]*ro;
197             auxiliar_Morse[i]=2*D[i]*a[i];
198         }
199         if(base=='G'){
200             S[i]=0.03616;
201             D[i]=1.5;
202             a[i]=1.5;
203             alfa[i]=0.20;
204             epsilon[i]=1.0;
205             auxiliar_STK[i]=0.5*S[i]*ro;
206             auxiliar_Morse[i]=2*D[i]*a[i];
207         }
208 
209     }
210 
211     fclose(secuencia);
212 
213 //Asignacion de variables del modelo
214     x=malloc(Ncadena*sizeof(double));
215     p=malloc(Ncadena*sizeof(double));
216     x_abs=malloc(Ncadena*sizeof(double));
217     p_abs=malloc(Ncadena*sizeof(double));
218     fuerza=malloc(Ncadena*sizeof(double));
219 
220 //Asignacion de observables:
221     config_promedio=malloc(Ncadena*sizeof(double));
222 
223 
224 //Dimensionalizacion de variables auxiliares para la funcion FUERZA:
225     uno=malloc(Ncadena*sizeof(double));
226     dos=malloc(Ncadena*sizeof(double));
227     tres=malloc(Ncadena*sizeof(double));
228     cuatro=malloc(Ncadena*sizeof(double));
229     exp0=malloc(Ncadena*sizeof(double));
230     exp1=malloc(Ncadena*sizeof(double));
231     exp2=malloc(Ncadena*sizeof(double));
232     exp3=malloc(Ncadena*sizeof(double));
233     dist=malloc(Ncadena*sizeof(double));
234     tanhip=malloc(Ncadena*sizeof(double));
235 
236 
237 //Bucle de parametros.
238         if(selector_Output==1){
239             sprintf(name_out,"Output_%s",etiqueta);
240             Output=fopen(name_out,"w");
241             fprintf(Output,"IndiceSim\tt_total\tt_term\tTemperatura\tGamma\tB\tAnch\
                    tSolvatacion\ttON\ttOFF\tAsimetria\tFuerzaExterna\tEnergiaCinetica\tdof*kT
                    /2\tEC_normalizada\t<Eficacia>\t<Velocidad>\n");
242         }
243 T=T0;
244 gamma=gamma0;
245 B=B0;
246 anch=anch0;
```

```
247 TON=TON0;
248 TOFF=TOFF0;
249 fuerza_externa=fuerza_externa0;
250  for(indTemp=0;indTemp<PuntosT;indTemp++){
251         for(indGamma=0;indGamma<PuntosGamma;indGamma++){
252                 for(indB=0;indB<PuntosB;indB++){
253                         for(indAnch=0;indAnch<PuntosAnch;indAnch++){
254                                 for(ind_TON=0;ind_TON<Puntos_TON;ind_TON++){
255                                     for(ind_TOFF=0;ind_TOFF<Puntos_TOFF;ind_TOFF++){
256                                             for(ind_Fexterna=0;ind_Fexterna<
                                                 PuntosFuerza_externa;ind_Fexterna++){
257
258  //Inicializacion de variables y observables
259         indice_simulacion=indTemp*PuntosGamma*PuntosB*PuntosAnch*Puntos_TON*Puntos_TOFF*
                PuntosFuerza_externa+indGamma*PuntosB*PuntosAnch*Puntos_TON*Puntos_TOFF*
                PuntosFuerza_externa+indB*PuntosAnch*Puntos_TON*Puntos_TOFF*PuntosFuerza_externa
                +indAnch*Puntos_TON*Puntos_TOFF*PuntosFuerza_externa+ind_TON*Puntos_TOFF*
                PuntosFuerza_externa+ind_TOFF*PuntosFuerza_externa+ind_Fexterna;
260
261         Energia_cinetica=0;
262         velocidad_media=0;
263
264         for(i=0;i<Ncadena;i++){
265             x[i]=0;
266             p[i]=0;
267             config_promedio[i]=0;
268         }
269         t=0;
270         x_prot=25;//rand()/(RAND_MAX*1.0)*Ncadena;
271         p_prot=0;
272         x_prot_previo=x_prot;
273         aux=sqrt(dtau*2*T); //OJO! Que m_base y eps_base son 1!
274         aux_prot=sqrt(m_prot*dtau*2*T*eps_prot);
275
276
277         if(selector_EP==1){
278             sprintf(name_ep,"EP_%s_%i",etiqueta,indice_simulacion);
279             EP_prot=fopen(name_ep,"w");
280             fprintf(EP_prot,"#t_term␣%lf\n#t_fnal␣%lf\n#dt␣%lf\n#Ncadena␣%i\n#muestra␣%i\n",
                    t_termalizacion,t_final,dtau,Ncadena,muestra);
281         }
282
283         if(selector_Cin==1){
284             sprintf(name_cin,"Cinematica_%s_%i",etiqueta,indice_simulacion);
285             Cinematica=fopen(name_cin,"w");
286             fprintf(Cinematica,"#t_term␣%lf\n#t_fnal␣%lf\n#dt␣%lf\n#Ncadena␣%i\n#muestra␣%i\
                    n",t_termalizacion,t_final,dtau,Ncadena,muestra);
287         }
288
289         if(selector_Tray==1){
290             sprintf(name_tray,"Trayectoria_%s_%i",etiqueta,indice_simulacion);
291             Trayectoria_prot=fopen(name_tray,"w");
292             fprintf(Trayectoria_prot,"Tiempo\tx_prot\tv_prot\testadoATP\n");
293         }
294
295         if(selector_Xprom==1){
296             sprintf(name_config,"Config_%s_%i",etiqueta,indice_simulacion);
297             config_final=fopen(name_config,"w");
298         }
299
300
301
302
303
304         N=(int)(t_final/dtau);
305         Nterm=(int)(t_termalizacion/dtau);
306         N_on=(int)(TON/dtau);
307         contadorON=0;
```

```
308         N_off=(int)(TOFF/dtau);
309         contadorOFF=0;
310         estadoATP=0;
311         if(N_off==0)
312             estadoATP=1;
313         contador_muestra=0;
314         auxiliar_Prot=B/(anch*sqrt(PI))*gamma;
315
316
317 //TERMALIZACION Y DINAMICA
318         for(n=0;n<N;n++){
319              if(100*n%N==0) //Porcentaje!
320                   printf("%f\n",100.0*n/N);
321
322 //Comienza integracion en dtau:
323             for(i=0;i<(Ncadena+1);i++){
324                 random1=ran1(&idum);
325                 random2=ran1(&idum);
326                     Y[0][i]=-sqrt(-2*log(random1))*cos(2*PI*random2);
327                     Y[1][i]=-lambda11*sqrt(-2*log(random1))*cos(2*PI*random2);
328                     Y[2][i]=-lambda21*sqrt(-2*log(random1))*cos(2*PI*random2)-lambda22*sqrt
                            (-2*log(random1))*sin(2*PI*random2);
329                     Y[3][i]=-lambda31*sqrt(-2*log(random1))*cos(2*PI*random2)-lambda32*sqrt
                            (-2*log(random1))*sin(2*PI*random2);
330                     Y[4][i]=-sqrt(-2*log(random1))*cos(2*PI*random2);
331             } //Produccion de la matriz Y
332
333                 for(i=0;i<Ncadena;i++){
334                     x_abs[i]=x[i];
335                     p_abs[i]=p[i]+aux*Y[1][i];
336                 }
337                 x_prot_abs=x_prot;
338                 p_prot_abs=p_prot+aux_prot*Y[1][Ncadena];
339                 //t_abs=t;
340
341                 fuerza_prot=Calc_fuerza(x_abs,x_prot_abs,a,D,alfa,S,G,C,xo,ro,gamma,anch,B,
                        asimetria,Ncadena,estadoATP,fuerza,fuerza_externa,auxiliar_STK,
                        auxiliar_Morse,auxiliar_Prot,uno,dos,tres,cuatro,exp0,exp1,exp2,exp3,
                        dist,tanhip);
342                 for(i=0;i<Ncadena;i++){
343                     gx[0][i]=p_abs[i];
344                     gp[0][i]=fuerza[i]-epsilon[i]*p_abs[i];
345                 }
346                 gx[0][Ncadena]=p_prot_abs/m_prot;
347                 gp[0][Ncadena]=fuerza_prot-eps_prot*p_prot_abs;
348                 //    g[0][2]=1;
349                 //PASO 1
350
351                 for(i=0;i<Ncadena;i++){
352                     x_abs[i]=x[i]+dtau*beta21*gx[0][i];
353                     p_abs[i]=p[i]+dtau*beta21*gp[0][i]+aux*Y[2][i];
354                 }
355                 x_prot_abs=x_prot+dtau*beta21*gx[0][Ncadena];
356                 p_prot_abs=p_prot+dtau*beta21*gp[0][Ncadena]+aux_prot*Y[2][Ncadena];
357                 //t_abs=t+dtau*0.516719;
358
359                 fuerza_prot=Calc_fuerza(x_abs,x_prot_abs,a,D,alfa,S,G,C,xo,ro,gamma,anch,B,
                        asimetria,Ncadena,estadoATP,fuerza,fuerza_externa,auxiliar_STK,
                        auxiliar_Morse,auxiliar_Prot,uno,dos,tres,cuatro,exp0,exp1,exp2,exp3,
                        dist,tanhip);
360                 for(i=0;i<Ncadena;i++){
361                     gx[1][i]=p_abs[i];
362                     gp[1][i]=fuerza[i]-epsilon[i]*p_abs[i];
363                 }
364                 gx[1][Ncadena]=p_prot_abs/m_prot;
365                 gp[1][Ncadena]=fuerza_prot-eps_prot*p_prot_abs;
366                 //    g[1][2]=1;
367                 //PASO 2
```

```
368
369                     for(i=0;i<Ncadena;i++){
370                         x_abs[i]=x[i]+dtau*(beta31*gx[0][i]+beta32*gx[1][i]);
371                         p_abs[i]=p[i]+dtau*(beta31*gp[0][i]+beta32*gp[1][i])+aux*Y[3][i];
372                     }
373                     x_prot_abs=x_prot+dtau*(beta31*gx[0][Ncadena]+beta32*gx[1][Ncadena]);
374                     p_prot_abs=p_prot+dtau*(beta31*gp[0][Ncadena]+beta32*gp[1][Ncadena])+
                            aux_prot*Y[3][Ncadena];
375                     //t_abs=t+dtau*(-0.397300+0.427690);
376
377                     fuerza_prot=Calc_fuerza(x_abs,x_prot_abs,a,D,alfa,S,G,C,xo,ro,gamma,anch,B,
                            asimetria,Ncadena,estadoATP,fuerza,fuerza_externa,auxiliar_STK,
                            auxiliar_Morse,auxiliar_Prot,uno,dos,tres,cuatro,exp0,exp1,exp2,exp3,
                            dist,tanhip);
378                     for(i=0;i<Ncadena;i++){
379                         gx[2][i]=p_abs[i];
380                         gp[2][i]=fuerza[i]-epsilon[i]*p_abs[i];
381                     }
382                     gx[2][Ncadena]=p_prot_abs/m_prot;
383                     gp[2][Ncadena]=fuerza_prot-eps_prot*p_prot_abs;
384                     //g[2][2]=1;
385                     //PASO 3
386
387                     for(i=0;i<Ncadena;i++){
388                         x_abs[i]=x[i]+dtau*(beta41*gx[0][i]+beta42*gx[1][i]+beta43*gx[2][i]);
389                         p_abs[i]=p[i]+dtau*(beta41*gp[0][i]+beta42*gp[1][i]+beta43*gp[2][i])+aux
                                *Y[4][i];
390                     }
391                     x_prot_abs=x_prot+dtau*(beta41*gx[0][Ncadena]+beta42*gx[1][Ncadena]+beta43*
                            gx[2][Ncadena]);
392                     p_prot_abs=p_prot+dtau*(beta41*gp[0][Ncadena]+beta42*gp[1][Ncadena]+beta43*
                            gp[2][Ncadena])+aux_prot*Y[4][Ncadena];
393                     //t_abs=t+dtau*(-1.587731+1.417263+1.170469);
394
395                     fuerza_prot=Calc_fuerza(x_abs,x_prot_abs,a,D,alfa,S,G,C,xo,ro,gamma,anch,B,
                            asimetria,Ncadena,estadoATP,fuerza,fuerza_externa,auxiliar_STK,
                            auxiliar_Morse,auxiliar_Prot,uno,dos,tres,cuatro,exp0,exp1,exp2,exp3,
                            dist,tanhip);
396                     for(i=0;i<Ncadena;i++){
397                         gx[3][i]=p_abs[i];
398                         gp[3][i]=fuerza[i]-epsilon[i]*p_abs[i];
399                     }
400                     gx[3][Ncadena]=p_prot_abs/m_prot;
401                     gp[3][Ncadena]=fuerza_prot-eps_prot*p_prot_abs;
402                     //    g[3][2]=1;
403                     //PASO 4
404
405                     for(i=0;i<Ncadena;i++){
406                         x[i]=x[i]+dtau*(A2*gx[1][i]+A3*gx[2][i]+A4*gx[3][i]);
407                         p[i]=p[i]+dtau*(A2*gp[1][i]+A3*gp[2][i]+A4*gp[3][i])+aux*Y[0][i];
408                     }
409                     x_prot_previo=x_prot; //Memoria de x_prot
410                     x_prot=x_prot+dtau*(A2*gx[1][Ncadena]+A3*gx[2][Ncadena]+A4*gx[3][Ncadena]);
411                     p_prot=p_prot+dtau*(A2*gp[1][Ncadena]+A3*gp[2][Ncadena]+A4*gp[3][Ncadena])+
                            aux_prot*Y[0][Ncadena];
412                     t+=dtau; //*(0.644468*g[1][2]+0.194450*g[2][2]+0.161082*g[3][2]);
413
414     if(selectorCPC){//No es necesario implementar CPC en la cinematica: ya está introducido
            en la fuerza (periodica).
415                 if(x_prot<0)
416                     x_prot=Ncadena+x_prot;
417
418                 if(x_prot>Ncadena)
419                     x_prot=x_prot-Ncadena; //CPC
420     }
421 ////////////////////////////////////////////////Aqui acaba la integracion en dtau.
422
423 //toma de medidas:
```

```
424             if(n>=Nterm){ //Termalizacion!
425                 contador_muestra++;
426
427                 if(selector_Xprom==1)
428                     Calcula_configPromedio(x,config_promedio,Ncadena,n,N,Nterm,config_final)
                            ;
429                 if(selector_Output==1)
430                     Calc_Output(p,p_prot,x_prot,x_prot_previo,m_prot,T,gamma,B,anch,TON,TOFF
                            ,&Energia_cinetica,&velocidad_media,t_termalizacion,t_final,dtau,
                            Ncadena,n,N,Nterm,indice_simulacion,selectorCPC,Output,
                            fuerza_externa,G,asimetria); //Observables promediados no tienen
                            sentido muestreados!
431
432                 if(contador_muestra==muestra){
433                     if(selector_Tray==1)
434                         CalculaTrayectoria_prot(x_prot,p_prot,m_prot,t,estadoATP,
                                Trayectoria_prot);
435                     if(selector_EP==1)
436                         Calcula_EP(x,x_prot,B,gamma,anch,asimetria,Ncadena,1000,estadoATP,
                                EP_prot);
437                     if(selector_Cin==1)
438                         CalculaCinematica(x,x_prot,t,Ncadena,Cinematica);
439
440                     contador_muestra=0;
441                 }
442
443             }
444
445
446             if(estadoATP==0)
447                 contadorOFF++;
448
449             if(estadoATP==1)
450                 contadorON++;
451
452             if(contadorON==N_on){
453                 estadoATP=0;
454                 contadorON=0;
455             }
456             if(contadorOFF==N_off){
457                 estadoATP=1;
458                 contadorOFF=0;
459             }
460         }//Fin del experimento l-esimo
461
462          if(selector_EP==1)
463             fclose(EP_prot);
464
465         if(selector_Cin==1)
466             fclose(Cinematica);
467
468         if(selector_Tray==1)
469             fclose(Trayectoria_prot);
470
471         if(selector_Xprom==1)
472             fclose(config_final);
473
474
475         printf("%/%i\n\n",indice_simulacion+1,PuntosT*PuntosGamma*PuntosB*PuntosAnch*
                Puntos_TON*Puntos_TOFF);
476
477                                         fuerza_externa=fuerza_externa+incr_fuerza_externa;
478                                             }
479
480                                     fuerza_externa=fuerza_externa0;
481                                     TOFF=TOFF*incremento_TOFF;
482                                     }
483                                     TOFF=TOFF0;
```

```
484                                     TON=TON*incremento_TON;
485                                 }
486                             fuerza_externa=fuerza_externa0;
487                             TOFF=TOFF0;
488                             TON=TON0;
489                             anch=anch+incrementoAnch;
490                         }
491                         fuerza_externa=fuerza_externa0;
492                         TOFF=TOFF0;
493                         TON=TON0;
494                         anch=anch0;
495                         B=B+incrementoB;
496                     }
497                     fuerza_externa=fuerza_externa0;
498                     TOFF=TOFF0;
499                     TON=TON0;
500                     B=B0;
501                     anch=anch0;
502                     gamma=gamma+incrementoGamma;
503             }
504             fuerza_externa=fuerza_externa0;
505             TOFF=TOFF0;
506             TON=TON0;
507             gamma=gamma0;
508             B=B0;
509             anch=anch0;
510             T=T+incrementoT;
511         }
512
513
514     //fin del programa.
515     printf("\n\n PROGRAMA FINALIZADO\n\n");
516     free(a);
517     free(alfa);
518     free(epsilon);
519     free(S);
520     free(D);
521     free(x);
522     free(x_abs);
523     free(p);
524     free(p_abs);
525     free(fuerza);
526     free(config_promedio);
527     free(auxiliar_STK);
528     free(auxiliar_Morse);
529     free(uno);
530     free(dos);
531     free(tres);
532     free(cuatro);
533     free(exp0);
534     free(exp1);
535     free(exp2);
536     free(exp3);
537     free(dist);
538     free(tanhip);
539
540
541     for(i=0;i<5;i++)
542         free(Y[i]);
543     for(i=0;i<4;i++)
544         free(gx[i]);
545     for(i=0;i<4;i++)
546         free(gp[i]);
547     if(selector_Output==1)
548         fclose(Output);
549
550         return 0;
551     }
```

```
552
553
554 double Calc_fuerza(double x[],double x_prot, double a[], double D[], double alfa[],double S
    [], double G, double C, double xo, double ro,double gamma,double anch,double B,double
    asimetria, int Ncadena,int estadoATP, double fuerza[],double fuerza_externa, double
    auxiliar_STK[], double auxiliar_Morse[],double auxiliar_Prot,double uno[], double dos[],
    double tres[], double cuatro[], double exp0[], double exp1[],double exp2[],double exp3
    [],double dist[],double tanhip[]){
555     double fuerza_prot,suma=0;
556     int i,j;
557
558
559     for(j=0;j<Ncadena;j++){
560         dist[j]=distanciaProt_Base(x_prot,(double)j,Ncadena);
561         exp0[j]=exp(-a[j]*x[j]);
562         exp1[j]=exp(-(x[j]-xo)*(x[j]-xo)/C);
563         exp2[j]=exp(-alfa[j]*x[j]);
564         exp3[j]=exp(-dist[j]*dist[j]/(anch*anch));
565         tanhip[j]=tanh(gamma*x[j]);
566     }
567
568     for(j=0;j<Ncadena;j++){
569
570             uno[j]=auxiliar_Morse[j]*exp0[j]*(exp0[j]-1)+2*G/C*(x[j]-xo)*exp1[j];
571
572        if((j>0) && (j<(Ncadena-1))){
573
574            dos[j]=(x[j]-x[j-1])*(auxiliar_STK[j]*exp2[j]*exp2[j-1]*(alfa[j]*(x[j]-x[j-1])-2)
                -S[j]);
575
576            tres[j]=(x[j]-x[j+1])*(auxiliar_STK[j]*exp2[j]*exp2[j+1]*(alfa[j]*(x[j]-x[j+1])
                -2)-S[j]);
577
578        }
579
580         if(j==(Ncadena-1)){
581
582             dos[j]=(x[j]-x[j-1])*(auxiliar_STK[j]*exp2[j]*exp2[j-1]*(alfa[j]*(x[j]-x[j-1])
                -2)-S[j]);
583
584             tres[j]=(x[j]-x[0])*(auxiliar_STK[j]*exp2[j]*exp2[0]*(alfa[j]*(x[j]-x[0])-2)-S[j
                ]);
585
586         }
587
588         if(j==0){
589
590             dos[j]=(x[j]-x[Ncadena-1])*(auxiliar_STK[j]*exp2[j]*exp2[Ncadena-1]*(alfa[j]*(x[
                j]-x[Ncadena-1])-2)-S[j]);
591
592             tres[j]=(x[j]-x[j+1])*(auxiliar_STK[j]*exp2[j]*exp2[j+1]*(alfa[j]*(x[j]-x[j+1])
                -2)-S[j]);
593
594         }
595
596         if(estadoATP==1){
597
598             cuatro[j]=auxiliar_Prot*funcion_asim(exp3[j],dist[j],anch,asimetria,B)*(1-tanhip
                [j]*tanhip[j]);
599
600             suma+=tanhip[j]*derivada_funcion_asim(exp3[j],dist[j],anch,asimetria,B);
601
602     }
603
604         if(estadoATP==0){
605
606             cuatro[j]=auxiliar_Prot*exp3[j]*(1-tanhip[j]*tanhip[j]);
607
```

```
608                 suma+=tanhip[j]*derivada_funcion_asim(exp3[j],dist[j],anch,0,B);
609
610
611             }
612
613             fuerza[j]=uno[j]+dos[j]+tres[j]+cuatro[j];
614
615         }
616
617
618         fuerza_prot=auxiliar_Prot/gamma*suma+fuerza_externa;
619
620
621         return fuerza_prot;
622
623     }
624
625
626     void Calc_Output(double p[],double p_prot,double x_prot,double x_prot_previo,double m_prot,
        double T,double gamma,double B,double anch,double TON,double TOFF,double *
        Energia_cinetica,double *velocidad_media,double t_term,double t_final, double dtau,int
        Ncadena,int n,int N,int Nterm,int indice_simulacion,int selectorCPC, FILE *Output,double
         fuerza_externa,double G,double asimetria){
627         int i;
628         double EC=0,time;
629
630         if(n==Nterm){
631             *Energia_cinetica=0;
632             *velocidad_media=0;
633
634         }
635
636         for(i=0;i<Ncadena;i++)
637             EC+=0.5*p[i]*p[i];
638         EC+=0.5*p_prot*p_prot/m_prot;
639
640         *Energia_cinetica=*Energia_cinetica+EC;
641         if(selectorCPC==0)
642             *velocidad_media+=(x_prot-x_prot_previo);
643
644
645         if(n==(N-1)){
646             time=dtau/(t_final-t_term);
647             if(selectorCPC==0)
648                 fprintf(Output,"%\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\n",indice_simulacion,t_final,t_term,T,gamma,B,anch,G,TON,
                        TOFF,asimetria,fuerza_externa,*Energia_cinetica*time,(Ncadena+1)*0.5*T,*
                        Energia_cinetica*time/((Ncadena+1)*0.5*T),*velocidad_media/(t_final-t_term)
                        *(TON+TOFF),(*velocidad_media)/(t_final-t_term));
649             if(selectorCPC==1)
650                 fprintf(Output,"%\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%s\t%s\n",indice_simulacion,t_final,t_term,T,gamma,B,anch,G,TON,
                        TOFF,asimetria,fuerza_externa,*Energia_cinetica*time,(Ncadena+1)*0.5*T,*
                        Energia_cinetica*time/((Ncadena+1)*0.5*T),"CPC","CPC");
651         }
652
653     }
654
655
656     float ran1(long *idum){
657         int j;
658         long k;
659         static long iy=0;
660         static long iv[NTAB];
661         float temp;
662         if (*idum <= 0 || !iy) {
663         if (-(*idum) < 1) *idum=1;
664         else *idum = -(*idum);
```

```
665
666     for (j=NTAB+7;j>=0;j--) {
667         k=(*idum)/IQ;
668         *idum=IA*(*idum-k*IQ)-IR*k;
669          if (*idum < 0)
670             *idum += IM;
671          if (j < NTAB)
672             iv[j] = *idum;
673         }
674         iy=iv[0];
675     }
676     k=(*idum)/IQ;
677     *idum=IA*(*idum-k*IQ)-IR*k;
678     if (*idum < 0) *idum += IM;
679     j=iy/NDIV;
680     iy=iv[j];
681     iv[j] = *idum;
682     if ((temp=AM*iy) > RNMX) return RNMX;
683     else return temp;
684 }
685
686
687
688 void Calcula_EP(double x[],double x_prot,double B,double gamma,double anch,double asimetria,
        int Ncadena,int puntos,int estadoATP, FILE *Epotencial){
689     double XP=0;
690     double dx=(double)(Ncadena)/puntos;
691     double V=0;
692     int i,j;
693
694     if(estadoATP==0){
695         for(i=0;i<puntos;i++){
696             for(j=0;j<Ncadena;j++){
697                 V+=B/(sqrt(PI)*anch)*tanh(gamma*x[j])*funcion_asim(exp(-distanciaProt_Base(
                        XP,(double)j,Ncadena)*distanciaProt_Base(XP,(double)j,Ncadena)/(anch*
                        anch)),distanciaProt_Base(XP,(double)j,Ncadena),anch,0,B);
698             }
699             fprintf(Epotencial,"%lf\t%lf\n",XP,V);
700             XP+=dx;
701             V=0;
702         }//Aqui acaba de pintar EPcadena. Indice PAR.
703         fprintf(Epotencial,"\n\n");
704
705
706         for(j=0;j<Ncadena;j++)
707             V+=B/(sqrt(PI)*anch)*tanh(gamma*x[j])*funcion_asim(exp(-distanciaProt_Base(
                    x_prot,(double)j,Ncadena)*distanciaProt_Base(x_prot,(double)j,Ncadena)/(anch
                    *anch)),distanciaProt_Base(x_prot,(double)j,Ncadena),anch,0,B);
708
709         fprintf(Epotencial,"%lf\t%lf\n",x_prot,V);
710
711         fprintf(Epotencial,"\n\n");
712     }
713
714
715     V=0;
716     if(estadoATP==1){
717         for(i=0;i<puntos;i++){
718             for(j=0;j<Ncadena;j++)
719                 V+=B/(sqrt(PI)*anch)*tanh(gamma*x[j])*funcion_asim(exp(-distanciaProt_Base(
                        XP,(double)j,Ncadena)*distanciaProt_Base(XP,(double)j,Ncadena)/(anch*
                        anch)),distanciaProt_Base(XP,(double)j,Ncadena),anch,asimetria,B);
720
721             fprintf(Epotencial,"%lf\t%lf\n",XP,V);
722             XP+=dx;
723             V=0;
724         }//Aqui acaba de pintar EPcadena. Indice PAR.
725         fprintf(Epotencial,"\n\n");
```

```
726
727     V=0;
728         for(j=0;j<Ncadena;j++)
729             V+=-B/(sqrt(PI)*anch)*tanh(gamma*x[j])*funcion_asim(exp(-distanciaProt_Base(
                    x_prot,(double)j,Ncadena)*distanciaProt_Base(x_prot,(double)j,Ncadena)/(anch
                    *anch)),distanciaProt_Base(x_prot,(double)j,Ncadena),anch,asimetria,B);
730
731         fprintf(Epotencial,"%f\t%f\n",x_prot,V);
732
733         fprintf(Epotencial,"\n\n");
734     }
735
736 }
737
738 double distanciaProt_Base(double x_prot,double i, int Ncadena){//OJO! d= (XP-i) Los signos
        no son arbitrarios!
739     double dx=0.000001,distancia1,distancia2,XP;  //Precision en la posicion de x_prot.
        ARBITRARIO! (hablar con Fernando).
740     int vueltas=(int)(x_prot/Ncadena);
741     if(x_prot<0)
742         vueltas=vueltas-1;
743     XP=x_prot-vueltas*Ncadena;
744    distancia1=fabs(XP-i);
745    distancia2=Ncadena-distancia1;
746    if(distancia1<distancia2)
747     return distancia1*(int)((XP-i)/distancia1);
748
749    else{
750     return distancia2*(int)((i-XP)/distancia1);
751    }
752  }
753
754
755
756 void CalculaCinematica(double *x, double x_prot,double t, int Ncadena,FILE *Cinematica){
757     int i,j;
758     for(i=0;i<Ncadena;i++)
759         fprintf(Cinematica,"%i\t%f\n",i,(*(x+i)));
760
761     fprintf(Cinematica,"\n\n");
762
763     for(i=0;i<Ncadena;i++)
764         fprintf(Cinematica,"%i\t%f\n",i,-(*(x+i)));
765
766     fprintf(Cinematica,"\n\n");
767
768     if((int)x_prot<Ncadena-1)
769         fprintf(Cinematica,"%f\t%f\n",x_prot,(*(x+(int)x_prot))+((*(x+(int)x_prot+1))-(*(x
                +(int)x_prot)))*(x_prot-(int)x_prot));
770     if((int)x_prot==Ncadena-1)
771         fprintf(Cinematica,"%f\t%f\n",x_prot,(*(x+(int)x_prot))+((*(x))-(*(x+(int)x_prot))
                )*(x_prot-(int)x_prot));
772
773     fprintf(Cinematica,"\n\n");
774
775 }
776
777 void CalculaTrayectoria_prot(double x_prot,double p_prot,double m_prot, double t, int
        estadoATP, FILE *Trayectoria_prot){
778     fprintf(Trayectoria_prot,"%f\t%f\t%f\t%i\n",t,x_prot,p_prot/m_prot,estadoATP);
779 }
780
781 void Calcula_configPromedio(double x[], double promedio[],int Ncadena, int j, int N,int
        Nterm, FILE *config_final){
782     int i;
783
784     for(i=0;i<Ncadena;i++)
785         promedio[i]=promedio[i]+x[i];
```

```
786
787         if(j==(N-1)){
788             fprintf(config_final,"Base\tconfig_promedio\tconfig_final\n");
789             for(i=0;i<Ncadena;i++)
790                 fprintf(config_final,"%i\t%lf\t%lf\n",i,promedio[i]/(double)(N-Nterm),x[i]);
791         }
792     }
793
794
795
796     double funcion_asim(double exp3_i,double dist_i,double anch,double asimetria,double B){
797         return (exp3_i*(1+erf(asimetria/anch*dist_i)));
798
799     }
800
801     double derivada_funcion_asim(double exp3_i,double dist_i,double anch,double asimetria,double
          B){
802         return (2/anch*exp3_i*(-dist_i/anch*(1+erf(asimetria*dist_i/anch))+asimetria/sqrt(PI)*
              exp(-asimetria*asimetria*dist_i*dist_i/(anch*anch))));
803     }
```

(8)

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
    unsigned int indice_inicial,indice_final,i,seleccion1,seleccion2,Ncadena,muestra;
    double t_dibujo_i,t_dibujo_f,dt,y_min,y_max,t_term,t_final,aux,factor_norm;
    char basura[256],name[256];
    FILE *pipe,*reading;
    printf("\nPINTA CINEMATICA DE LA CADENA Y LA PROTEINA: fichero 'Cinematica_base_prot' \n
        ------------------------------------\n\n");

//Lectura de datos del fichero:
    printf("Nombre del fichero:\n");
    scanf("%s",name);
    reading=fopen(name,"r");
    fscanf(reading,"%s %f\n",basura,&t_term);
    fscanf(reading,"%s %f\n",basura,&t_final);
    fscanf(reading,"%s %f\n",basura,&dt);
    fscanf(reading,"%s %i\n",basura,&Ncadena);
    fscanf(reading,"%s %i\n",basura,&muestra);

//Busqueda del factor de normalizacion:
    y_min=0;
    y_max=0;
    for(i=0;i<(int)((2*Ncadena+1)*(t_final-t_term)/dt);i++){
        fscanf(reading,"%s\t%f\n",basura,&aux);
        if(aux<y_min)
            y_min=aux;
        if(aux>y_max)
            y_max=aux;
    }
    if(y_max>fabs(y_min))
        factor_norm=y_max;
    else
        factor_norm=fabs(y_min);
    printf("y_max= %f\ny_min= %f\nFactor= %f\n",y_max,y_min,factor_norm);

    fclose(reading); //Ya tenemos el dato de dt, y el factor de normalizacion.


//Representacion
    pipe=popen("gnuplot","w");

    printf("Tiempo de medida: %f\n\n",t_final-t_term);



    printf("Instante inicial de la representacion:\n");
    scanf("%f",&t_dibujo_i);
    printf("Instante final de la representacion:\n");
    scanf("%f",&t_dibujo_f);

    indice_inicial=(int)t_dibujo_i/(dt*muestra);
    indice_final=(int)t_dibujo_f/(dt*muestra);

    printf("\n\n");

    printf("Modo de  representacion: video(0) / uno-a-uno(1)\n");
    scanf("%i",&seleccion1);


    printf("Representacion normalizada? SI(0) / NO(1)\n");
    scanf("%i",&seleccion2);
    getchar();
```

```
66
67      if(seleccion2==1){
68          fprintf(pipe,"set␣yrange␣[%lf:%lf]\n",y_min,y_max);
69
70          if(seleccion1==0){
71              for(i=indice_inicial;i<indice_final;i++){
72                  fprintf(pipe,"plot␣'%s'␣index␣%i:%i␣using␣1:2␣w␣l,␣'%s'␣index␣%i␣using␣1:2\n
                        ",name,3*i,3*i+1,name,3*i+2);
73                  fflush(pipe);
74              }
75          }
76
77          if(seleccion1==1){
78                  for(i=indice_inicial;i<indice_final;i++){
79                  fprintf(pipe,"plot␣'%s'␣index␣%i:%i␣using␣1:2␣w␣l,␣'%s'␣index␣%i␣using␣1:2\n
                        ",name,3*i,3*i+1,name,3*i+2);
80                  fflush(pipe);
81                  getchar();
82                  }
83          }
84      }
85
86
87
88      if(seleccion2==0){
89              fprintf(pipe,"set␣yrange␣[%lf:%lf]\n",y_min/factor_norm,y_max/factor_norm);
90          if(seleccion1==0){
91                  for(i=indice_inicial;i<indice_final;i++){
92                  fprintf(pipe,"plot␣'%s'␣index␣%i:%i␣using␣1:($2/%lf)␣w␣l,␣'%s'␣index␣%i␣
                        using␣1:($2/%lf)\n",name,3*i,3*i+1,factor_norm,name,3*i+2,factor_norm);
93                  fflush(pipe);
94                  }
95          }
96          if(seleccion1==1){
97              for(i=indice_inicial;i<indice_final;i++){
98                  fprintf(pipe,"plot␣'%s'␣index␣%i:%i␣using␣1:($2/%lf)␣w␣l,␣'%s'␣index␣%i␣
                        using␣1:($2/%lf)\n",name,3*i,3*i+1,factor_norm,name,3*i+2,factor_norm);
99                  fflush(pipe);
100                 getchar();
101                 }
102         }
103     }
104
105
106
107
108 fclose(pipe);
109     return 0;
110 }
```

(9)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
    int indice_inicial,indice_final,i,seleccion1,seleccion2,Ncadena,muestra;
    double t_dibujo_i,t_dibujo_f,dt,y_min,y_max,t_term,t_final,aux,factor_norm;
    char basura[256],name[256];
    FILE *pipe,*reading;
printf("\nPINTA POTENCIAL DE LA PROTEINA: fichero 'EP_prot' \n
    ----------------------------------\n\n");

//Lectura de datos del fichero:
    printf("Nombre del fichero:\n");
    scanf("%s",name);
    reading=fopen(name,"r");
    fscanf(reading,"%s %f\n",basura,&t_term);
    fscanf(reading,"%s %f\n",basura,&t_final);
    fscanf(reading,"%s %f\n",basura,&dt);
    fscanf(reading,"%s %i\n",basura,&Ncadena);
    fscanf(reading,"%s %i\n",basura,&muestra);

//Busqueda del factor de normalizacion:
    y_min=0;
    y_max=0;
    for(i=0;i<(int)(112*(t_final-t_term)/dt);i++){
        fscanf(reading,"%s\t%f\n",basura,&aux);
        if(aux<y_min)
            y_min=aux;
        if(aux>y_max)
            y_max=aux;
    }
    if(y_max>fabs(y_min))
        factor_norm=y_max;
    else
        factor_norm=fabs(y_min);
    printf("y_max= %f\ny_min= %f\nFactor= %f\n",y_max,y_min,factor_norm);

    fclose(reading); //Ya tenemos el dato de dt, y el factor de normalizacion.

//Representacion
    pipe=popen("gnuplot","w");

    printf("Tiempo de medida: %f\n\n",t_final-t_term);



    printf("Instante inicial de la representacion:\n");
    scanf("%f",&t_dibujo_i);
    printf("Instante final de la representacion:\n");
    scanf("%f",&t_dibujo_f);
    indice_inicial=(int)t_dibujo_i/(dt*muestra);
    indice_final=(int)t_dibujo_f/(dt*muestra);

    printf("\n\n");

    printf("Modo de  representacion: video(0) / uno-a-uno(1)\n");
    scanf("%i",&seleccion1);


    printf("Representacion normalizada? SI(0) / NO(1)\n");
    scanf("%i",&seleccion2);
    getchar();

    if(seleccion2==1){
```

```
66          fprintf(pipe,"set␣yrange␣[%lf:%lf]\n",y_min,y_max);
67          if(seleccion1==0){
68              for(i=indice_inicial;i<indice_final;i++){
69                  fprintf(pipe,"plot␣'%s'␣index␣%i␣using␣1:2␣w␣l,␣'%s'␣index␣%i␣using␣1:2\n",
                        name,2*i,name,2*i+1);
70                  fflush(pipe);
71              }
72          }
73
74          if(seleccion1==1){
75                  for(i=indice_inicial;i<indice_final;i++){
76                  fprintf(pipe,"plot␣'%s'␣index␣%i␣using␣1:2␣w␣l,␣'%s'␣index␣%i␣using␣1:2\n",
                        name,2*i,name,2*i+1);
77                  fflush(pipe);
78                  getchar();
79                  }
80          }
81      }
82
83
84
85      if(seleccion2==0){
86              fprintf(pipe,"set␣yrange␣[%lf:%lf]\n",y_min/factor_norm,y_max/factor_norm);
87          if(seleccion1==0){
88                  for(i=indice_inicial;i<indice_final;i++){
89                  fprintf(pipe,"plot␣'%s'␣index␣%i␣using␣1:($2/%lf)␣w␣l,␣'%s'␣index␣%i␣using␣
                        1:($2/%lf)\n",name,2*i,factor_norm,name,2*i+1,factor_norm);
90                  fflush(pipe);
91                  }
92          }
93          if(seleccion1==1){
94              for(i=indice_inicial;i<indice_final;i++){
95                  fprintf(pipe,"plot␣'%s'␣index␣%i␣using␣1:($2/%lf)␣w␣l,␣'%s'␣index␣%i␣using␣
                        1:($2/%lf)\n",name,2*i,factor_norm,name,2*i+1,factor_norm);
96                  fflush(pipe);
97                  getchar();
98                  }
99          }
100     }
101
102
103
104
105 fclose(pipe);
106     return 0;
107 }
```