UNIVERSIDAD DE ZARAGOZA

GRADO DE FÍSICA

Departamento de Física Teórica

Estudio de la cromatina con modelos sencillos de Física Estadística

Universidad Zaragoza

Estudiante: Joaquín Milían Lope
Director: Pierpaolo Bruscolini

Zaragoza 3 de Diciembre de 2021

Índice

1. Introducción

1.1. ¿La física estadística puede modelar la cromatina?

Todos los seres vivos estamos formados por células, y los eucariotas tenemos dentro de ellas al núcleo, que contiene nuestro material genético. Éste tiene toda la información que concierne a la célula y dependiendo del tipo de célula, tendrá una función u otra y será transcrito[1] para tal fin; dentro del núcleo el material genético se encuentra en forma de cromatina cuando la célula no se está dividiendo.

La cromatina es un polímero complejo, formado por DNA e histonas (un tipo de proteína), y dentro del núcleo permanece compactificada presentando una jerarquía de dominios anidados entre sí (Figura 3b). En la Drosophila Melanogaster, o mosca del vinagre, que tan solo tiene 4 cromosomas. Se observa que a una escala de 10^6 pares de bases, la cromatina se pliega en unas regiones llamadas compartimentos que interactúan fuertemente. Tan sólo existen dos tipos de compartimentos: Los **Tipo A o Heterocromatina** muy empaquetada y sin posibilidad de transcripción; y los **Tipo B o Eucromatina** débilmente empaquetada y con posibilidad de transcripción.

En particular, nos hemos centrado en el estudio de los cromosomas de la Drosophila, sobre los cuales disponemos de resultados experimentales de HiC. [2] Para su estudio, en primer lugar analizaremos, en el resto de esta Introducción, en que consisten los datos HiC, y como los podemos convertir en un mapa de contactos y en una matriz de covarianza. Después de una breve mención de los métodos más comunes para abordar su interpretación, en el Capítulo 2 introduciremos una versión generalizada del **Modelo de Wako–Saitô–Muñoz–Eaton (WSME)**, y nos preguntaremos si es capaz de modelar correctamente los datos HiC. Para ello veremos en qué se basa este modelo, sobre qué observables basar la comparación entre predicciones teóricas y datos experimentales, cómo optimizar los parámetros del modelo para obtener el mejor ajuste a los datos experimentales, utilizando el algoritmo de Simulated Annealing y otros algoritmos de optimización.

En la sección 3, presentaremos los resultados de optimizar los parámetros μ_{ij} (los promedios de las variables estadísticas del modelo) para reproducir las covarianzas c_{ij}, y a qué tipo de parámetros de interacción θ_{ij} esos parámetros se corresponden. Intentaremos ver si existe algún tipo de dependencia de las θ_{ij} según la distancia, e intentaremos averiguar si es posible aproximar los datos Hi-C con las elecciones estandard de los parámetros θ_{ij}, correspondientes a los modelos de Wako-Saito-Muñoz-Eaton y Poland-Scheraga, respectivamente. Finalmente, en la sección 4 trazaremos nuestras conclusiones, delineando los desarrollos futuros que se pueden abordar.

1.2. La estructura de la cromatina y los datos HiC

1.2.1. La cromatina y los cromosomas

Los cromosomas son polímeros[3] complejos y son los responsables de almacenar la información genética de una célula u organismo. Están formados por ADN, proteínas y hasta ARN, otro tipo de ácido nucleico que se diferencia fundamentalmente del ADN en que no tiene Timina y sí Urácilo, ambas bases nitrogenadas.

[1] La Transcripción es un proceso en el cuál se leen las bases que contienen el ADN, para dar lugar al ARN y este se traducirá a proteínas, que son las que expresan los genes

[2] Guido Tiana, comunicación personal.

[3] Un polímero es una molécula formada por unidades repetidas unidas entre sí por un enlace covalente, y que se distribuyen de la forma de una cadena larga

Por otro lado, en la célula podemos encontrar tanto cromatina como cromosomas: estos últimos representan un estado de la cromatina altamente condensado y compactificado, que aparece durante la división celular para que esta pueda tener lugar de una forma ordenada y segura. Sin embargo, durante la interfase celular, la cromatina está en una fase globular relativamente desordenada, ocupando todo el núcleo y presentando un tamaño de $5 - 20\ \mu m$ de diámetro. Las diferentes cadenas correspondientes a cada cromosoma ocupan diferentes "territorios nucleares", regiones en que domina un cromosoma, que de todas formas se solapan parcialmente (es decir, hay más contactos intra-cromosoma que inter-cromosoma).

La necesidad de confinar los cromosomas, que globalmente tienen una extensión del orden del metro, en el volumen del núcleo celular requiere un plegado extensivo y para nada aleatorio, ya que deben dejarse zonas de la cromatina libres para que, durante la transcripción, se pueda acceder a la información genética que contienen. Por tanto, la conformación de la cromatina juega un papel muy importante en la expresión genética, en la cuál su información será interpretada por la célula para producir el ARN, en un proceso llamado **Transcripción**, que es iniciado por la enzima **ARN Polimerasa**.

De hecho existen hasta tres tipos de ARN Polimerasas: La Polimerasa I, II y III.
La Polimerasa II se encarga de sintetizar el ARN mensajero, que es el que la célula utiliza para producir las proteínas que regulan la expresión genética; dichas proteínas se sintetizan en un proceso llamado Traducción.
En concreto, la Transcripción comienza con el ensamblado de la maquinaria transcripcional en el promotor. El promotor es una región especial de la cromatina, que permite a las polimerasas saber donde adherirse. Las proteínas encargadas de determinar estas secuencias son los Factores de Transcripción (TFs); estas se enlazan con el promotor y ayudan a que la polimerasa, que llevará a cabo la transcripción, forme el complejo de iniciación.

Finalizado el proceso de iniciación, se da comienzo al proceso de elongación, en el cuál la ARN polimerasa generará la nueva cadena de ácido ribonucleico o ARN, complementaria a la secuencia de ADN que se leyó. Una vez que la polimerasa alcanza el final de la secuencia de ADN, al encontrar la secuencia de terminación, se le añade una cola poliA al final de la nueva cadena de ARN, llamada así porque es rica en Adeninas.

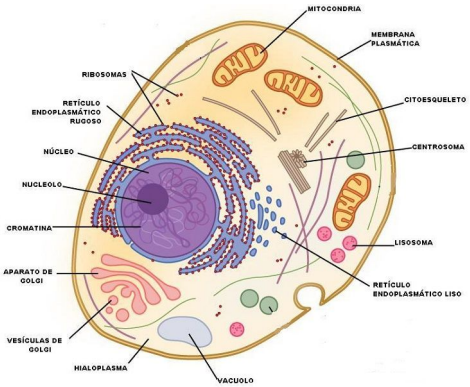
El proceso de Transcripción está fuertemente influenciado por el plegamiento de la cromatina, ya que si una fibra está plegada de manera diferente a la que debería, se dificultará al complejo de transcripción adherirse a la fibra de cromatina. Además, en numerosos genes Eucariotas[4] intervienen reguladores como los **Enhancers o amplificadores**. Los amplificadores son proteínas que pueden estar incluso a una distancia de megabases (10^6) de la secuencia promotor, región donde comienza la transcripción.
Aunque estén tan lejanos del promotor, la cromatina dista de ser una cadena lineal y su configuración nativa permanece plegada, ya que la cromatina puede formar los denominados **chromatin loops**, es decir, la cromatina puede formar una estructura en forma de lazo.

Por lo tanto, el plegamiento genómico afecta a la funcionalidad de la cromatina y también al proceso de transcripción. Esta relación entre plegamiento y su funcionalidad ha sido fuertemente estudiada tanto por técnicas experimentales como por modelos matemáticos y simulaciones.

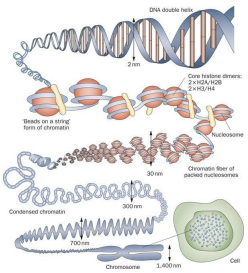
Por otro lado, las razones de utilizar la **Drosophila Melanogaster** y no cualquier otro tipo de ser vivo son las siguientes:

[4]Las células se clasifican en dos tipos: Procariotas, para organismos bacterianos; o Eucariotas, para organismos más complejos como el ser húmano.

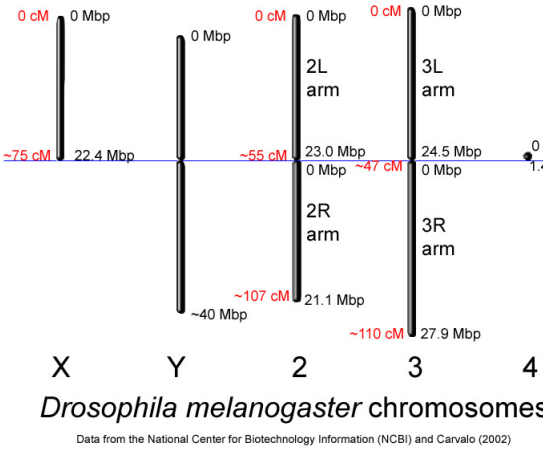
- Su genoma es pequeño y permite obtener una alta resolución a bajo coste, ya que tiene un par de cromosomas sexuales (X e Y) y tres autosomas o cromosomas no sexuales (2,3,4); además se ha secuenciado su genoma completo que consta de 132 millones de pares de bases o 13.767 genes.
- Es fácil de obtener en laboratorio.
- El plegamiento tridimensional de su cromatina, está bien caracterizado en fases de desarrollo temprano.



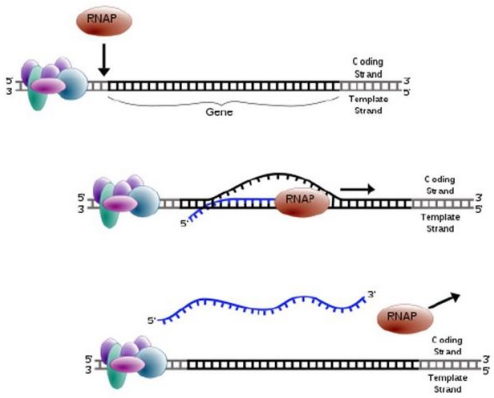
(a) Célula Eucariota



(b) Representación de la cromatina y los cromosomas



(c) Cromosomas de la Dropsophila Melangaster



(d) Transcripción celular

Figura 1: a) Imagen de una célula Eucariota donde podemos ver que la cromatina se sitúa en el núcleo. **b)** Representación del plegamiento que sufre el ADN, la cromatina se compactifica aún más hasta formar los cromosomas. **c)** Representación de todos los cromosomas de la Dropsophila Melangaster, donde se observan los cromosomas sexuales X e Y y los cromosomas no sexuales. **d)** Representación del proceso de transcripción celular

1.2.2. Intepretación de los datos HiC

Actualmente, en el ámbito de la investigación aplicada al plegamiento de la cromatina, destaca el método experimental de Captura de las Conformaciones de los Cromosomas o triple C **(3C)**.
Esta técnica es tan utilizada porque consigue una alta resolución y unos mapas de frecuencias de contacto entre diferentes posiciones genómicas muy amplios (3).
El método 3C consiste en convertir la cercanía de las pares de bases o cadenas de cromatina en matrices de interacción, siendo cada componente proporcional a la fracción de células donde dos fragmentos fueron encontrados para un cierto rango de entrelazado. En particular, Hi-C lee todos los contactos a la vez, gracias a esto es capaz de cuantificar las interacciones entre todos los posibles fragmentos de la cromatina a la vez , estableciendo una puntuación para dichos contactos con una frecuencia de contacto virtual (2).

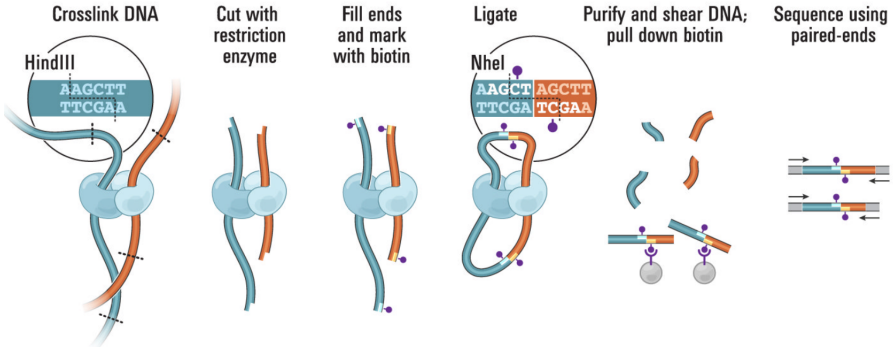


Figura 2: Esquema de la técnica de HiC (4)

En la Figura 2 podemos ver una descripción rápida sobre en que consiste el protocolo Hi-C. Concretamente, primero se 'congelan' las interacciones, es decir, se debe conseguir que las interacciones entre los fragmentos de cromatina (los loci) y las proteínas se mantengan en el transcurso del proceso, para ello se utiliza el Formolaldehído, que consigue fijar a la cromatina y a las proteínas. El siguiente paso es 'digerir' la cromatina mediante el uso de una enzima: el resultado son fragmentos cuyas terminaciones están llenos de Biotina, una pequeña molécula que no afecta a las funciones biológicas y por ende no altera los resultados. Después, se deja a los fragmentos interactuar libremente en condiciones extremadamente diluidas y así producir moléculas de mayor tamaño. Acto seguido, se purifica el ADN, esta vez por la extracción de la Biotina (3).

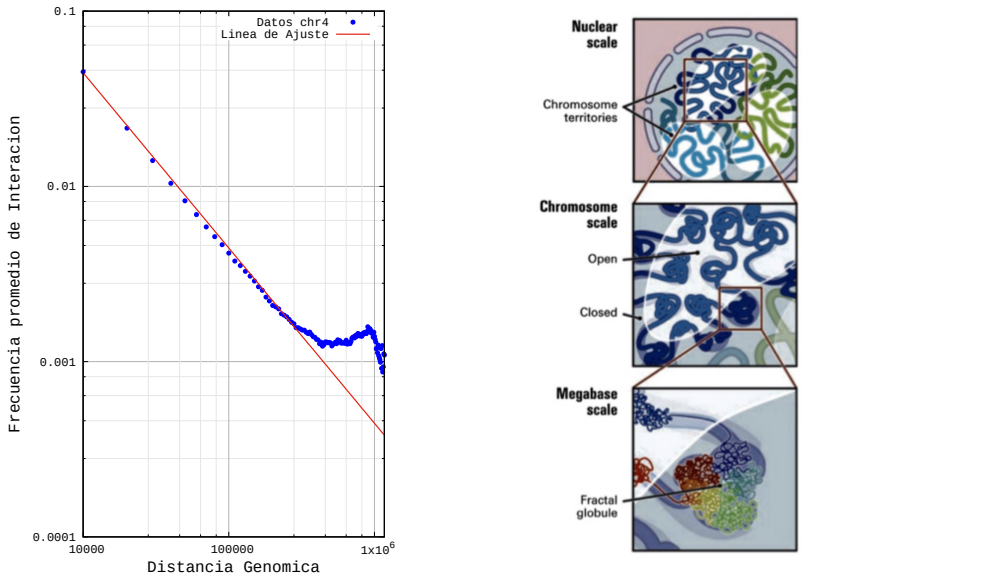
La resolución que consigue la técnica Hi-C está limitada por la resolución de la enzima que se utilice, aunque se han desarrollado otro tipo de protocolos que consiguen mejor resolución en regiones diana mediante el enriquecimiento de las zonas de ligación (5).
En la sección2 detallaremos los procedimientos de normalización, filtrado para obtener una matriz de contactos manejables y veremos cálculo de las matrices de covarianza, que será el punto de partida para nuestro algoritmo.
Wolphram Matematica será el software que utilizaremos para el trato de datos HiC; con él también podemos representar las matrices como mapas de contacto, donde cada píxel se identifica como una componente (i,j) de la matriz que estemos representando.

A continuación hablaremos de las características que tienen los mapas de contacto que obtenemos de los datos Hi-C y la información que obtiene de ellos.
Los mapas de contacto pueden visualizarse como un mapa de Calor (figura 4d), sin embargo su interpretación no es cosa sencilla, ya que pueden solaparse patrones distintos debido a las

diferentes características de la cromatina, lo que dificulta el análisis.
En particular los principales patrones y características que pueden obtenerse de los datos Hi-C son:

1. Ratio de Interacción Cis/Trans: Se trata de patrones a escala genómica y que tienen forma de bloques cuadrados con una gran interacción, por tanto, su color es más oscuro, a lo largo de la diagonal principal (Figura 4a). Este patrón confirma la evidencia experimental de que los loci tienden a interactuar más con otros loci del mismo cromosoma que con otros loci de cromosomas distintos.

2. La frecuencia de interacción depende de la distancia: Se observa una disminución gradual de la frecuencia de interacción cuanto más nos alejamos de la diagonal (hablamos a escala genómica). En la Figura 3a, que fue realizada representando el promedio de la frecuencia de interacción en función de la distancia génomica y que fue ajustada y representada en escala logarítmica con Gnuplot, muestra una dependencia lineal.
Una posible explicación es que los loci que están próximos en la cadena lineal genómica, interactuarán más que aquellos que estén más alejados. Esto está de acuerdo con el decaimiento de la frecuencia de interacción con la distancia, que se predice de un modelo de polímero lineal : $p(x, y) = A \cdot dist(x, y)^{-3/2}$ y con la mayoría de modelos de polímeros.
Sin embargo, la existencia de otros patrones locales complican la estructura del polímero y hacen que la cromatina se aleje de ser descrita totalmente como un polímero ideal.



(a) Ajuste a un Polímero lineal **(b)** Jerarquización de la cromatina

Figura 3: a) Ajuste de un polímero lineal, obteniendo $p(x, y) = 445{,}008 \cdot dist(x, y)^{-3/2}$
b) Representación de la jerarquización de la cromatina

3. Patrón de tablero de Ajedrez: Es un patrón de interacción específico que aparecn en la región de las Mb (Millones de Bases) y que literalmente adopta la forma de un patrón de tablero de Ajedrez, alternando bloques de alta y baja frecuencia de interacción, como vemos en la Figura 4b

Por otro lado, el hecho que los mapas Hi-C nos arrojen este tipo de patrones y características

es una evidencia de que las interacciones entre dos loci no es uniforme, ni mucho menos aleatoria, por el contrario la cromatina está bien organizada en regiones específicas a diferentes escalas.

Por tanto, estos hechos refuerzan la evidencia de que la cromatina se pliega jerárquicamente en una serie de dominios de interacción anidados unos a otros (6) (Figura 3b)dando lugar a:

• **Grandes territorios cromosómicos**, que ocupan regiones específicas dentro del núcleo de la célula y que son estructuras muy dinámicas (7).

• **Compartimentos**: Se trata de las estructuras observadas a una escala de 500kbp a 1Mbp y que tienden a interactuar más fuertemente entre sí, si son compartimentos del mismo tipo y tienden a excluirse si son de diferentes tipos.

Gracias a la existencia de patrones de tablón de ajedrez, se ha podido identificar dos tipos de compartimentos: **Tipo A** que están muy asociados a cromatina abierta, accesible y que permite la transcripción (**Eucromatina**) y los **Tipo B** que son compartimentos con una densidad de compactificación mayor (4).

• **Topological Associated Domains (TADs)**: Son regiones de una gran frecuencia de contactos intradominio y de reducidos contactos interdominio y aparecen a una escala de 70kbp (Figura 4c).
También son fundamentales en la regulación de genes y actualmente hay una creciente evidencia de que son formados por extrusión activa de bucles de cromatina por la acción de la Cohesina (10)

• **Lazos:** Son estructuras que pueden hacer de puente entre los sitios de unión del ADN y el factor de transcripción CTCF (8). Se identifican en los mapas de contacto cuando se ve un incremento de la probabilidad de contactos a escalas muy pequeñas.

1.3. Modelado teórico de la cromatina: estado del arte

El modelado del plegamiento de la cromatina, es un problema que se puede estudiar de diversos puntos de vista, uno de ellos es mediante modelos de dinámica molecular.
La dinámica molecular consiste en simular la molécula de cromatina en su totalidad. Sin embargo, modelar una molécula de cromatina átomo a átomo tendría un coste computacional enorme, por esa razón se simplifica la molécula de cromatina como una unión de subunidades (cada una correspondiente a un segmento de ADN cuyo tamaño es, en orden de magnitud, de miles o decenas de miles de pares de bases) sometidas a diversas interacciones entre subunidades (fuerzas internas), a un baño térmico (fuerzas estocásticas) y quizá alguna fuerza externa; a este método se le conoce como **Simulación de Dinámica Molecular de Grano Grueso** La dinámica del modelo vendrá descrita por la ecuación de Langevin[5], que indica que un proceso estocástico no guarda memoria con el paso anterior:

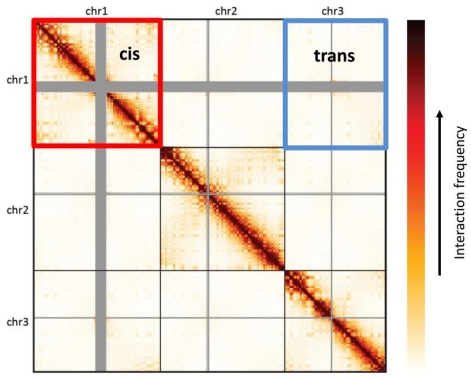
$$m\frac{d^2\vec{r}}{dt^2} = -\zeta\frac{d\vec{r}}{dt} + \xi(t) - \vec{\nabla}V(\vec{r}) \tag{1}$$

y la dinámica de cada subunidad se calcula mediante los algoritmos típicos de las ecuaciones diferenciales (Runge-kutta, Euler, etc) que son algoritmos iterativos capaces de resolver la ecuación diferencial de Langevin discretizando el movimiento, es decir, dividiéndolo en fragmentos en los que se calcula la posición y velocidad de cada subunidad y que servirán como input para la

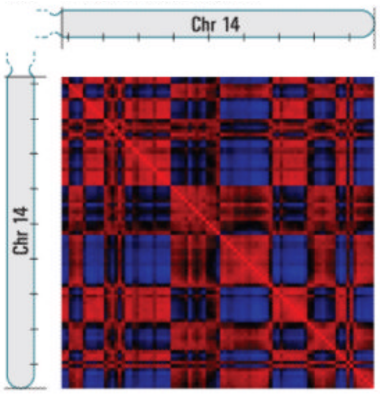
[5]Mirando a lado derecho de la igualdad, el primer término es un termino de carácter disipativo, el tercero se refiere a las fuerzas conservativas y el segundo es el termino estocástico para el baño térmico, debe cumplir que $\langle\xi(t)\rangle = 0$ y $\langle\xi(t)\xi(t')\rangle = 2\zeta k_B T\delta(t-t')$

posición y velocidad del fragmentos a un tiempo posterior. Cabe la pena destacar que la precisión de la simulación está relacionada con la cercanía temporal (Δt). Por ejemplo una simulación con un $\Delta t = 0{,}01s$ será más precisa que una de $\Delta t = 0{,}1s$ pero el tiempo de simulación aumentará.

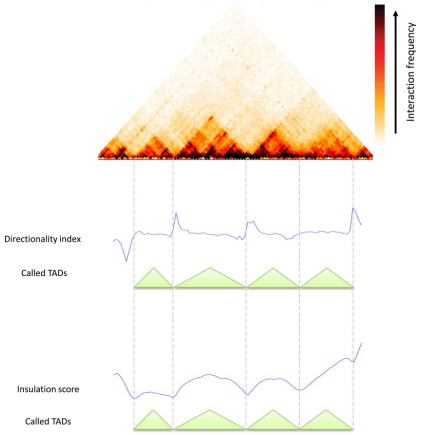
Las simulaciones de dinámica molecular, incluso después de la reducción al 'Grano grueso', tienen un elevado coste computacional; por ejemplo, si queremos simular la cromatina de un cromosoma de 40 Mbp a una resolución de 80 kpb, tendríamos un sistema de 500 partículas y 3000 grados de libertad [6].
En alternativa a estas simulaciones, decidimos utilizar otro enfoque más estadístico, en particular usaremos el **Modelo de Wako-Saito-Muñoz-Eaton**, que ha dado buenos resultados en el pasado en proteínas y puede resolverse de manera exacta.



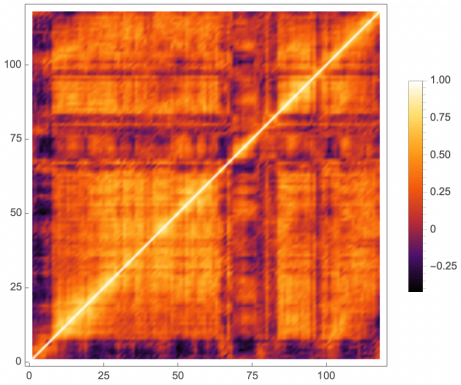
(a) Patrón Cis-Trans



(b) Patrón de Ajedrez para la matriz de correlación



(c) Detección de TADs con diferentes algoritmos



(d) Mapa de Calor de Cromosoma 4 de la Drosophila Melangaster

Figura 4: a) Representación del Patrón Cis-Trans. (9) **b)** Mapa de contactos de patrón de ajedrez (4), usando la matriz de correlaciones. **c)** Detección de TADs con diferentes algoritmos (9) **d)** Mapa de calor para la matriz de covarianza del cromosoma 4 de la Drosophila Melangaster

[6] En un espacio tridimensional cada partícula tiene que ser descrita por 3 coordenadas espaciales y 3 velocidades, una por cada dirección espacial.

2. Métodos

Hamiltoniano eficaz del modelo generalizado Partimos de un hamiltoniano general, de la siguiente forma.

$$H(m) = -\sum_{i=1}^{N}\sum_{j=i}^{N}\theta_{ij}\prod_{k=i}^{j}m_k \tag{2}$$

donde $m_k = 0, 1$ son variables que especifican el estado de los N monómeros del polímero lineal, y θ_{ij} son parámetros posiblemente dependientes de la temperatura (es decir, la $H(m)$ representa en realidad una energía libre)).

Según la interpretación que demos a las m_i y θ_{ij}, este hamiltoniano comprende diversos modelos, como por ejemplo el modelo de **Poland-Scheraga** para la desnaturalización del ADN, y el de Wako-Saito-Muñoz-Eaton, para el plegamiento de proteinas. El primero se describe por el siguiente Hamiltoniano:

$$H(m) = \sum_{i=0}^{N}(a_i + b_i m_i) + T\sum_{i=0}^{N}\sum_{j=i+1}^{N+1} f_{ij}(1-m_i)(\prod_{k=i+1}^{j-1} m_k)(1-m_j) \tag{3}$$

donde $m_i = 0$ indica que la pareja de nucleótidos en la posición i en cada hebra de ADN, están acoplados y formando la doble hebra de ADN, mientras que $m_i = 1$ indica que este acoplamiento está roto, y en la posición i se forma una 'burbuja' o 'lazo' donde las dos hebras están desacopladas. Asumiendo condiciones de contorno $m_0 = m_{N+1} = 0$ y $a_0 = b_0 = 0$, el primer sumatorio representa el termino energético del acoplamiento entre los dos nucleótidos en la posición i y el segundo, el termino entrópico de tener un 'lazo' que empieza en $i+1$ y acaba en $j-1$: $m_{i+1} = m_{i+2} = ... = m_{j-1} = 1$ y $m_i = m_j = 0$; este lazo tiene un coste entrópico f_{ij} para cerrarlo. Es importante destacar que en este modelo no hay interacción entre pares de bases en posiciones diferentes, ni siquiera entre primeros vecinos i, $i+1$: eventuales interacciones de este tipo quedan difuminadas en una redefinición del valor de los 'campos externos' a_i, b_i.

Este modelo será utilizado más adelante para estudiar la cromatina a través de los parámetros θ_{ij}.

Otro modelo de la forma Ec. 2 es el modelo de **Wako-Saito-Muñoz-Eaton** de plegamiento de las proteínas, que describe a una proteína de N aminoácidos. En este caso la variable binaria m_i corresponde con los estado: **Nativo** si $\mathbf{m_i = 1}$ y **Desplegado** si $\mathbf{m_i = 0}$. El hamiltoniano del modelo WSME es:

$$H(m) = -\sum_{i=1}^{N}\sum_{j=i}^{N}\theta_{ij}\left(\prod_{k=i}^{j}m_k\right) - k_B T\sum_{i=1}^{N} q_i(1-m_i) \tag{4}$$

El primer termino es el termino energético y nos dice que se consideran las interacciones i, j solo dentro de una 'isla' de monómeros consecutivos todos en el estado nativo. El segundo termino es el termino entrópico, que es mayor a mayor cantidad de aminoácidos $m_i = 0$ en la cadena, es decir, cuanto más desnaturalizada sea la proteína; los parámetros q_i representan el coste entrópico para ordenar el i-ésimo aminoácido.

En todos los casos, el hamiltoniano generalizado Ec. 2 define una probabilidad de Boltzmann asociada a cualquier estado $\{m_k, k = 1, \ldots, N\}$ del polímero: $p = Z^{-1}e^{-\beta H(m)}$, donde Z es la función de partición. Así, para región i, j podemos definir las cantidades $\mu_{ij} = \langle\prod_{k=i}^{j} m_k\rangle$, que se calculan realizando el promedio estadístico sobre todas las configuraciones posibles con $m_k = 1$ desde $k = i$ hasta $k = j$ pesadas con la probabilidad de que esa configuración ocurra.
Es importante destacar que los parámetros μ_{ij} quedan univocamente y analíticamente determinados a partir de las θ_{kl} (12), y con aquellos podemos calcular las covarianzas $c_{ij} =$

$\langle m_i m_j \rangle - \langle m_i \rangle \langle m_j \rangle$ a través del cálculo de los momentos $\tilde{c}_{ij} = \langle m_i m_j \rangle$ (12) de la siguiente manera:

$$\tilde{c}_{ij} = \langle m_i m_j \rangle = \mu_{ij} + \begin{cases} 0 & \text{si } j \leq i+1 \\ \sum_{n=i+1}^{j-1} \frac{(\mu_{i\,n-1} - \mu_{i\,n})(\tilde{c}_{jj} - \tilde{c}_{nj})}{1 - \tilde{c}_{nn}} & \text{si } j > i+1 \end{cases} \tag{5}$$

Por lo tanto

$$c_{ij} = \langle m_i m_j \rangle - \langle m_i \rangle \langle m_j \rangle = \tilde{c}_{ij} - \mu_{ii}\mu_{jj} \tag{6}$$

por lo que es la relación entre la matriz de covarianza experimental y los parámetros μ_{ij}.
Nuestra estrategia consistirá en proponer una matriz de parámetros μ_{ij} dentro del modelo generalizado, a partir de la cuál calcularemos la matriz de las covarianzas c_{ij} y la compararemos con la matriz de covarianza experimental. Una vez que obtengamos al matriz más parecida posible a la matriz experimental interpretaremos los parámetros μ_{ij} y θ_{ij} correspondientes desde un punto de vista biológico (Sección 3). Pretratamiento de los datos HiC para su comparación con el modelo teórico Nuestros datos de partida, que se refieren a los datos Hi-C de los genomas de la Drosophila Melanogaster, pueden visualizarse en forma de matriz, de tal manera que elemento (i,i+k) de la matriz nos dice el número de contactos medidos en laboratorio entre los monómeros i e i+k; este elemento pertenece a la diagonal $(i, i+k)$, es decir, a la diagonal que está k pasos a la derecha de la diagonal principal (i, i). Es importante destacar que, dependiendo de la resolución de los datos, el elemento (i,i+k) representará una distancia genómica u otra.

Por ejemplo en el caso de los datos que hemos analizado, cada monómero representaba a 10000 pares de bases, luego un elemento a distancia uno de la diagonal, está a una distancia de 10Kbp. A continuación explicaremos los pasos a seguir para obtener la matriz de covarianza y usaremos como ejemplo el caso de los datos del cromosoma 4 de la Drosophila.

En primer lugar, vamos a extraer la submatriz que comienza por la tercera fila y que acaba en la fila número 120, y que empieza a partir de la tercera columna y que acaba en la columna número 120, en la Figura 5 se pueden aprecia esta selección.
La razón de está reducción viene de que los términos de fila o columna menores que 3, son comparables con la resolución del proceso de lectura y términos de fila y columna mayores que 120 probablemente representan regiones mal ensambladas o repetidas, por lo que estas medidas entorpecerían el estudio (11).

En segundo lugar, para reducir el sesgo se eliminan los elementos diagonales, poniéndolos a cero, ya que esos elementos contienen interacciones dentro del mismo "monómero", y representan un factor de confusión a la hora de investigar la estructura a gran escala de la cromatina. En tercer lugar, normalizaremos cada elemento con el promedio de contactos a su misma distancia. Concretamente dividiremos un elemento (i, j), con $j > i$, entre el promedio de los elementos pertenecientes a su misma diagonal:

$$K_{i\,i+d} = \frac{Red_{i\,i+d}}{\frac{1}{N-d}\sum_{k=0}^{N-1-d} Red_{k\,k+d}} \tag{7}$$

siendo $d = 1, ..., N$ el índice que denota la diagonal o la distancia a la diagonal principal; K_{ij} la matriz de contactos[7], y Red_{ij} la submatriz que resulta de los datos experimentales y de anular la diagonal. La normalización trata de compensar el decaimiento de número de contactos con la distancia entre monómeros, como evidenciado en la Figura 3a, para evitar que los cálculos siguientes sean dominados por los contactos a distancia corta, que son más probables sólo por la naturaleza de polímero lineal, y no dan cuenta de una real importancia funcional o energética de la interacción.

[7]En la práctica, usamos la versión simetrizada $(K + K^T)/2$ de esa matriz, para evitar problemas numéricos

Siguiendo el protocolo introducido en (11), en último lugar, se calcula la matriz de covarianza (que llamaremos C^{exp}) de la matriz de contactos anterior, de tal manera que su elemento (i,j) representa la covarianza (Figura 4d) entre los vectores columna i y j de la matriz de frecuencias de interacción anterior.Este paso es necesario porque los datos de contactos originales son de difícil interpretación desde el punto de vista de un modelo de física estadística: no se pueden interpretar como el mapa de contactos asociado a una estructura tridimensional dada, a los cuales asociar una energía, ya que no proceden de una única estructura, sino del promedio de observaciones sobre un conjunto de estructuras diferentes de la cromatina contenida en células diferentes. Por lo tanto, esos mapas recogen contactos que hasta pueden ser incompatibles entre ellos si pensados dentro de una estructura individual. En este contexto, el calcular la covarianza entre vectores columnas de la matriz de frecuencias de contactos equivale a preguntarse que monómeros comparten en promedio los mismo vecinos de interacción, estableciendo una correlación estructural entre ellos.

Gracias al software Wolphram Mathematica podemos implementar, visualizar de una forma fácil y efectiva todos estos pasos, y calcular los resultados de una manera sencilla.
La matriz de covarianza final será nuestra referencia para comparar con aquella predicha por el modelo WSME generalizado, y optimizaremos los parámetros del modelo, utilizando el algoritmo de Simulated Annealing", para minimizar la diferencia entre las dos matrices.

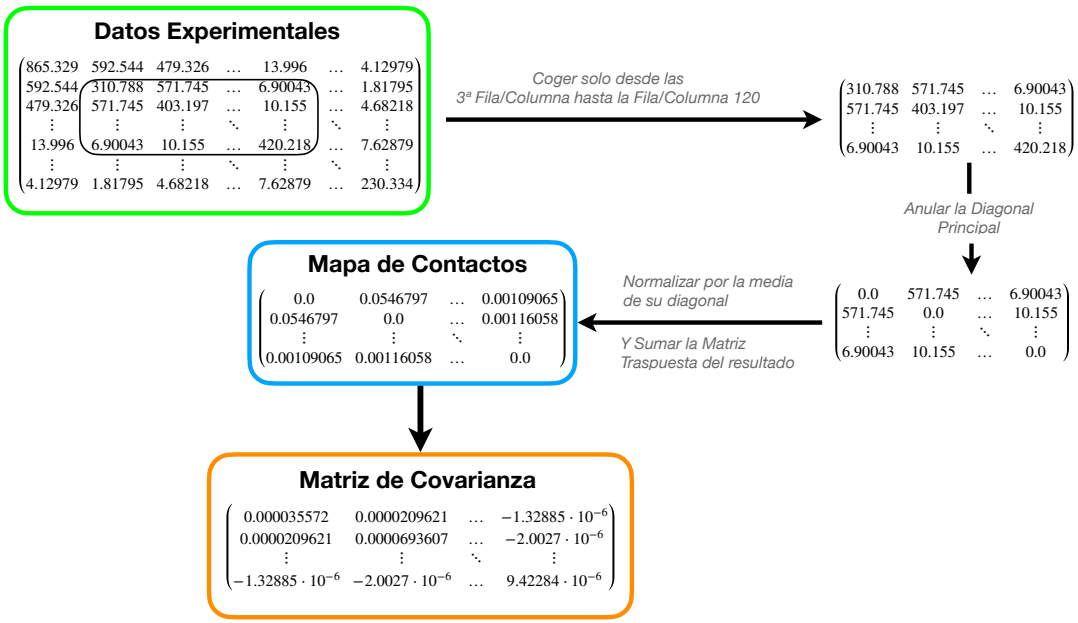


Figura 5: Procedimiento de normalización y de calculo de la matriz de covarianza, primero reducimos la matriz para evitar errores y medidas erróneas,luego normalizamos por la media de contactos a su misma distancia genómica, nos aseguramos que la matriz sea simétrica y finalmente se calcula la matriz de covarianza.

2.1. Ajuste de los parámetros del modelo con Simulated Annealing

Una vez que hemos conseguido la matriz de covarianza a partir de los datos experimentales, el paso siguiente es encontrar los valores de los parámetros del modelo general Ec. 2 que mejor permiten reproducir esa covarianza experimental con la covarianza del modelo Ec. 6, siendo conscientes que nada garantiza que las covarianzas experimentales se puedan reproducir exactamente con ninguna parametrización del modelo, así que nos proponemos minimizar una oportuna distancia entre la covarianza experimental y teórica. La elección de esta distancia, que usaremos como 'Energía' a minimizar con el algoritmo de Simulated Annealing, no es única: nosotros hemos experimentado con

$$E(C, C^{exp}) = \sum_{i=0}^{N-1} \sum_{j=i+1}^{N-1} (C_{ij} - C_{ij}^{exp})^2 \tag{8}$$

donde cada elemento tiene el mismo peso, así que es esperable que los elementos con $j - i$ pequeños, que naturalmente serán más grandes, dominen el cálculo de la energía. También hemos probado una versión que pretende "homogeneizar"las contribuciones de parejas a distancias distintas en el polímero:

$$E(C, C^{exp}) = \frac{2}{N(N+1)} \sum_{i=0}^{N-1} \sum_{j=i}^{N-1} \left(\frac{|C_{ij} - C_{ij}^{exp}|}{|C_{ij}| + |C_{ij}^{exp}|} \right)^2 \tag{9}$$

En cuanto a los parámetros a optimizar, en lugar de trabajar con las θ_{ij}, nos centramos en los parámetros $\mu_{ij} = \langle \prod_{k=i}^{j} m_k \rangle$ del modelo, ya que, estando limitados entre 0 y 1, en principio resultan más cómodos de utilizar en el marco de un algoritmo de optimización. Sin embargo, por su definición, estos parámetros no son independientes sino que están sujetos a las siguientes condiciones (12):

$$\begin{cases} Tipo\ 1: & 0 < \mu_{jj} - \mu_{j-1\ j} < 1 - \mu_{j-1\ j-1} & \text{si } 0 < j \leq N-1 \\ Tipo\ 2: & 0 < \mu_{i+1\ j} - \mu_{i\ j} < \mu_{i+1\ j-1} - \mu_{i\ j-1} & \text{si } 0 < i+1 < j \leq N-1 \\ Tipo\ 3: & 0 < \mu_{0\ j} < \mu_{0\ j-1} & \text{si } 0 < j \leq N-1 \end{cases} \tag{10}$$

Con estos ingredientes, el algoritmo de **Simulated Annealing** (SA) ((**?**) que utilizamos consta de los siguientes pasos:

1. crear una matriz inicial μ_{ij} que satisface los vínculos;
2. calcular la energía asociada a esa matriz;
3. fijar una temperatura inicial T_0 para el algoritmo de SA. Nosotros escogemos $T_0 = max(C^{exp})$ que nos garantiza que la mayoría de cambios de Monte Carlo se aceptarán;
4. escoger de forma aleatoria y uniforme un elemento μ_{ij} y proponer un cambio que satisfaga los vínculos
5. calcular la nueva energía; aceptar el cambio con el criterio de Metropolis, es decir: si $e^{-(E_{new}-E_{old})/T} \geq \omega$ siendo ω un número aleatorio plano entre 0 y 1, se acepta el cambio (esto implica que si E_{new} es menor que la energía anterior E_{old} el cambio se acepta siempre).
6. después de un cierto números de iteraciones n_{iter} , durante las cuales se registra el número de cambios aceptados, bajar la temperatura. En nuestro caso $n_{iter} = 3000\ sweeps$ [8] Nuestra elección para el enfriamiento es dada por $T_{new} = \alpha T_{old}$, con $\alpha < 1{,}0$ el parámetro que denota la brusquedad a la que se enfría el sistema.

[8] Denominamos *sweep* al número mínimo teórico de cambios necesarios para poder cambiar la matriz entera; como la matriz es simétrica, este número coincide con el número de elementos en la parte triangular superior, $N(N+1)/2$ con N el tamaño de la matriz.

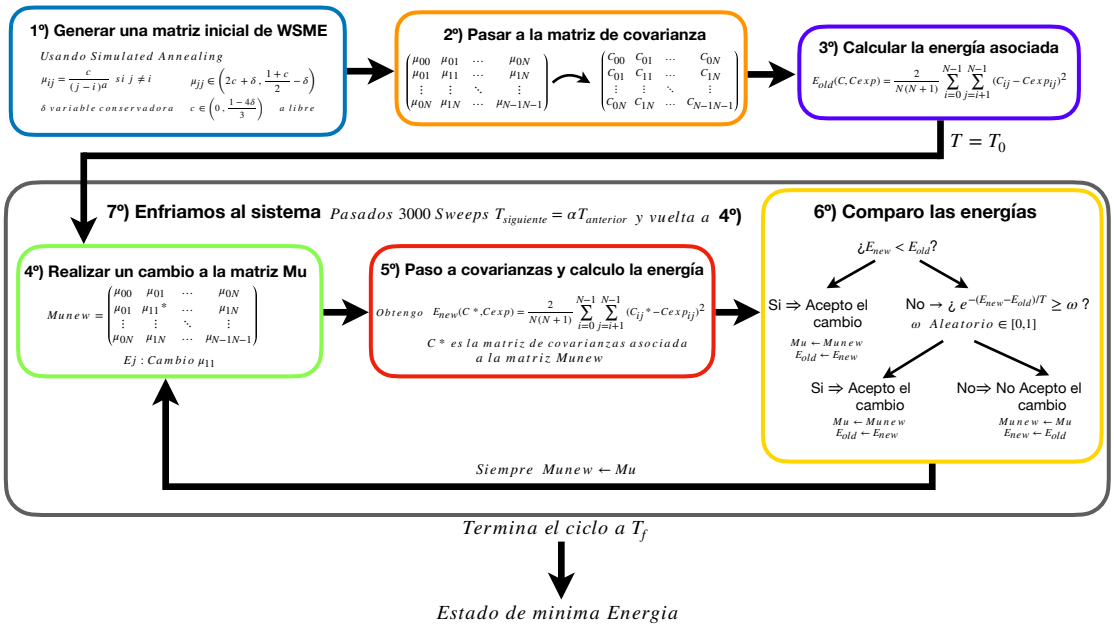


Figura 6: Descripción del procedimiento de Simulated Annealing utilizado.

Como llegar a energía 0 (igualdad de la matriz del modelo y la experimental) no es posible, se busca enfriar hasta una temperatura tan baja que los cambios que se acepten sean muy pocos o que la energía se mantenga constante en un número elevado de ciclos de enfriamiento, lo que se considerará como el estado de mínima energía del sistema.
Por el contrario, al inicio nos interesa tener una temperatura elevada, para que se acepten muchos cambios y el sistema se mueva libremente por el espacio de configuraciones.[9]
Conforme se va disminuyendo la temperatura, los cambios se aceptarán más por ser de una menor energía que por la distribución de Boltzmann, ya que cuanto menor sea la temperatura la distribución de Boltzmann $e^{-\Delta E/T}$ decaerá a cero cada vez a una menor ΔE .
También es importante enfriar de manera suave, de esta manera, evitaremos caer en mínimos relativos por el camino y quedarnos atascados en ellos.

Podemos ver que los aspectos relevantes y limitantes para el algoritmo son:

- la **elección de la función energía**, ya que está dictará el aspecto del espacio de fases, ya que puede ser con muchos mínimos relativos y muy abruptos (sistema frustrado) o con pocos mínimos relativos (sistema poco frustrado).
- la **elección del parámetro**α, que determina la velocidad de enfriamiento, siempre cuanto más grande sea α menor probabilidad hay de quedarte atrapado en un mínimo relativo pero mayor tiempo de computación se requiere, por eso hay que llegar a un consenso entre ambos, en nuestro caso elegimos $\alpha = 0{,}75$.
- Por último, la **Temperatura final $\mathbf{T_f}$** que tras realizar unas cuantas pruebas obtuvimos que $T_f = \alpha^{50} T_0$ la aceptancia era lo suficientemente baja y al energía tan estable como para decir que la optimización ha terminado.

Sin embargo, otros aspectos que se han revelado cruciales, han sido los relativos a la creación de una matriz inicial μ y de su actualización en cada paso de Monte Carlo. En más detalle:

- La creación de una matriz μ inicial "legal", es decir, que satisfaga todos los vínculos Ec. 10 se ha relevado más problemática de lo esperado. Nuestro primer intento consistió en generar los elementos $\mu_{ii} \in (0, 1)$ y generar el resto de elementos por diagonales, es decir, desde

[9] La distribución $e^{-\Delta E/T} \approx 1$ para temperaturas elevadas.

la diagonal $j = i + 1$ hasta la diagonal $j = N - 1$ realizando: $\mu_{i\,i+k} = \mu_{i\,i+k-1} \cdot \mu_{i+k\,i+k}$ y finalmente realizar un número elevado de cambios (100 Sweeps de MC) a todos elementos de manera aleatoria y respetando las inecuaciones 10.

El método que adoptamos finalmente consiste en generar una condición inicial más ajustada a la física de polímeros, asumiendo una dependencia de los parámetros μ_{ij} con el inverso de la distancia, es decir, $\mu_{ij} = \frac{c}{(j-i)^a}$ si $j \neq i$; sustituyendo esto en los vínculos 10, se obtienen condiciones sobre los elementos diagonales:

$$c\left(2 - \frac{1}{2^a}\right) < 2c < \mu_{jj} < \frac{1+c}{2} \tag{11}$$

Por lo que $\mu_{jj} \in \left(2c + \delta, \frac{1+c}{2} - \delta\right)$, luego que $c \in \left(0, \frac{1-4\delta}{3}\right)$, con δ un número que denota lo conservador que quieres ser con las ecuaciones, en nuestro caso fue $\delta = 10^{-4}$ y a libre[10]. Para encontrar los mejores valores de c, a, μ_{jj} se usa aplica un Simulated Annealing, donde a $T = cte$ se genera primero c al azar en su intervalo y a al azar entre 0 y 0,5, y los elementos mu diagonales se igualan a los elementos diagonales de C^{exp}. Después se entran en 3 bucles anidados donde se hacen 100 cambios a los elementos diagonales y se aceptan según el algoritmo de Simulated Annealing, tras ellos se cambia $a = a \cdot 1,1^{ra}$ con ra un aleatoria en el intervalo $(-1, 1)$, tras completar 100 cambios de a se cambia c de manera similar a a pero respetando que $c < \frac{1-4\delta}{3}$. Tras estos 10000 cambios puesto que a c y a constantes se hacen 100 cambios de los elementos diagonales, se disminuye la temperatura y tomaremos la matriz con menor energía como condición inicial.

- Realizar los cambios de la matriz μ en cada paso de Monte Carlo: Las inecuaciones 10, hacen que un μ_{ij} deba pertenecer a dos intervalos[11] (como máximo), por lo que se debe generar μ_{ij} en el intervalo intersección entre ambos.
  Denotaremos a este intervalo como$(\mu_{ij}^{lower}, \mu_{ij}^{upper})$, lo que en principio no tiene más problema que la ralentización del programa, por tener que calcular los extremos a cada paso. Sin embargo, esta estructura implica un problema difícilmente detectable, sobre todo en el marco de un Monte Carlo con variables reales: llega un punto que los extremos del intervalo son muy cercanos para algunas (o muchas) μ_{ij}, y a veces directamente indistinguibles dentro de la precisión numérica. Esto causa dos problemas que han entorpecido nuestras investigaciones, hasta detectarlos: por un lado, los pasos se vuelven extremadamente locales, y no permiten explorar eficazmente el espacio de configuraciones; por otro lado, la tasa de aceptación, muy buena, sugiere que la exploración es eficaz y no se ha quedado atrapada en un mínimo local. La única alarma de que algo estuviera mal estaba relacionada con que la tasa de aceptación seguía alta incluso a temperaturas muy bajas[12]: lo que en realidad estaba pasando es que el sistema se alejaba de la energía de la configuración inicial lo suficiente como para sugerir una exploración eficaz, y sin embargo exploraba una región de configuraciones limitada, sin importar cuan alta fuera la temperatura inicial, ya que la limitación no procedía de las barreras en energía a superar, sino de la imposibilidad de efectuar grandes cambios. La solución a este problema ha sido, evaluar todos los extremos de generación de todos los μ_{ij} y si $\frac{\mu_{ij}^{upper} - \mu_{ij}^{lower}}{\mu_{ij}^{med}} < \varepsilon$, donde μ_{ij}^{med} es la media entre μ_{ij}^{lower} y μ_{ij}^{upper}, entonces se elimina ese termino (i, j) de los posibles términos a cambiar. En nuestras simulaciones hemos experimentado con $\varepsilon = 10^{-4}$ *y* 10^{-6}.

[10] Ver el Anexo II para saber su deducción

[11] Ver el Anexo I para ver como se deducen según la posición en la matriz.

[12] Si la matriz μ cambia apenas, la variación de energía es prácticamente cero, por lo que $e^{-\Delta E/T} = e^0 = 1$ que es siempre mayor a un aleatorio plano entre 0 y 1, y esto implica que el cambio se acepta.

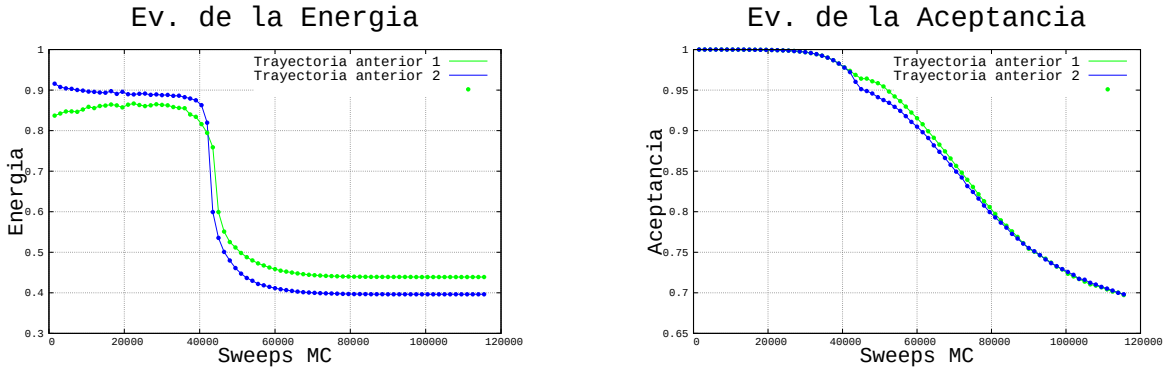
3. Estudio del chr4 de la Drosophila

3.1. Minimización de la distancia entre covarianza experimental y teórica

En este trabajo nos hemos centrado en el cromosoma 4 de *Drosophila melanogaster*, ya que es su cromosoma más pequeño, lo que implica tiempos más cortos de simulación. Para la minimización de la distancia entre la predicción del modelo y el valor de la matriz de covarianza obtenido desde los datos de Hi-C, hemos utilizado el algoritmo de Simulated Annealing, utilizando ambas energías descritas en la sección 2.1.

Los resultados obtenidos con las dos versiones de la energía, ecuaciones 8 y 9, en general no presentan grandes diferencia, aunque la correlación con la definición de compartimentos 'clásica', basada en el análisis de las componentes principales de la matriz de covarianza experimental, proporciona mejores resultados con la elección de la energía más sencilla8, como veremos a continuación. Sin embargo, queremos empezar esta sección de resultados enseñando las gráficas, obtenidas con la energía más homogénea' 9, que finalmente nos hicieron sospechar que el algoritmo 'ingenuo' de Simulated Annealing que estábamos utilizando nos estaba dando problemas, lo que sugiere que probablemente este tipo de algoritmos no sea el más indicado en este tipo de problemas, con variables fuertemente vinculadas.

En Fig. 7 se pueden ver dos trayectoria para la energía Ec. 9, utilizando el algoritmo 'ingenuo', en que sencillamente se elige una pareja i, j con probabilidad uniforme, y se propone un nuevo valor para μ_{ij} dentro de sus límites. La primera trayectoria empieza desde una condición inicial cualquiera, encontrada como explicado en los Métodos, la segunda empieza desde la condición final de otra simulación, y entonces corresponde a su 'recalentamiento'. En ambos casos la temperatura inicial es alta $T = 9,090 \cdot 10^{-6}$, y se va enfriando cada 1500 'sweeps', como explicado en la sección 2. La primera curva (Trayectoria anterior 3) no levanta sospechas: con una temperatura alta, en pocos pasos el sistema abandona la condición inicial y empieza a explorar zonas de alta energía del espacio de configuraciones; al bajar la temperatura, la energía disminuye, hasta un punto en que se realiza un salto brusco a configuraciones con energía más baja.



(a) Evolución de la Energía

(b) Evolución de la Aceptancia

Figura 7: a) Evolución de la Energía para dos trayectorias, que empiezan desde una condición inicial cualquiera, y desde una condición de baja energía identificada en una trayectoria previa. Se dibuja un punto a final de cada intervalo de termalización. **b)** Evolución de la Aceptancia en las mismas condiciones del panel (a).

Sin embargo, lo que nos hizo replantearnos la bondad del algoritmo ha sido la segunda trayectoria: se ve como a partir de una condición que es ya de baja energía, al sistema le cuesta casi 40000 sweeps hasta alcanzar su energía máxima: en este tiempo, la temperatura baja 20 veces según nuestro protocolo. También la gráfica de la aceptancia es sorprendente, ya que hasta a temperaturas que se suponen bajas, hacia el final de las simulaciones, la aceptancia se mantiene en valores muy altos.

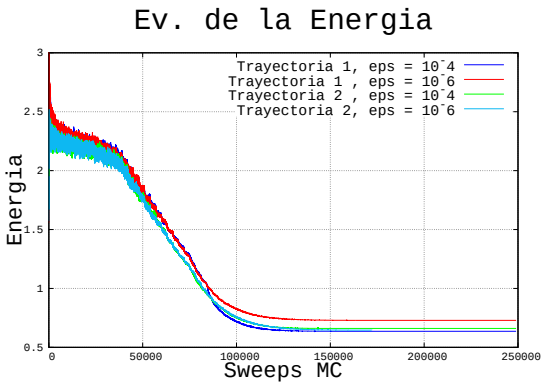
Como mencionado en la sección 2, este comportamiento se explica pensando en que el sistema está en realidad intentando y aceptando pasos microscópicos todo el tiempo, así que le cuesta alejarse de la configuración inicial y aumentar su energía, si esa es baja, lo que implica una mala exploración del espacio de configuraciones,independientemente de la altura y posición de las barreras. Este descubrimiento fue lo que nos hizo plantear el cambio del criterio de elección de los índices de matrices a cambiar en cada paso, que para evitar inflar la aceptancia.

A continuación, tuvimos que rediseñar el protocolo de elección de los elementos ij a cambiar, de acuerdo con lo explicado en la Sección 2.1. El nuevo protocolo contiene un parámetro arbitrario ε, que controla cuándo los dos extremos de variabilidad de μ_{ij} están tan cerca que se pueden considerar iguales, y descartar ese valor de ij desde la tabla de cambios. Nuestras pruebas siguientes fueron hechas con dos condiciones iniciales distintas (Trayectoria 1 y Trayectoria 2) y dos valores distintos del parámetro de control: $\varepsilon = 10^{-6}$ y $\varepsilon = 10^{-4}$.

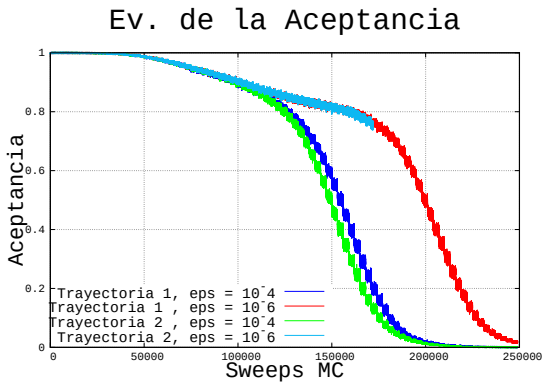
Los resultados para Trayectoria 1:
$\varepsilon = 10^{-4}$, $E = 0,636397$ y aceptancia $0,11\,\%$
$\varepsilon = 10^{-6}$, $E = 0,728849$ y aceptancia $2,18\,\%$

Los resultados para Trayectoria 2:
$\varepsilon = 10^{-4}$, $E = 0,660368$ y aceptancia $0{,}71\,\%$
$\varepsilon = 10^{-6}$, $E = 0,646121$ y aceptancia $74,53\,\%$

Como podemos observar en las Figuras 8a y 8b la energía se estabiliza en cualquiera de las cuatro pruebas y una aceptancia que decae a cero, cabe la pena destacar que en la prueba con Trayectoria 2 y $\varepsilon = 10^{-6}$ fue detenida antes que las otras pruebas ya que la energía apenas variaba conforme se realizaban mayores sweeps de MonteCarlo.



(a) Evolución de la Energía en las diferente pruebas



(b) Evolución de la Aceptancia para las diferentes pruebas.

Figura 8: a) Evolución de la Energía en las diferente pruebas, la Temperatura se reduce un 25 % tras 3000 Sweeps de Monte Carlo. **b)** Evolución de la Aceptancia para las diferentes pruebas.

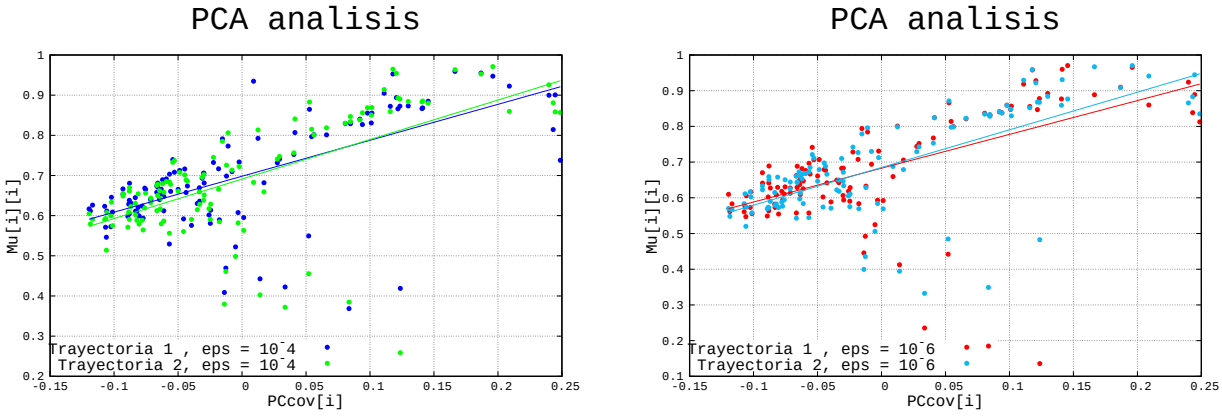
3.1.1. Análisis de covarianza y compartimentos obtenidos

La minimización con el algoritmo de Simulated Annealing nos proporciona los valores de los parámetros μ_{ij}, e indirectamente, los valores los parámetros θ_{ij} del hamiltoniano general Ec. 2, que proporcionan el mejor acuerdo con los datos experimentales. Sin embargo, no nos dan una clave para interpretar qué significan las variables del modelo en lo que se refiere a la estructura de la cromatina, aunque una hipótesis bastante natural sería la de identificar el estado $m_i = 0$ como eucromatina (compartimento A) y $m_i = 1$ como heterocromatina (compartimento B), o viceversa.

Para comprobar si esta hipótesis es cierta se representan los elementos diagonales $\mu_{ii} = \langle m_i \rangle$ en función del autovector principal de la matriz de covarianza experimental.Recordemos que la matriz de covarianza es simétrica, por tanto, podemos descomponerla en una base de autovectores y dicha matriz de covarianza será combinación lineal de los proyectores determinados por los autovectores de los subespacios propios que se obtengan , con unos coeficientes de proporcionalidad iguales a sus autovalores[13], por ende el autovector que más repercute a esta combinación es el de mayor autovalor, que es conocido como **autovector principal(PC)**.

Recordemos también que sobre el signo de las componentes de ese autovector se basa la interpretación de un monómero como perteneciente al compartimento A o B de tal manera que si los parámetros diagonales de nuestro modelo presentan una dependencia lineal con un elevado coeficiente Pearson con la Componente Principal, será un indicio de que el modelo describe correctamente a la cromatina.

La figura 9 reproduce los resultado que obtenemos para diferentes valores de ε



(a) PCA para $\varepsilon = 10^{-4}$ **(b)** PCA para $\varepsilon = 10^{-6}$

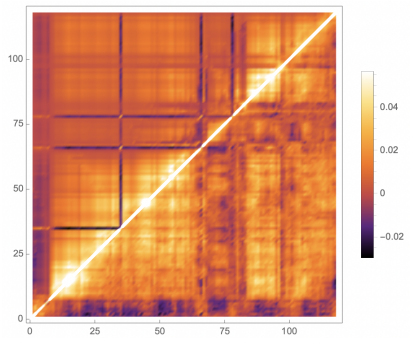
Figura 9: a) PCA para pruebas con $\varepsilon = 10^{-4}$. **b)** PCA para pruebas con $\varepsilon = 10^{-6}$.

Los resultados para Trayectoria 1:	Los resultados para Trayectoria 2:
$\varepsilon = 10^{-4}$, $y = 0,895\ x + 0,6982$ y $\rho = 0,6522$	$\varepsilon = 10^{-4}$, $y = 0,9839\ x + 0,6911$ y $\rho = 0,6469$
$\varepsilon = 10^{-6}$, $y = 0,9473\ x + 0,6829$ y $\rho = 0,5863$	$\varepsilon = 10^{-6}$, $y = 1,055\ x + 0,6849$ y $\rho = 0,7025$

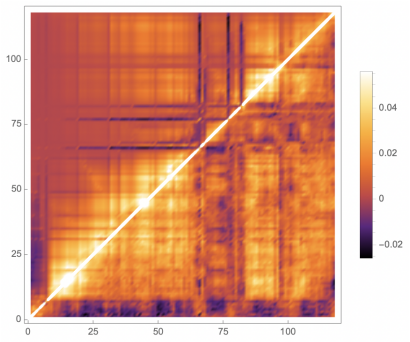
[13]Si los espacios propios no son degenerados, cada proyector se construye a partir de un autovector de su subespacio propio.

Podemos destacar como en todos casos, aparece una tendencia lineal, estropeada sin embargo por unos cuantos "outliers" que bajan el coeficiente de correlación de Pearson $\rho = \frac{Covarianza(\mu_{ii},PC)}{\sqrt{\sigma_{\mu_{ii}}\sigma_{PC}}}$ a unos valores significativos, aunque no excepcionales, que destacan como la interpretación de las μ_{ii} como indicadoras del tipo de compartimento puede estar bien encaminada.

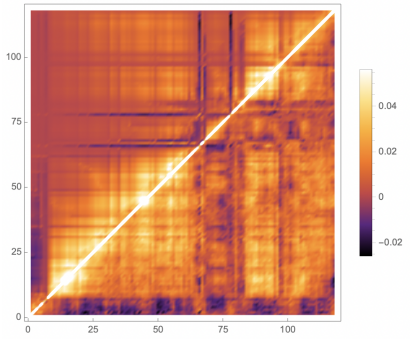
Por otro lado, podemos representar de forma gráfica las matrices de covarianza obtenidas en ambas pruebas junto a la matriz de covarianza experimental, para poder comparar visualmente y valorar qué tipo de acuerdo se corresponde a los valores de energía mínima obtenidos.



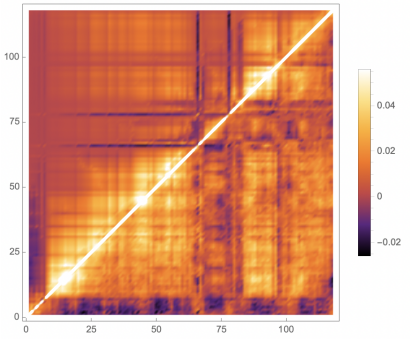
(a) Matriz de Covarianza Trayectoria 1 y $\varepsilon = 10^{-4}$



(b) Matriz de Covarianza Trayectoria 1 y $\varepsilon = 10^{-6}$



(c) Matriz de Covarianza Trayectoria 2 y $\varepsilon = 10^{-4}$



(d) Matriz de Covarianza Trayectoria 2 y $\varepsilon = 10^{-6}$

Figura 10: a) Matriz de covarianza experimental (por debajo de la diagonal) y matriz de covarianza obtenida para $\varepsilon = 10^{-4}$ **b)** Matriz de covarianza experimental (por debajo de la diagonal) y del modelo, con $\varepsilon = 10^{-6}$

La figura 10 evidencia como el modelo reproduce los rasgos principales de los patrones de "tablero de ajedrez"de la covarianza experimental, aunque hay que reconocer que no se logran reproducir todos los detalles, y además aparecen unas líneas muy marcadas de covarianza negativas, que no sabemos interpretar.

3.1.2. Obtención de la Barrera óptima con una curva ROC

La figura 9 sugiere una relación entre μ_{ii} y compartimentos, pero no permite averiguar cuántos monómeros el modelo clasifica correctamente. Para averiguarlo, necesitamos saber cuál sería el criterio para clasificar los valores experimentales en dos compartimentos, como ocurre en la realidad para la cromatina, que existe de dos formas como Eucromatina y como Heterocromatina. El método consistía en tomar la Componente Principal de la matriz de covarianza experimental y clasificar los términos en función del signo dominante, para nuestros datos el

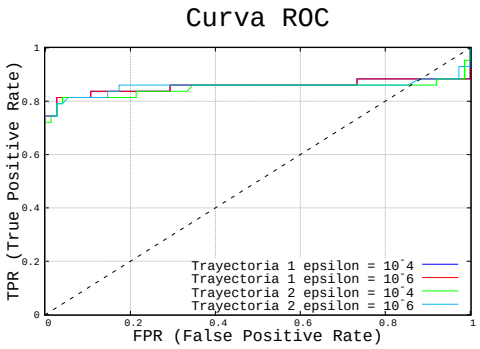
signo dominante es el negativo, por lo que los términos de la componente Principal negativos diremos que pertenecen al Compartimento A (up)y los positivos (minoritarios) al compartimento B ($down$).

Esta clasificación se utiliza como referencia para los valores μ_{ii} de nuestras pruebas y lo que nos dispusimos a hacer es un análisis con una curva ROC (Figura 11a), la forma de trabajar será proponer una barrera entre 0 y 1 que es el rango de los valores μ y calcular la Tasa de verdaderos positivos (TPR) y de Falsos positivos (FPR) con respecto a la referencia de los datos experimentales, de tal manera que la barrera con la mayor $TPR = \frac{TruePositive}{up}$ y menor $FPR = \frac{FalsePositive}{down}$ será la barrera óptima para clasificar los valores teóricos por compartimentos, matemáticamente la barrera optima será aquella con el **máximo coeficiente Youden (J)** $J = TPR - FPR$ o también el **máximo coeficiente de correlación Matthews (MCC)** $MCC = \frac{(TP \cdot TN) - (FP \cdot FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$[14].

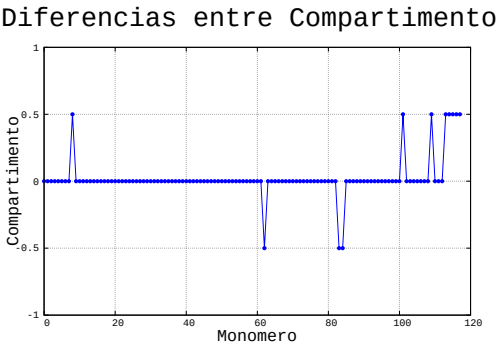
El resultado fue una **Barrera Optima** son los siguientes:

Tabla 1: Umbral de μ_{ii} para la clasificación en compartimentos

Pruebas	Barrera óptima (Youden)	Barrera óptima (MCC)
Trayectoria 1, $\varepsilon = 10^{-4}$	$0,734000$	$0,792000$
Trayectoria 1, $\varepsilon = 10^{-6}$	$0,742000$	$0,742000$
Trayectoria 2, $\varepsilon = 10^{-4}$	$0,737000$	$0,746000$
Trayectoria 2, $\varepsilon = 10^{-6}$	$0,742000$	$0,796000$



(a) Curvas ROC para todas las pruebas realizadas.



(b) Diferencia entre los compartimentos experimental y ROC.

Figura 11: a) Curvas ROC para todas las pruebas realizadas y todas las trayectorias. **b)** Diferencia entre los compartimentos experimental y ROC para la Trayectoria 1 $\varepsilon = 10^{-4}$ para barrera $0,734000$

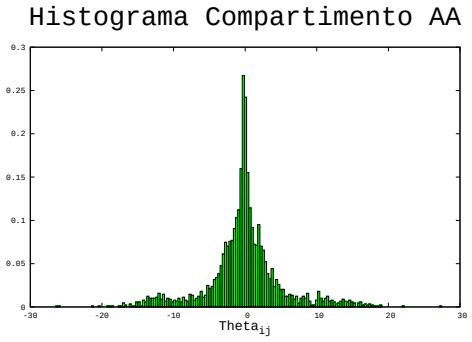
En la figura 11b representamos la semidiferencia entre el vector binario de los compartimentos para la referencia experimental y los compartimentos que dicta la barrera óptima (según el método de Youlen) $0,734000$ para la prueba de la Trayectoria 1 y $\varepsilon = 10^{-4}$. Esta barrera

[14]Si el denominador es nulo entonces $MCC = (TP \cdot TN) - (FP \cdot FN)$,
$TP = TruePositive$,$FP = FalsePositive$,$TN = TrueNegative$ y $FN = FalseNegative$

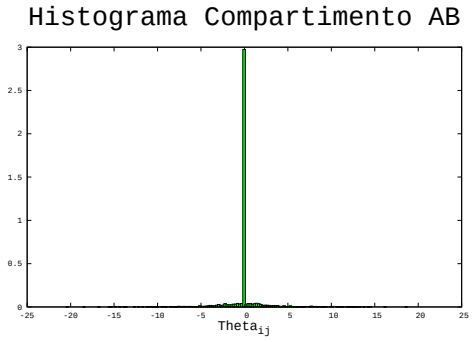
corresponde a un punto $(FPR, TPR) = (0,040000, 0{,}813953)$ de la curva ROC o una tasa de acierto del 81 % y de tasa de falsos positivos del 4 %.

3.1.3. Análisis de los parámetros θ_{ij} obtenidos

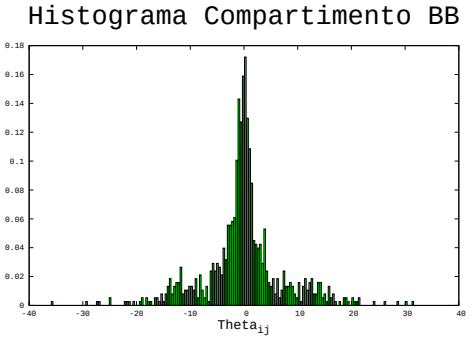
Nuestro algoritmo ha trabajado realizando cambios tentativos sobre los parámetros μ_{ij} y ha comparado con los valores experimentales a través de los covarianzas c_{ij}. Sin embargo, en el marco del modelo de WSME generalizado se pueden obtener exactamente los parámetros de interacción θ_{ij} a partir de los μ_{ij} directamente (12).
En primer lugar, estudiaremos la distribución de los parámetros θ_{ij} en función a que tipo de compartimento pertenece i y a cuál pertenece j. La definición de compartimento se realiza con el autovector principal de la matriz de covarianza experimental; sin embargo, la arbitrariedad del signo del autovector implica que necesitamos un criterio para definir (aunque de forma arbitraria) los compartimentos. Procedemos de la forma siguiente: asociamos los términos con signo mayoritario en el autovector principal, al compartimento A y los de signo minoritario al compartimento B. Esto no resuelve la cuestión si el compartimento A así definido corresponde a la Eurocromatina o a la Hetereocromatina, sin embargo, nos permite una definición unívoca. Esta clasificación nos permite asociar a cada monómero i un compartimento, y así analizar por separado los parámetros θ_{ij} con índice i y j ambos de Compartimento A, ambos de Compartimento B o i de Compartimento A y j de Compartimento B. Los histogramas resultantes son los siguientes:



(a) Histograma de Compartimentos A y A



(b) Histograma de Compartimentos A y B



(c) Histograma de Compartimentos B y B

Figura 12: a) Histograma entre Compartimentos A y A. **b)** Histograma entre Compartimentos A y B. **c)** Histograma entre Compartimentos B y B.

Como se puede observar, los histogramas para una pareja de monómeros en compartimentos diferentes A y B presentan una distribución casi total en torno a un valor levemente negativo. Recordando que el termino energético del hamiltoniano (2) tiene la forma $-\theta_{ij}\prod_{k=i}^{j} m_k$, un valor negativo de θ_{ij} implica que la interacción entre i y j será desfavorable, lo que desaconsejará la presencia de islas de $m_k = 1$ con $k = i, ..., j$. Es decir, las interacciones entre compartimentos distintos aparecen como desfavorables. Para el caso de histogramas entre compartimentos del mismo tipo, las distribuciones no son tan abruptas y están centradas en un máximo negativo, aunque en el caso BB, el pico está más cercano a cero.
Podemos argumentar que como el pico de los histogramas de los tres tipos es negativo, el termino hamiltoniano se puede expresar como $|\theta_{ij}|\prod_{k=i}^{j} m_k$, por lo que en el estado de mínima energía será más favorable tener $m_k = 0$ con $k = i, ..., j$, está argumentación es válida para todos los valores negativos de θ y es la contraria para todo los valores positivos de θ donde el estado de mínima energía más favorable será con $m_k = 1$ con $k = i, ..., j$.

A continuación veremos cuál es la distribución de los parámetros θ_{ij} en función de la distancia genómica. En particular, podríamos esperar que, si trataramos con un homopolímero, donde todos los monómeros son iguales, las energías de interacción dependieran solo de la distancia entre los monómeros $j - i$, y por lo tanto en un histograma para una distancia fija, veríamos un solo pico, que posiblemente se desplaza en histogramas para diferentes distancias $j - i$. En cambio, para un heteropolímero con interacciones completamente aleatorias, podríamos esperar un histograma poblado de muchas barras, distribuidas de forma homogenea en correspondencia de varios valores de la energía.

Finalmente, para un polímero con solo dos tipos de monómeros, A y B, podríamos esperar obtener distribuciones con tres picos, correspondientes a las interacciones AA, AB, BB. El análisis de los valores obtenidos no permite obtener indicaciones muy claras: existe al menos un pico predominante levemente negativo; a partir de distancia 4 se aprecia dos picos a parte del cercano a 0 pero que conforme se va incrementando la distancia, se hacen más difusos, lo que estaría de acuerdo con una dependencia con la distancia; para distancias muy grandes la distribución es muy uniforme .

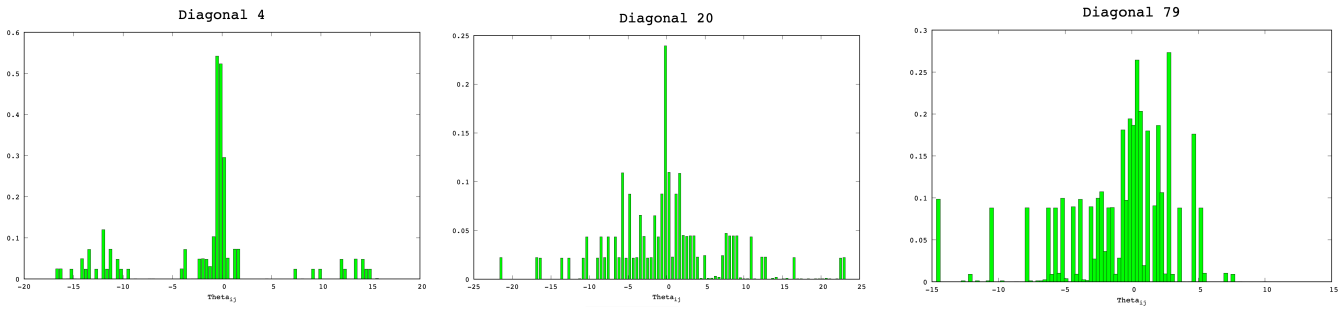


Figura 13: Histogramas según la distancia genómica.

Lo que no podemos estimar en ninguno de estos casos es la función distribución de los parámetros θ_{ij} porque no tenemos suficientes datos al ser la matriz θ relativamente pequeñas.

3.2. Ajuste de la covarianza experimental con un modelo de Poland-Scheraga o WSME

Los resultados de la sección anterior no permiten interpretar fácilmente el significado de los valores de las interacciones θ_{ij} en términos de las interacciones entre dos tipos de monómeros. Para investigar ulteriormente este asunto, por último realizamos un ajuste de la covarianza

experimental con modelos que tienen exactamente la forma de Poland-Scheraga o WSME, es decir, donde $m_i = 1$ implica pertenecer a un bucle (contribución totalmente entrópica, sin interacciones), o a una región çompacta"(contribución totalmente energética), respectivamente. A pesar de estas diferentes interpretaciones del significado de $m_i = 1$, ambos modelos permiten definir una secuencia de monómeros de dos tipos distintos A y B, y trabajar con un número reducido de parámetros, en lugar que los $N(N+1)/2$ correspondientes al conjunto completo de las θ_{ij}.
Cabe destacar también que para el ajuste a los datos experimentales, esta vez abandonamos el Simulated Annealing y utilizamos un algoritmo semi determinista llamado Subplex.

Para el modelo de **Poland-Scheraga** 3, imponiendo que el coste entrópico de cerrar el loop tenga una dependencia con la distancia $f_{ij} = \alpha ln(j-i)$ y que el parámetro a_i tenga solo dos valores dependiendo del tipo de compartimento del monómero i, se llega a una dependencia con 4 parámetros $(\varepsilon_0, \varepsilon_1, q, \alpha)$:

$$\theta_{ij} = \begin{cases} \varepsilon_0\delta_{\sigma_i 0} + \varepsilon_1\delta_{\sigma_i 1} + q + \alpha \cdot ln2 & \text{si } (i=j) \text{ y } 0 \le i \le N-1 \\ \alpha \cdot ln\left(\frac{(j-i+1)^2}{(j-i)(j-i+2)}\right) & \text{si } 0 \le i < j \le N-1 \end{cases} \tag{12}$$

Esta matriz se usa para calcular la matriz μ y con ella la matriz de correlación. A la que comparamos con la matriz de Correlación experimental con el fin de miminizarla y obtener la mejor matriz de la forma de Poland-Scheraga que mejor describa a los datos experimentales, pero la forma de calcular la energía es como la expresión 8.
La optimización produce unos parámetros
$(\varepsilon_0, \varepsilon_1, q, \alpha) = (-0,535404, -0,967698, -3{,}077087, 3{,}052826)$ y una energía de correlación final $E_{corr} = 252,482079$.

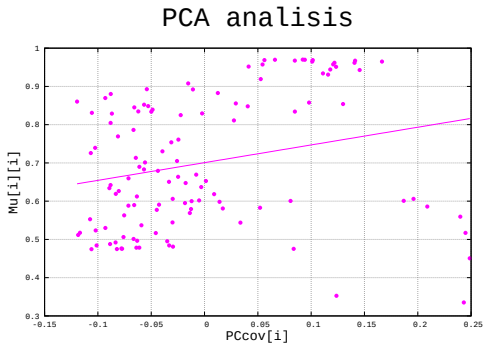
En cambio. si partimos de una matriz θ en el marco del modelo de **WSME**, necesitamos especificar también qué tipo de dependencia Δ_{ij} de la distancia $j-i$ tienen las interacciones entre monómeros: inspirados por la física de polímeros, que producen una probabilidad de contacto que va como una ley de potencia, asumimos $\Delta_{ij} = |j-i|^{-\gamma}$ y que solo existen tres energías de interacción entre dos monómeros dependiendo de los compartimentes a los que pertenecen. Con referencia a la Ec. 4, escribimos la matriz θ en función de cuatro parámetros $(e_{11}, e_{01}, e_{00}, \gamma)$:

$$\theta_{ij} = \begin{cases} e_{11}\delta_{\sigma_i 1} + e_{00}\delta_{\sigma_i 0} & \text{si } (i=j) \text{ y } 0 \le i \le N-1 \\ \left[e_{11}\delta_{\sigma_i 1}\delta_{\sigma_j 1} + e_{01}\left(\delta_{\sigma_i 1}\delta_{\sigma_j 0} + \delta_{\sigma_i 0}\delta_{\sigma_j 1}\right) + e_{00}\delta_{\sigma_i 0}\delta_{\sigma_j 0}\right] \cdot |j-i|^{-\gamma} & \text{si } 0 \le i < j \le N-1 \end{cases} \tag{13}$$

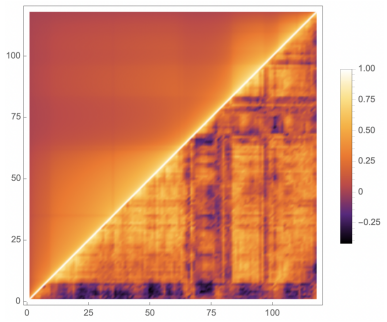
En este caso obtuvimos unos parámetros [15] $(e_{11}, e_{01}, e_{00}, \gamma) = (7,459135E-02, -2,592056E-02, 1,815221E-01, 4,343827E-01)$ y una energía de correlación $E_{corr} = 155,309354$, lo que es una energía menor a la obtenida con el modelo de Poland Scheraga e indicaría que la cromatina se describe mejor con un modelo WSME donde las 'islas' de $m_k = 1$ se asocian a interacciones que con el modelo de Poland-Scheraga, donde las 'islas' de $m_k = 1$ se asocian a la entropía de un lazo.

A continuación realizamos un análisis con la componente Principal de la matriz de covarianza experimental utilizando los μ_{ii} correspondientes al valor optimizado y también representaremos las matrices de correlación correspondientes:

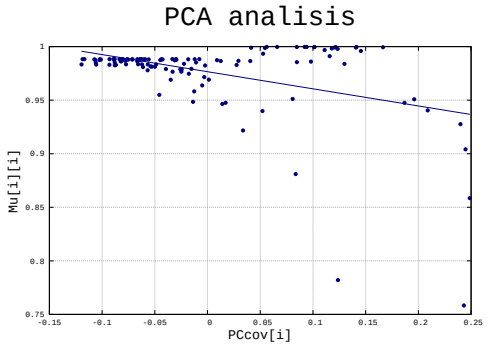
[15] Nótese que el parámetro q de la Ec. 4 se ha incorporado en los valores de θ_{ii}



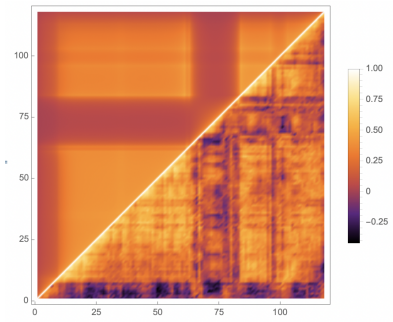
(a) PCA para Poland-Scheraga



(b) Matriz de Correlación para Poland-Scheraga



(c) PCA para WSME



(d) Matriz de Correlación para WSME

Figura 14: a) PCA para Poland-Scheraga. **b)** Matriz de Correlación para Poland-Scheraga. **c)** PCA para WSME. **d)**Matriz de Correlación para WSME.

El resultado del ajuste lineal es $y = 0,4629 \cdot x + 0,7007$ con $\rho = 0,2457$ para Poland-Scheraga.
Y para WSME $y = -0,1598 \cdot x + 0,9766$ con $\rho = -0{,}4205$
En cuanto a estos resultados vemos una dependencia lineal decreciente para WSME, que, sin embargo, no debe preocuparnos: debido a la interpretación de las m_i en los dos modelos, valores de μ_{ii} altos implican prevalencia de zonas de alta entropía y baja interacción en el modelo de Poland-Scheraga, mientras que en el modelo de WSME estas están representadas por $m_i = 0$, es decir por valores bajos de μ_{ii}.

Por otro lado si comparamos las covarianzas de los modelos de Poland-Scheraga y WSME con las experimentales (Figura 14), se ve una mejor acuerdo para el modelo de WSME, ya que es más detallada que para el algoritmo de Poland-Scheraga (pero no lo es tanto como para las pruebas de la sección anterior con todas las θ_{ij} como parámetros libres). Además el coeficiente de correlación Pearson de la prueba de Poland-Scheraga es menor en valor absoluto al de la prueba de WSME.

4. Conclusiones

En este trabajo hemos aplicado una generalización de los modelos de Poland-Scheraga para el ADN y WSME para las proteínas, a la predicción de la estructura a gran escala de la

cromatina, centrándonos en particular en el análisis de los datos de Hi-C relativos al cromosoma 4 de *Drosophila melanogaster*, de longitud $N = 118$ monómeros. Los resultados obtenidos optimizando los $N(N+1)/2$ con el algoritmo de Simulated Annealing son prometedores, y sugieren una interpretación de los valores promedios $\mu_{ii} = \langle m_i \rangle$ como indicadores de los compartimentos de la cromatina (identificados, de manera estandard, basándose en el signos de los elementos del autovector principal de la covarianza experimental), con los cuales presentan unos coeficientes de correlación de Pearson positivos y superiores a $0,5$, llegando como máximo en una de nuestras pruebas hasta $0,7025$.Además se han obtenido unas matrices de covarianza muy similares a la matriz experimental.

La buena correlación entre μ_{ii} y compartimentos es aún más llamativa si nos preguntamos cuantos y cuales monómeros se clasifican correctamente por el modelo. Para este fin hemos implementado una clasificación en dos compartimentos mediante un análisis con la curva ROC, llegando a un umbral óptimo para identificar los compartimentos dado por $\mu \sim 0,7$ para los parámetros μ_{ii} del modelo: con este criterio, la tasa de acierto ha sido del 81 %. También hemos descubierto que los parámetros θ_{ij} obtenido con los parámetros μ del Simulated Annealing corresponden a una pequeña interacción repulsiva si los monómeros pertenecen a compartimentos diferentes, mientras que la interacción puede ser tanto positiva como negativa si pertenecen al mismo compartimento.

Sin embargo, utilizando el Simulated Annealing hemos descubierto que las condiciones de los parámetros del modelo son muy restrictivas y hay que manejarlas con cuidado: en nuestro caso, hemos tenido que descartar muchas trayectorias simuladas con un protocolo de Monte Carlo que finalmente se ha revelado insuficiente para abordar una dinámica con tantos vínculos en las variables.

Finalmente, hemos tratado de simplificar la interpretación de los parámetros ajustando la covarianza experimental directamente con los modelos de Poland-Scheraga y WSME çlásicos", utilizando solo cuatro parámetros de interacción, que tienen una interpretación más fácil. Como era de esperar, el acuerdo con los datos experimentales ha sido notablemente peor que en el caso de utilizar todos los θ_{ij} como parámetros libres. Sin embargo, es interesante notar como los resultados, tanto los relativos a la energía mínima, como la correlación con los compartimentos experimentales, sugieren una interpretación de las variables m_i más acorde al modelo de WSME que al de Poland-Scheraga.

Como conclusión podemos decir que el uso de modelos estadísticos para el modelado de la cromatina ha sido satisfactorio ya que hemos obtenido resultados con una gran correlación teórico experimental, además gracias a los resultados de estos modelos, hemos podido obtener mucha información acerca de como interacciona la cromatina y de como se puede clasificar en dos compartimentos. Cabe destacar que unos de los puntos de fuerza del enfoque utilizado es que no hemos tenido que recurrir a simulaciones detalladas de dinámica molecular de la cromatina, y ha bastado con utilizar unos modelos que tienen solución exacta.

Bibliografía

[1] G. Tiana, L. Giorgetti y H.D. Sherali
Integrating experiment, theory and simulation to determine the structure and dynamics of mammalian chromosomes
Current Opinion in Structural Biology 2018, 49:11–17.

[2] M. Imakaev, G. Fudenberg, L. Mirny

Modeling chromosomes: Beyond pretty pictures
FEBS Lett. 2015 Oct 7; 589(20 0 0): 3031–3036.

[3] C.Ugolini
Study of a simplified model to describe chromosome conformations*Master Thesis*, 2019, Universidad de Milán. Private Communication

[4] E. Lieberman-Aiden, N.L. van Berkum, L. Williams, M. Imakaev, T. Ragoczy , A. Telling, I. Amit, B.R. Lajoie, P.J. Sabo, M.O. Dorschner, R. Sandstrom, B. Bernstein, M.A. Bender, M. Groudine, A. Gnirke, J. Stamatoyannopoulos, L.A. Mirny, E.S. Lander, J. Dekker
Comprehensive mapping of long range interactions reveals folding principles of the human genome
Science. 2009 ; 326(5950): 289–293.

[5] K. Pal, M. Forcato y F. Ferrari
Hi-C analysis: from data generation to integration
Biophysical Reviews (2019) 11:67–78

[6] G. Tiana y L. Giorgetti.
Coarse graining of a giant molecular system: the chromatin fiber.
Methods Mol Biol. 2019; 2022:399-411.

[7] C. Lanctôt, T. Cheutin, M. Cremer, G. Cavalli, T. Cremer.
Dynamic genome architecture in the nuclear space: regulation of gene expression in three dimensions.
Nat. Rev. Genet., 2007, vol. 8, pp 104–115.

[8] S.S.P. Rao, M.H. Huntley, N.C. Durand, E.K. Stamenova, I.D. Bochkov, J.T. Robinson, A.L. Sanborn, I. Machol, A.D. Omer, E.S. Lander et al.
A 3D map of the human genome at kilobase resolution reveals principles of chromatin looping. *Cell, 2014, 159: 1665–1680.*

[9] B.R. Lajoie, J. Dekker, N. Kaplan.
The Hitchhiker's guide to Hi-C analysis: practical guidelines. *Methods, 2015; 72: 65-75.*

[10] J. Nuebler, G. Fudenberg, M. Imakaev, N. Abdennur y L. A. Mirny
Chromatin organization by an interplay of loop extrusion and compartmental segregation. *PNAS, 2018, 115 (29) E6697-E6706.*

[11] M. Imakaev, G. Fudenberg, R. Patton McCord, N. Naumova, A. Goloborodko, B. R. Lajoie, J. Dekker y L. A. Mirny
Iterative correction of Hi-C data reveals hallmarks of chromosome organization
Nature; 2012; VOL. 9 NO.10:999-1006.

[12] M. Zamparo, **A nice exponential family**, *unpublished*

5. Anexos

5.1. Anexo I: Condiciones que debe cumplir cada elemento (i, j) en una matriz WSME

En una matriz Mu de WSME podemos ver los siguientes casos según la posición del elemento μ_{ij} con $1 \leq i \leq j \leq N$:

a) Diagonal Principal ($\mathbf{i = j}$): Aquí distinguimos los siguientes subcasos.
a.1) Elemento $(\mathbf{1, 1})$: Solo cumple dos condiciones
Tipo 3 : $\mu_{12} < \mu_{11}$
Tipo 1 : $0 < \mu_{22} - \mu_{12} < 1 - \mu_{11}$
Por ende $\boxed{\mu_{11} \in (\mu_{12}, 1 - \mu_{22} + \mu_{12})}$

a.2) Elemento $(\mathbf{N, N})$: Solo cumple una condición
Tipo 3 : $0 < \mu_{NN} - \mu_{N-1\ N} < 1 - \mu_{N-1\ N-1}$
Por ende $\boxed{\mu_{NN} \in (\mu_{N-1\ N}, 1 - \mu_{N-1\ N-1} + \mu_{N-1\ N})}$

a.3) Resto de elementos de la Diagonal Principal: Solo cumple tres condiciones
Tipo 1 : $0 < \mu_{jj} - \mu_{j-1\ j} < 1 - \mu_{j-1\ j-1}$
Tipo 1 : $0 < \mu_{j+1\ j+1} - \mu_{j\ j+1} < 1 - \mu_{jj}$
Tipo 2 : $0 < \mu_{j\ j+1} - \mu_{j-1\ j+1} < \mu_{jj} - \mu_{j-1\ j}$
Por ende $\boxed{\mu_{jj} \in (\mu_{j\ j+1} - \mu_{j-1\ j+1} + \mu_{j-1\ j}, 1 - \mu_{j-1\ j-1} + \mu_{j-1\ j})}$ y $\boxed{\mu_{jj} \in (0, 1 + \mu_{j\ j+1} - \mu_{j+1\ j+1})}$

b) Fuera de la Diagonal Principal ($\mathbf{i \neq j}$): Aquí distinguimos los siguientes subcasos.
b.1) Elementos de la Diagonal Secundaria ($\mathbf{j = i + 1}$) :

b.1.1) Elemento $(\mathbf{1, 2})$: Solo cumple tres condiciones:
Tipo 1 : $0 < \mu_{22} - \mu_{12} < 1 - \mu_{11}$
Tipo 2 : $0 < \mu_{23} - \mu_{13} < \mu_{22} - \mu_{12}$
Tipo 3 : $\mu_{13} < \mu_{12} < \mu_{11}$
Por ende $\boxed{\mu_{12} \in (\mu_{11} + \mu_{22} - 1, \mu_{13} + \mu_{22} - \mu_{23})}$ y $\boxed{\mu_{12} \in (\mu_{13}, \mu_{11})}$

b.1.2) Elemento $(\mathbf{N - 1, N})$: Solo cumple dos condiciones:
Tipo 1 : $0 < \mu_{NN} - \mu_{N-1\ N} < 1 - \mu_{N-1\ N-1}$
Tipo 2 : $0 < \mu_{N-1\ N} - \mu_{N-2\ N} < \mu_{N-1\ N-1} - \mu_{N-2\ N-1}$
Por ende $\boxed{\mu_{N-1\ N} \in (\mu_{N-1\ N-1} + \mu_{NN} - 1, \mu_{NN})}$ y $\boxed{\mu_{N-1\ N} \in (\mu_{N-2\ N}, \mu_{N-1\ N-1} - \mu_{N-2\ N-1} + \mu_{N-2\ N})}$

b.1.3) Resto de la Diagonal Secundaria: Solo cumplen cuatro condiciones:
Tipo 1 : $0 < \mu_{i+1\ j} - \mu_{i\ j} < 1 - \mu_{i\ j-1}$
Tipo 2 : $0 < \mu_{i\ j} - \mu_{i-1\ j} < \mu_{i\ j-1} - \mu_{i-1\ j-1}$
Tipo 2 : $0 < \mu_{i\ j+1} - \mu_{i-1\ j+1} < \mu_{i\ j} - \mu_{i-1\ j}$
Tipo 2 : $0 < \mu_{i+1\ j+1} - \mu_{i\ j+1} < \mu_{i+1\ j} - \mu_{i\ j}$
Por ende $\boxed{\mu_{i\ j} \in (\mu_{i\ j+1} - \mu_{i-1\ j+1} + \mu_{i-1\ j}, \mu_{i\ j-1} - \mu_{i-1\ j-1} + \mu_{i-1\ j})}$ y
$\boxed{\mu_{i\ j} \in (\mu_{i\ j-1} + \mu_{i+1\ j} - 1, \mu_{i\ j+1} + \mu_{i+1\ j} - \mu_{i+1\ j+1})}$

b.2) Elementos fuera de la Diagonal Principal y Secundaria ($\mathbf{j > i + 1}$): Tenemos estos subcasos.
b.2.1) Primera Fila$(\mathbf{0, j})$: Dentro de él tenemos dos posibilidades.

b.2.1.1) Elemento $(\mathbf{1, N})$: Solo cumple dos condiciones.
Tipo 3 : $0 < \mu_{1N} < \mu_{1\ N-1}$
Tipo 2 : $0 < \mu_{2\ N} - \mu_{1\ N} < \mu_{2\ N-1} - \mu_{1\ N-1}$
Por ende $\boxed{\mu_{1\ N} \in (\mu_{1\ N-1} + \mu_{2\ N} + \mu_{2\ N-1}, \mu_{2\ N})}$ y $\boxed{\mu_{1\ N} \in (0, \mu_{1\ N-1})}$

b.2.1.2) Resto de la Primera fila $(\mathbf{1, j})$: Solo cumplen cuatro condiciones.
Tipo 3 : $0 < \mu_{1\ j+1} < \mu_{1\ j}$

$Tipo\ 3:\ 0 < \mu_{1\,j} < \mu_{1\,j-1}$
$Tipo\ 2:\ 0 < \mu_{2\,j} - \mu_{1\,j} < \mu_{2\,j-1} - \mu_{1\,j-1}$
$Tipo\ 2:\ 0 < \mu_{2\,j+1} - \mu_{1\,j+1} < \mu_{2\,j} - \mu_{1\,j}$
Por ende $\boxed{\mu_{1\,j} \in (\mu_{1\,j+1}, \mu_{1\,j-1})}$ y $\boxed{\mu_{1\,j} \in (\mu_{1\,j-1} + \mu_{2\,j} - \mu_{2\,j-1}, \mu_{1\,j+1} + \mu_{2\,j} - \mu_{2\,j+1})}$

b.2.2) Elementos restantes: Dentro de él tenemos dos posibilidades:
b.2.2.1) Última Columna (i, N)**:** Cumplen 2 condiciones:
$Tipo\ 2:\ 0 < \mu_{i\,N} - \mu_{i-1\,N} < \mu_{i\,N-1} - \mu_{i-1\,N-1}$
$Tipo\ 2:\ 0 < \mu_{i-1\,N} - \mu_{i-\,N} < \mu_{i+1\,N-1} - \mu_{i\,N-1}$
Por ende $\boxed{\mu_{i\,N} \in (\mu_{i-1\,N}, \mu_{i\,N-1} - \mu_{i-1\,N-1} + \mu_{i-1\,N})}$ y $\boxed{\mu_{i\,N} \in (\mu_{i\,N-1} + \mu_{i+1\,N} - \mu_{i+1\,N-1}, \mu_{i+1\,N})}$

b.2.2.2) Otros: Cumplen 4 condiciones:
$Tipo\ 2:\ 0 < \mu_{ij} - \mu_{i-1\,j} < \mu_{i\,j-1} - \mu_{i-1\,j-1}$
$Tipo\ 2:\ 0 < \mu_{i\,j+1} - \mu_{i-1\,j+1} < \mu_{ij} - \mu_{i-1\,j}$
$Tipo\ 2:\ 0 < \mu_{i+1\,j} - \mu_{ij} < \mu_{i+1\,j-1} - \mu_{i\,j-1}$
$Tipo\ 2:\ 0 < \mu_{i+1\,j+1} - \mu_{i\,j+1} < \mu_{i+1\,j} - \mu_{ij}$
Por ende $\boxed{\mu_{ij} \in (\mu_{i\,j+1} - \mu_{i-1\,j+1} + \mu_{i-1\,j}, \mu_{i\,j-1} - \mu_{i-1\,j-1} + \mu_{i-1\,j})}$ y
$\boxed{\mu_{ij} \in (\mu_{i\,j-1} + \mu_{i+1\,j} - \mu_{i+1\,j-1}, \mu_{i\,j+1} + \mu_{i+1\,j} - \mu_{i+1\,j+1})}$

5.2. Anexo II: Demostración de las condiciones inciales impuestas por las ecuaciones de WSME

Implementamos $\mu_{ij} = \frac{c}{(j-i)^a}$ si $j \neq i$ y que todos los μ_{jj} son de las misma forma.

Condición $Tipo\ 1:\ 0 < \mu_{jj} - \mu_{j-1\,j} < 1 - \mu_{j-1\,j-1} \Leftrightarrow \mu_{j-1\,j} < \mu_{jj} < 1 + \mu_{j-1\,j} - \mu_{j-1\,j-1}$
Imponemos que $\mu_{jj} = \mu_{j-1\,j-1}$, entonces se obtienen dos condiciones:
$\mu_{j-1\,j} < \mu_{jj} \Leftrightarrow c < \mu_{jj}$ y $\mu_{jj} < \frac{1+c}{2}$, puesto que $\mu_{ij} = \frac{c}{(j-i)^a}$ si $j \neq i$.

Condición $Tipo\ 2:\ 0 < \mu_{i+1\,j} - \mu_{ij} < \mu_{i+1\,j-1} - \mu_{ij-1}$
Para el caso $j = i+2$, $0 < \mu_{i+1\,i+2} - \mu_{i\,i+2} < \mu_{i+1\,i+1} - \mu_{i\,i+1} \Leftrightarrow \mu_{i\,i+1} < \mu_{i+1\,i+2} - \mu_{i\,i+2} + \mu_{i\,i+1} < mu_{i+1\,i+1}$.
De la condición $\mu_{i+1\,i+2} - \mu_{i\,i+2} + \mu_{i\,i+1} < mu_{i+1\,i+1}$ deducida de la expresión anterior y con $\mu_{ij} = \frac{c}{(j-i)^a}$ si $j \neq i$ se llega a que:
$2c - \frac{c}{2^a} < \mu_{i+1\,i+1}$, que es una condición más restrictiva que el extremo inferior obtenida por la Condición $Tipo\ 1$ y que podemos usar puesto que los elementos μ_{jj} hemos supuesto que son todos iguales.
Luego $c\left(2 - \frac{1}{2^a}\right) < 2c < \mu_{jj} < \frac{1+c}{2}$ y para ser conservadores:

$\boxed{\mu_{jj} \in \left(2c + \delta, \frac{1+c}{2} - \delta\right)}$ siendo δ un pequeño número que permite crear un intervalo en el que pertenecen los mus diagonales.

Por otro lado se debe cumplir que $2c+\delta < \frac{1+c}{2} - \delta \Leftrightarrow 4c+2\delta < 1+c-2\delta \Leftrightarrow 3c < 1-4\delta \Leftrightarrow c < \frac{1-4\delta}{3}$.

O lo que es lo mismo $\boxed{\text{c} \in (0, \frac{1-4\delta}{3})}$

5.3. Anexo III: Demostración de que $\mu_{i\,i+k} = \mu_{i\,i+k-1} \cdot \mu_{i+k\,i+k}$ cumplen las condiciones (10)

Si los elementos diagonales son generados de la siguiente forma: $\mu_{jj} \in (0,1)$. Además el resto de elementos son generados diagonal a diagonal de esta forma $\mu_{i\,i+k} = \mu_{i\,i+k-1} \cdot \mu_{i+k\,i+k}$ con $k = 1, ..., N-1$, índice que indexa a las diagonales, entonces se cumplen las condiciones 10:

En primer lugar quitaremos en el indice k de tal manera que $i + k \to j$, con el fin de hacer la expresión más general, quedando $\mu_{i\,j} = \mu_{i\,j-1} \cdot \mu_{j\,j}$

Condición *Tipo* 1 : $0 < \mu_{jj} - \mu_{j-1\,j} < 1 - \mu_{j-1\,j-1}$ si $0 < j \leq N-1$.
Aplicando la expresión, $0 < \mu_{jj} - \mu_{j-1\,j-1}\mu_{jj} < 1 - \mu_{j-1\,j-1} \Leftrightarrow 0 < \mu_{jj}(1 - \mu_{j-1\,j-1}) < 1 - \mu_{j-1\,j-1} \Leftrightarrow 0 < \mu_{jj} < 1$, lo que cumple la hipótesis.

Condición *Tipo* 2 : $0 < \mu_{i+1\,j} - \mu_{i\,j} < \mu_{i+1\,j-1} - \mu_{i\,j-1}$ si $0 < i+1 < j \leq N-1$.
Aplicando la expresión, $0 < \mu_{i+1\,j-1}\mu_{jj} - \mu_{i\,j-1}\mu_{jj} < \mu_{i+1\,j-1} - \mu_{i\,j-1} \Leftrightarrow 0 < \mu_{jj}(\mu_{i+1\,j-1} - \mu_{i\,j-1}) < \mu_{i+1\,j-1} - \mu_{i\,j-1} \Leftrightarrow 0 < \mu_{jj} < 1$, lo que está de acuerdo con la hipótesis.

Condición *Tipo* 3 : $0 < \mu_{0\,j} < \mu_{0\,j-1}$ si $0 < j \leq N-1$.
Aplicando de nuevo la expresión, $0 < \mu_{0\,j-1}\mu_{jj} < \mu_{0\,j-1} \Leftrightarrow 0 < \mu_{jj} < 1$.

Por lo tanto imponiendo estas expresiones se cumplen las condiciones de los parámetros μ (10).

5.4. Anexo IV: Deducción de la matriz θ_{ij} de Poland Scheraga

La forma general de la matriz θ_{ij} es la siguiente:

$$\theta_{ij} = \begin{cases} -b_i - Tf_{i-1\,i+1} & \text{si } (i=j) \text{ y } 0 \leq i \leq N-1 \\ T(-f_{ij} + f_{i-1\,j} + f_{i\,j+1} - f_{i-1\,j+1} & \end{cases}$$

Ahora si imponemos que $b_i = -a_i - Tq$ y que $f_{ij} = \alpha ln(j-i)$. Para a_i le imponemos que sea distinto su valor dependiendo del compartimento al que pertenece el monómero, de tal manera que $a_i = e_0\delta_{\sigma_i\,0} + e_1\delta_{\sigma_i\,1}$, siendo σ_i una variable binaria que dependiendo su valor sabemos si el monómero está en un compartimento u otro.
Realizando operaciones con los logaritmos y expresando el resultado en unidades de Temperatura, llegamos a que:

$$\boxed{\theta_{ij} = \begin{cases} \varepsilon_0\delta_{\sigma_i 0} + \varepsilon_1\delta_{\sigma_i 1} + q + \alpha \cdot ln2 & \text{si } (i=j) \text{ y } 0 \leq i \leq N-1 \\ \alpha \cdot ln\left(\frac{(j-i+1)^2}{(j-i)(j-i+2)}\right) & \text{si } 0 \leq i < j \leq N-1 \end{cases}}$$

*Anexo V: Deducción de la matriz θ_{ij} de WSME La matriz θ_{ij} de la forma de WSME es de la forma $\theta_{ij} = \varepsilon_{ij}\Delta_{ij}$ siendo Δ_{ij} la matriz de contactos del polímero.
Considerando que $\Delta_{ij} = |j-i|^{-\gamma}$ si $i \neq j$, entonces llegamos a que :

$$\boxed{\theta_{ij} = \begin{cases} e_{11}\delta_{\sigma_i 1} + e_{00}\delta_{\sigma_i 0} & \text{si } (i=j) \text{ y } 0 \leq i \leq N-1 \\ \left[e_{11}\delta_{\sigma_i 1}\delta_{\sigma_j 1} + e_{01}\left(\delta_{\sigma_i 1}\delta_{\sigma_j 0} + \delta_{\sigma_i 0}\delta_{\sigma_j 1}\right) + e_{00}\delta_{\sigma_i 0}\delta_{\sigma_j 0}\right] \cdot |j-i|^{-\gamma} & \text{si } 0 \leq i < j \leq N-1 \end{cases}}$$

5.5. Anexo V: Código de Simmulated Annealing

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

```
#include <math.h>

#define NormRANu (2.3283063671E-10F)

//Parisi Rapuano, para los aleatorios planos
extern float Random (void);
extern void ini_ran (int SEMILLA);

unsigned int irr[256];
unsigned int ir1;
unsigned char ind_ran,ig1,ig2,ig3;

//Funcion de lectura de la matriz de covarianza experimental
void lectura (double **A,int *N);
void maximo (double **C, double *maxi,int *N);
//Funcion que crea una matriz WSME de cero
void calculamatrizmu (double **Mu,int *I,int *J, int *D);
//NUEVA funcion para generar una matriz WSME con el algoritmo
void matriz_inicial_mu (double **Mu,int *N);
//Calcula el numero de condiciones que no se cumplen y lo devuelve normalizados
void comprueba (double **Mu,int *fail,int *N);
//Funcion para actualizar la cesta de valores que podemos cambiar, ya que
void actualizalacesta (double **Mu,double **Muup, double **Mulow,int *Modificable, int *I, i
//Funcion que devuelve los extremos de generación del elemento (i,j) input
void buscalosextremos (double **Mu, double *a, double *b,int *is,int *js,int *N);
//Funcion que creaun Muij a partir valido con el resto
void switchcreaunmu (double **Mu, int *is,int *js,int *N);
//Iguala la Matriz Mu como los valores de Mu2
void iguala (double **Mu,double **Mu2,int *N);
void igualacov (double **C,double **C2,double **Cov,double **Cov2,int *is, int *js,int *N);
//Funcion que calcule la matriz de covarianzas completa
void matrizC (double **C,double **Cov, double **Mu, int *N);
//Funcion que calcula las covarianzas que cambian al realizar un cambio en la matriz mu
void pasaacovsred (double **Cold, double **Cnew, double **Cov, double **Mu,double **Munew, i
//Funcion que calcula la energia asociada a lamatriz de correlación
void energia (double **Cexp,double **Cold,double *E,int *N);
//Para condicion inicial
void condicioninicial (double **Mu, double **Munew, double **Cold,double **Covold,double **C
void matrizporparametros (double **Mu, double *cs, double *as,double *epsa,double *ws,int *N
//Funcion que calcule los thetas a partir de los mus calculados
void calculatheta (double **Mu,double **Thetas, int *N);
//Funciones que guarde los mus finales y thetas en un fichero
void guardamu (double **Mu, int *N);
void guardamuinicial (double **Mu, int *N);
void guardamuminima (double **Mu,double *E,double *T,int *N);
```

```
void guardaCovarianzaminima (double **Cov,double *E,double *T, int *N);
void guardatheta (double **Theta, int *N);

extern void  ini_ran(int SEMILLA)
{
    int INI,FACTOR,SUM,i;
    srand(SEMILLA);
    INI=SEMILLA;
    FACTOR=67397;
    SUM=7364893;
    for(i=0;i<256;i++)
    {
        INI=(INI*FACTOR+SUM);
        irr[i]=INI;
    }
    ind_ran=ig1=ig2=ig3=0;
}

float Random(void)
{
    float  r;
    ig1=ind_ran-24;
    ig2=ind_ran-55;
    ig3=ind_ran-61;
    irr[ind_ran]=irr[ig1]+irr[ig2];
    ir1=(irr[ind_ran]^irr[ig3]);
    ind_ran++;
    r=ir1*NormRANu;

    return r;
}

int main() {
          FILE *f;
          FILE *f2;
          f=fopen("./Mu_E=0.780198.txt","r");
          f2=fopen("./v3_evolucion_condE=0.780198.txt","w");
        ini_ran(123456789);
    //Variables enteras (int)
        int i,j;//Para usar como indice de fila, columna en los bucles for
        int N; //Para guardar la dimensión de las matrices, marcado por la dimensión de la m
        int cont,k,iter,itermax;//Variable para contar los cambios en un bucle de termalizad
        int is,js; //Variables enteras para mostrar elemento (is,js) que se pasara en un pas
        int nf;//Para guardar los fallos de las condiciones WSME, usando comprueba
        int alek,M; //Sera el numero aleatorio principal para generar los indices
```

```
    int contador;
    int contk,kmax;
    int contintraacept; //Para guardar los cambios realizados cada un numero fijo de swe
//Variables decimales (double)
    double w;// Guardara un aleatorio plano para comparar con expnew en un paso de terma
    double beta,acept,intraacept;//Variable para beta y para guardar la aceptancia
    double T; //Variable para tener la temperatura y la temperatura de referencia
    double expnew,Enew,Eold,Emin,Delta; // Variable para guardar la exponencial que se c
    double aux;//para releer una matriz de un fichero
    double alpha;
    double max;
    double maxalek;
    double epsilon;
//Variables Matriz (double)
    double **Cexp,**Cold,**Covold,**Cnew,**Covnew,**Mu,**Munew,**Muup,**Mulow,**Theta;//
//Vectores enteros
    int *I,*J,*Modificable;
//Inicializacion de las variables
    i=j=N=0;cont=i=j=k=iter=itermax=0; is=js=0; nf=0; alek=M=0;contador=0; contk=kmax=0;
    w=0.0;beta=acept=intraacept=0.0; T=0.0;  expnew=Enew=Eold=Emin=Delta=0.0;alpha=0.0;
    aux=0.0;max=0.0; maxalek=0.0; epsilon=0.0;
//Dar valor a las variables importantes
    N=118;
    M=(N*(N+1))/2;
    alpha=0.75;
    itermax=82;//lo marca alpha
    kmax=3.0*M*pow(10,3);
    Emin=10.0;
    epsilon=pow(10,-6);
//Definir las matrices por memoria dinamica
    Cexp=malloc(N*sizeof(double*)); for(i=0;i<N;i++) Cexp[i]=malloc(N*sizeof(double));
    Cold=malloc(N*sizeof(double*)); for(i=0;i<N;i++) Cold[i]=malloc(N*sizeof(double));
    Covold=malloc(N*sizeof(double*)); for(i=0;i<N;i++) Covold[i]=malloc(N*sizeof(double)
    Cnew=malloc(N*sizeof(double*)); for(i=0;i<N;i++) Cnew[i]=malloc(N*sizeof(double));
    Covnew=malloc(N*sizeof(double*)); for(i=0;i<N;i++) Covnew[i]=malloc(N*sizeof(double)
    Mu=malloc(N*sizeof(double*)); for(i=0;i<N;i++) Mu[i]=malloc(N*sizeof(double));
    Munew=malloc(N*sizeof(double*)); for(i=0;i<N;i++) Munew[i]=malloc(N*sizeof(double));
    Muup=malloc(N*sizeof(double*)); for(i=0;i<N;i++) Muup[i]=malloc(N*sizeof(double));
    Mulow=malloc(N*sizeof(double*)); for(i=0;i<N;i++) Mulow[i]=malloc(N*sizeof(double));
    Theta=malloc(N*sizeof(double*)); for(i=0;i<N;i++) Theta[i]=malloc(N*sizeof(double));
//Definir los vectores int
    I=malloc(M*sizeof(int));
    J=malloc(M*sizeof(int));
    Modificable=malloc(M*sizeof(int));
//Creacion de la lookuptable completa
```

```
        alek=0;
        for(i=0;i<N;i++)
        { for(j=i;j<N;j++)
            {
                I[alek]=i;
                J[alek]=j;
                alek++;
            }
        }
    //Lectura de la matriz de covarianzas experimental, busqueda de la
        lectura(Cexp,&N);
        //Secuencia de control de la lectura
        printf("¿Ha leido bien?\n");
        printf("\tCexp[0][0]=%lf y Cexp[117][117]=%lf\n",Cexp[0][0],Cexp[117][117]);
    //Creacion de la condicion inicial o lectura de un fichero
        do{
            fscanf(f,"%d %d %lf\n",&i,&j,&aux);
            Mu[i][j]=aux;
            fflush(f);
        }while(feof(f)==0);
        i=j=0;
        fclose(f);
//        condicioninicial(Mu, Munew, Cold, Covold, Cnew,Covnew,Cexp, &N);
//        guardamuinicial(Mu,&N);
        comprueba(Mu, &nf, &N);
        printf("¿Ha calculado bien Calculamatrizmu?\n");
        printf("\tFallos=%d\n",nf);
    //Temperatura inicial
        maximo(Cexp,&max,&N);
        T=max;
        beta=1.0/T;
        printf("betaini=%lf T_0=%E\n",beta,T); //Mostrar por pantalla ese valor
    //Convertir la matriz Mu inicial a covarianzas (Cold)
        matrizC(Cold,Covold,Mu,&N);
    //Calcular la energia asociada (Eold) a esa matriz Cold
        energia(Cexp,Covold,&Eold, &N);
        printf("\tEold(inicial)=%lf\n",Eold); //Mostrar por pantalla esa energia
    //Secuencias de guardado
        fprintf(f2,"#Enegia (inicial)=%lf\n",Eold);
        printf("iter    Sweep   Energia   Aceptancia\n");
        printf("-----------------------------------\n");
    //Igualamos las matrices C y las matrices de covarianza y Munew, para realizar el cambio
        iguala(Cnew, Cold, &N);
        iguala(Covnew, Covold, &N);
        iguala(Munew,Mu,&N);
```

```
// Comienzo del SIMULATED ANNEALING
    for(iter=0;iter<itermax+1;iter++)
    {
        /*Algoritmo de Metropolis para Montecarlo*/
        for(k=0;k<kmax;k++)//PONER KMAX+1 CUANDO SE ACABE ESTÁ PRUEBA
        {
            actualizalacesta(Mu,Muup,Mulow,Modificable,I,J,&maxalek,&epsilon,&N,&M);
            //Generacion del proximo elemento (is,js) a cambiar
            do{
                alek=floor(Random()*maxalek);
            }while(alek==maxalek);
            contk=Modificable[alek];
            is=I[contk];
            js=J[contk];
            //Cambiar el termino (is,js) generandolo al azar entre sus extremos de gener
            Munew[is][js]=(Muup[is][js]-Mulow[is][js])*Random() + Mulow[is][js];
            //Calculo de la nueva matriz C y de Covarianza
            pasaacovsred(Cold, Cnew,Covnew, Mu, Munew, &is, &js, &N);
            //Calculo de la nueva energia
            energia(Cexp, Covnew, &Enew, &N);
            //Comprobacion de esta matriz (opcional)
                //comprueba(Munew, &nf, &N);
            //Si aceptamos el cambio debemos guardar en Cold y en Covold las matrices bu
            //cuando el cambio no se acepte, donde la matriz Cnew y Covnew debe ser la m
            if (Enew<Eold)
            {
                //Guardamos el cambio
                  Mu[is][js]=Munew[is][js];
                  igualacov(Cold, Cnew, Covold, Covnew, &is, &js, &N);
                  Eold=Enew;
                  cont++;
                  contintraacept++;
                  if(Enew<Emin)
                  {
                      if (iter > 0)
                      {
                          Emin=Enew;
                          guardamuminima(Mu,&Emin,&T,&N);
                          guardaCovarianzaminima(Covnew,&Emin,&T,&N);
                      }

                  }
            }
            else
            {
```

```
                w=Random();
                expnew=exp(-beta*(Enew-Eold));
                if (expnew >= w)
                {
                    Mu[is][js]=Munew[is][js];
                    igualacov(Cold, Cnew, Covold, Covnew, &is, &js, &N);
                    cont++;
                    contintraacept++;
                    Eold=Enew;
                }
                else
                {
                    //Reseteeamos las matriz Munew para partir con la matriz Mu
                    Munew[is][js]=Mu[is][js];
                    igualacov(Cnew, Cold, Covnew, Covold, &is, &js, &N);
                }
            }
            if(contador==M-1)
            {
                intraacept=(double)contintraacept/M;
                fprintf(f2,"%d %d %lf %lf %E\n",iter,(k+iter*kmax)/M,Eold,intraacept,T);
                printf("%d %d %lf %lf %E\n",iter,k/M,Eold,intraacept,T);
                contador=0;
                contintraacept=0;
                intraacept=0.0;
            }
            else
            {
                contador++;
            }

        }
        acept=(double)cont/kmax; //Calculo de la aceptancia
        cont=0; //Reiniciar el contador de la aceptancia
        printf("%d %lf %lf %lf\n",iter,Eold,acept,T); //Resultado del termalizado
        T=alpha*T; //Enfriamiento Geometrico
        beta=1.0/T;
    }
        fclose(f2);
        printf("Ha salido del bucle principal\n");
    //Liberar memoria de las matrices que ya no ser usarán
        //Vectores
        free(I);
        free(J);
        free(Modificable);
```

```
            //Matrices
            for(i=0;i<N;i++)
            {
                free(Cexp[i]);
                free(Cnew[i]);
                free(Munew[i]);
                free(Covnew[i]);
                free(Muup[i]);
                free(Mulow[i]);
            }
            free(Cexp);
            free(Cnew);
            free(Munew);
            free(Covnew);
            free(Muup);
            free(Mulow);
        //Guardar la Matriz Mu final
//            guardamu(Mu,&N);
        //Generacion de los parametros Theta a partir de los Mu
            calculatheta(Mu,Theta,&N);
        //Guardar la Matriz Theta en un fichero
            guardatheta(Theta, &N);
    //Liberar memoria del resto de matrices que faltaban
        i=0;
        for(i=0;i<N;i++)
        {
            free(Cold[i]);
            free(Covold[i]);
            free(Mu[i]);
            free(Theta[i]);
        }
        free(Cold);
        free(Covold);
        free(Mu);
        free(Theta);
        return 0; //FIN
}

void lectura (double **Old,int *N)
{
   FILE *f2;
   f2=fopen("./covmatrix.txt","r");
   int i,j,Nmax;
   double aux;
    i=j=Nmax=0;aux=0.0; //Inicializaciones
```

```
    Nmax=*N;
    for(i=0;i<Nmax;i++)
    {for(j=0;j<Nmax;j++)
        {
            fscanf(f2,"%lf\t",&aux);
            Old[i][j]=aux;
        }
        fscanf(f2,"\n");
    }
    fclose(f2);
}
void maximo (double **C, double *maxi,int *N)
{
    int i,j,Nmax;
    double max;
    i=j=Nmax=0;
    max=0.0;
    Nmax=*N;
    max=C[0][0];
    for(i=0;i<Nmax;i++)
    {
        for(j=i;j<Nmax;j++)
        {
            if(C[i][j]>max)
            {
                max=C[i][j];
            }
        }
    }
    *maxi=max;
}

//Funcion calculamatrizmu con una lookuptable
void calculamatrizmu (double **Mu,int *I,int *J, int *D)
{
    //Variables enteras
       int N; //Tamaño de la matriz
       int is,js;//Variables para marcar la posicion de un cambio
       int k,kmax; //Variables del bucle de cambios y numero maximo de cambios
       int M,alek;
   //Inicializacion
       N=0;is=js=0;k=kmax=0;;M=alek=0;
   //Dar valor a las variables importantes
       N=*D; //Dimension de la matriz
       kmax=100;//Habra qu esubir los cambios
```

```
       M=(N*(N+1))/2;
       kmax=kmax*M;//Numero de pasos reales que se deben realizar
       alek=0;
   //Creo condicion inicial con newcalculamu
       matriz_inicial_mu(Mu,&N);//Una vez hecho este paso tenemos una matriz WSME, pero sus
//REALIZAMOS KMAX CAMBIOS A LA MATRIZ MU Y ASEGURARNOS QUE YA NO TIENEN RELACION (el algorit
    for(k=0;k<kmax;k++)//k: variable contadora de sweeps de MC
    {
       do{
           alek=floor(Random()*M);
       }while(alek==M);
       is=I[alek];
       js=J[alek];
       //Creo un cambio en (is,js) y los guardo en Munew
//       creaunmu(Mu, &is, &js, &N);
        switchcreaunmu(Mu, &is, &js, &N);
    }
}


void matriz_inicial_mu (double **Mu,int *N)
{
    int i,k,Nmax;//Indice para bucles, para bucles por diagonales y para la dimension de la
    i=k=Nmax=0;
    Nmax=*N;
    for(i=0;i<Nmax;i++)//Generamos los elementos de la diagonal aleatoriamente
    {
        Mu[i][i]=Random();
    }
    //Aplico el Algoritmo Mu[i][j]=Mu[i][j-1]*Mu[j][j]
    for(k=1;k<Nmax;k++)//Bucle diagonal
    {for(i=0;i<(Nmax-k);i++)//Para moverme por la diagonal
        {
            Mu[i][i+k]=Mu[i][i+k-1]*Mu[i+k][i+k];
            Mu[i+k][i]=Mu[i][i+k];
        }
    }
}

void comprueba (double **Mu,int *fail,int *N)
{
    int i,j,Nmax; //indice para los bucles de fila , de columnas y para guardar el tamaño de
    int nfail; //varibles para guardar los fallos
    double inter,sup; //Variables que representan la diferencia intermedia y la diferencia 1
    double inter2,sup2;//Variables auxiliares para comprobar las condiciones Tipo2, analogas
```

```
    i=j=Nmax=0; nfail=0; inter=sup=0.0; inter2=sup2=0.0;
    Nmax=*N; //Asignar el valor de N


    //Comprobacion de las condiciones Tipo 3
        for(j=0;j<(Nmax-1);j++)
        {
            if((Mu[0][j]<=Mu[0][j+1])||(Mu[0][j+1]<=0.00))
            {
                nfail++;
            }
        }
    //Comprobacion de las condiciones Tipo1
        inter=sup=100.0;//Le ponemos este valor paraevitar falsos pasos a las condiciones si
        for(j=1;j<Nmax;j++)
        {
            inter=Mu[j][j]-Mu[j-1][j];
            sup=1.0-Mu[j-1][j-1];
            if((inter<=0.00)||(sup<=inter))
            {
                nfail++;
                inter=sup=100.0;
            }
        }


    //Comprobacion de las Tipo 2
    inter2=sup2=100.0; //Ponemos esta condicion para evitar falsos positivos
    for(i=1;i<(Nmax-1);i++)
    { for(j=(i+1);j<Nmax;j++)
        {
            inter2=Mu[i][j]-Mu[i-1][j];
            sup2=Mu[i][j-1]-Mu[i-1][j-1];
            if((inter2 <= 0.00)||(sup2 <= inter2))
            {
                nfail++;
                inter2=sup2=100.0;
            }
        }
    }
    *fail=nfail;
}
void actualizalacesta (double **Mu,double **Muup, double **Mulow,int *Modificable, int *I, i
{
    int i,j,k,Nmax,Mmax;
```

```
    int contador;
    i=j=k=Nmax=Mmax=0;
    contador=0;
    Nmax=*N;
    Mmax=*M;
    double a,b,eps,rho;
    a=b=eps=rho=0.0;
    eps=*epsilon;
    for(k=0;k<Mmax;k++)//Recorremos todos los elementos de la manera vectorial, con los M el
    {
        i=I[k];
        j=J[k];
        buscalosextremos(Mu,&a,&b,&i,&j,&Nmax);
        Mulow[i][j]=a;
        Muup[i][j]=b;
        rho=2.0*((b-a)/(b+a));
        if(rho >= eps)//Estos terminos los podemos cambiar
        {
            Modificable[contador]=k;
            contador++;
        }
    }
    *maxalek=contador+1;
}

void buscalosextremos (double **Mu, double *a, double *b,int *is,int *js,int *N)
{
    int i,j,Nmax;
    int deltaindices;
    double a1,b1,a2,b2;
    i=j=Nmax=0;deltaindices=0;
    a1=b1=a2=b2=0.0;
    Nmax=*N;
    i=*is;
    j=*js;
    deltaindices=j-i;

    switch (deltaindices)
    {
        case 0://Diagonal Principal
            switch (i) {
                case 0: //Subcaso Mu[0][0]
                    b1=1.0-Mu[1][1]+Mu[0][1];
                    a1=Mu[0][1];
                    *a=a1;
```

```
            *b=b1;
            break;
        case 117: //Subcaso Mu[Nmax-1][Nmax-1]
            a1=Mu[Nmax-2][Nmax-1];
            b1=1.0-Mu[Nmax-2][Nmax-2]+Mu[Nmax-2][Nmax-1];
            *a=a1;
            *b=b1;
            break;
        default: //Subcaso resto de la Diagonal Principal
            a1=Mu[i][i+1]-Mu[i-1][i+1]+Mu[i-1][i];
            b1=1.0-Mu[i-1][i-1]+Mu[i-1][i];
            a2=0.0;
            b2=1.0+Mu[i][i+1]-Mu[i+1][i+1];
            if(a1<a2) //Genero desde a2
            {
                if(b2<b1)//Genero hasta b2
                {
                    *a=a2;
                    *b=b2;
                }
                else //Genero hasta b1
                {
                    *a=a2;
                    *b=b1;
                }
            }
            else //Genero desde a1
            {
                if(b2<b1)//Genero hasta b2
                {
                    *a=a1;
                    *b=b2;
                }
                else //Genero hasta b1
                {
                    *a=a1;
                    *b=b1;
                }
            }
    }
    break;
case 1://Diagonal Secundaria
    switch (j) {
        case 1: //Subcaso Mu[0][1]
            a1=Mu[0][0]+Mu[1][1]-1.0;
```

```
    b1=Mu[1][1]+Mu[0][2]-Mu[1][2];
    a2=Mu[0][2];
    b2=Mu[0][0];
    if(a1<a2) //Genero desde a2
    {
        if(b2<b1)//Genero hasta b2
        {
            *a=a2;
            *b=b2;
        }
        else //Genero hasta b1
        {
            *a=a2;
            *b=b1;
        }
    }
    else //Genero desde a1
    {
        if(b2<b1)//Genero hasta b2
        {
            *a=a1;
            *b=b2;
        }
        else //Genero hasta b1
        {
            *a=a1;
            *b=b1;
        }
    }
    break;
case 117: //Subcaso Mu[Nmax-2][Nmax-1]
    a1=Mu[Nmax-2][Nmax-2]+Mu[Nmax-1][Nmax-1]-1.0;
    b1=Mu[Nmax-1][Nmax-1];
    a2=Mu[Nmax-3][Nmax-1];
    b2=Mu[Nmax-2][Nmax-2]-Mu[Nmax-3][Nmax-2]+Mu[Nmax-3][Nmax-1];
    if(a1<a2) //Genero desde a2
    {
        if(b2<b1)//Genero hasta b2
        {
            *a=a2;
            *b=b2;
        }
        else //Genero hasta b1
        {
            *a=a2;
```

```
                *b=b1;
            }
        }
        else //Genero desde a1
        {
            if(b2<b1)//Genero hasta b2
            {
                *a=a1;
                *b=b2;
            }
            else //Genero hasta b1
            {
                *a=a1;
                *b=b1;
            }
        }
        break;
    default://Subcaso resto de elementos de la Diag. Secundaria
        a1=Mu[i][j+1]-Mu[i-1][j+1]+Mu[i-1][j];
        b1=Mu[i][j-1]-Mu[i-1][j-1]+Mu[i-1][j];
        a2=Mu[i][j-1]+Mu[i+1][j]-1.0;
        b2=Mu[i][j+1]+Mu[i+1][j]-Mu[i+1][j+1];
        if(a1<a2) //Genero desde a2
        {
            if(b2<b1)//Genero hasta b2
            {
                *a=a2;
                *b=b2;
            }
            else //Genero hasta b1
            {
                *a=a2;
                *b=b1;
            }
        }
        else //Genero desde a1
        {
            if(b2<b1)//Genero hasta b2
            {
                *a=a1;
                *b=b2;
            }
            else //Genero hasta b1
            {
                *a=a1;
```

```
                    *b=b1;
                }
            }
    }
    break;
default://Resto de elementos
    switch (i) {
        case 0:
            switch (j) {
                case 117: //Subcaso Mu[0][N-1]
                    a1=Mu[0][Nmax-2]+Mu[1][Nmax-1]-Mu[1][Nmax-2];
                    b1=Mu[1][Nmax-1];
                    a2=0.0;
                    b2=Mu[0][Nmax-2];
                    if(a1<a2) //Genero desde a2
                    {
                        if(b2<b1)//Genero hasta b2
                        {
                            *a=a2;
                            *b=b2;
                        }
                        else //Genero hasta b1
                        {
                            *a=a2;
                            *b=b1;
                        }
                    }
                    else //Genero desde a1
                    {
                        if(b2<b1)//Genero hasta b2
                        {
                            *a=a1;
                            *b=b2;
                        }
                        else //Genero hasta b1
                        {
                            *a=a1;
                            *b=b1;
                        }
                    }
                    break;
                default://Resto en la fila 0
                    a1=Mu[0][j-1]+Mu[1][j]-Mu[1][j-1];
                    b1=Mu[0][j+1]+Mu[1][j]-Mu[1][j+1];
                    a2=Mu[0][j+1];
```

```
                b2=Mu[0][j-1];
                if(a1<a2) //Genero desde a2
                {
                  if(b2<b1)//Genero hasta b2
                  {
                      *a=a2;
                      *b=b2;
                  }
                  else //Genero hasta b1
                  {
                      *a=a2;
                      *b=b1;
                  }
                }
                else //Genero desde a1
                {
                  if(b2<b1)//Genero hasta b2
                  {
                      *a=a1;
                      *b=b2;
                  }
                  else //Genero hasta b1
                  {
                      *a=a1;
                      *b=b1;
                  }
                }
    }
    break;
default:
    switch (j) {
        case 117: //Subcaso Ultima columna
            a1=Mu[i-1][Nmax-1];
            b1=Mu[i][Nmax-2]+Mu[i-1][Nmax-1]-Mu[i-1][Nmax-2];
            a2=Mu[i][Nmax-2]+Mu[i+1][Nmax-1]-Mu[i+1][Nmax-2];
            b2=Mu[i+1][Nmax-1];
            if(a1<a2) //Genero desde a2
            {
                if(b2<b1)//Genero hasta b2
                {
                    *a=a2;
                    *b=b2;
                }
                else //Genero hasta b1
                {
```

```
                *a=a2;
                *b=b1;
            }
        }
        else //Genero desde a1
        {
            if(b2<b1)//Genero hasta b2
            {
                *a=a1;
                *b=b2;
            }
            else //Genero hasta b1
            {
                *a=a1;
                *b=b1;
            }
        }
        break;
    default://Elementos restantes
        a1=Mu[i][j+1]-Mu[i-1][j+1]+Mu[i-1][j];
        b1=Mu[i][j-1]-Mu[i-1][j-1]+Mu[i-1][j];
        a2=Mu[i][j-1]+Mu[i+1][j]-Mu[i+1][j-1];
        b2=Mu[i][j+1]+Mu[i+1][j]-Mu[i+1][j+1];
        if(a1<a2) //Genero desde a2
        {
            if(b2<b1)//Genero hasta b2
            {
                *a=a2;
                *b=b2;
            }
            else //Genero hasta b1
            {
                *a=a2;
                *b=b1;
            }
        }
        else //Genero desde a1
        {
            if(b2<b1)//Genero hasta b2
            {
                *a=a1;
                *b=b2;
            }
            else //Genero hasta b1
            {
```

```
                                    *a=a1;
                                    *b=b1;
                                }
                            }
                        }
                }
    }
}

void switchcreaunmu (double **Mu, int *is,int *js,int *N)
{
    int i,j,Nmax;
    int deltaindices;
    double a1,b1,a2,b2;
    i=j=Nmax=0;deltaindices=0;
    a1=b1=a2=b2=0.0;
    Nmax=*N;
    i=*is;
    j=*js;
    deltaindices=j-i;

    switch (deltaindices)
    {
        case 0://Diagonal Principal
            switch (i) {
                case 0: //Subcaso Mu[0][0]
//                    printf("\tDiagonal Principal, i=0\n");
                    b1=1.0-Mu[1][1]+Mu[0][1];
                    a1=Mu[0][1];
                    Mu[0][0]=(b1-a1)*Random()+a1;
                    break;
                case 117: //Subcaso Mu[Nmax-1][Nmax-1]
//                    printf("\tDiagonal Principal, i=N-1\n");
                    a1=Mu[Nmax-2][Nmax-1];
                    b1=1.0-Mu[Nmax-2][Nmax-2]+Mu[Nmax-2][Nmax-1];
                    Mu[Nmax-1][Nmax-1]=(b1-a1)*Random()+a1;
                    break;
                default: //Subcaso resto de la Diagonal Principal
//                    printf("\tDiagonal Principal, Resto\n");
                    a1=Mu[i][i+1]-Mu[i-1][i+1]+Mu[i-1][i];
                    b1=1.0-Mu[i-1][i-1]+Mu[i-1][i];

                    a2=0.0;
                    b2=1.0+Mu[i][i+1]-Mu[i+1][i+1];
                    if(a1<a2) //Genero desde a2
```

```
                    {
                        if(b2<b1)//Genero hasta b2
                        {
                            Mu[i][i]=(b2-a2)*Random()+a2;
                        }
                        else //Genero hasta b1
                        {
                            Mu[i][i]=(b1-a2)*Random()+a2;
                        }
                    }
                    else //Genero desde a1
                    {
                        if(b2<b1)//Genero hasta b2
                        {
                            Mu[i][i]=(b2-a1)*Random()+a1;
                        }
                        else //Genero hasta b1
                        {
                            Mu[i][i]=(b1-a1)*Random()+a1;
                        }
                    }
                }
                break;
            case 1://Diagonal Secundaria
                switch (j) {
                    case 1: //Subcaso Mu[0][1]
//                      printf("\tDiagonal Secundaria, Mu[0][1]\n");
                    a1=Mu[0][0]+Mu[1][1]-1.0;
                    b1=Mu[1][1]+Mu[0][2]-Mu[1][2];
                    a2=Mu[0][2];
                    b2=Mu[0][0];
                    if(a1<a2) //Genero desde a2
                    {
                        if(b2<b1)//Genero hasta b2
                        {
                            Mu[0][1]=(b2-a2)*Random()+a2;
                            Mu[1][0]= Mu[0][1];
                        }
                        else //Genero hasta b1
                        {
                            Mu[0][1]=(b1-a2)*Random()+a2;
                            Mu[1][0]= Mu[0][1];
                        }
                    }
                    else //Genero desde a1
```

```
                    {
                        if(b2<b1)//Genero hasta b2
                        {
                            Mu[0][1]=(b2-a1)*Random()+a1;
                            Mu[1][0]= Mu[0][1];
                        }
                        else //Genero hasta b1
                        {
                            Mu[0][1]=(b1-a1)*Random()+a1;
                            Mu[1][0]= Mu[0][1];
                        }
                    }
                    break;
                case 117: //Subcaso Mu[Nmax-2][Nmax-1]
//                    printf("\tDiagonal Secundaria, Mu[N-2][N-1]\n");
                    a1=Mu[Nmax-2][Nmax-2]+Mu[Nmax-1][Nmax-1]-1.0;
                    b1=Mu[Nmax-1][Nmax-1];
                    a2=Mu[Nmax-3][Nmax-1];
                    b2=Mu[Nmax-2][Nmax-2]-Mu[Nmax-3][Nmax-2]+Mu[Nmax-3][Nmax-1];

                    if(a1<a2) //Genero desde a2
                    {
                        if(b2<b1)//Genero hasta b2
                        {
                            Mu[Nmax-2][Nmax-1]=(b2-a2)*Random()+a2;
                            Mu[Nmax-1][Nmax-2]= Mu[Nmax-2][Nmax-1];
                        }
                        else //Genero hasta b1
                        {
                            Mu[Nmax-2][Nmax-1]=(b1-a2)*Random()+a2;
                            Mu[Nmax-1][Nmax-2]= Mu[Nmax-2][Nmax-1];
                        }
                    }
                    else //Genero desde a1
                    {
                        if(b2<b1)//Genero hasta b2
                        {
                            Mu[Nmax-2][Nmax-1]=(b2-a1)*Random()+a1;
                            Mu[Nmax-1][Nmax-2]= Mu[Nmax-2][Nmax-1];
                        }
                        else //Genero hasta b1
                        {
                            Mu[Nmax-2][Nmax-1]=(b1-a1)*Random()+a1;
                            Mu[Nmax-1][Nmax-2]= Mu[Nmax-2][Nmax-1];
                        }
```

```
                    }
                    break;
                default://Subcaso resto de elementos de la Diag. Secundaria
//                    printf("\tDiagonal Secundaria, Resto\n");
                    a1=Mu[i][j+1]-Mu[i-1][j+1]+Mu[i-1][j];
                    b1=Mu[i][j-1]-Mu[i-1][j-1]+Mu[i-1][j];

                    a2=Mu[i][j-1]+Mu[i+1][j]-1.0;
                    b2=Mu[i][j+1]+Mu[i+1][j]-Mu[i+1][j+1];

                    if(a1<a2) //Genero desde a2
                    {
                        if(b2<b1)//Genero hasta b2
                        {
                            Mu[i][j]=(b2-a2)*Random()+a2;
                            Mu[j][i]=Mu[i][j];
                        }
                        else //Genero hasta b1
                        {
                            Mu[i][j]=(b1-a2)*Random()+a2;
                            Mu[j][i]=Mu[i][j];
                        }
                    }
                    else //Genero desde a1
                    {
                        if(b2<b1)//Genero hasta b2
                        {
                            Mu[i][j]=(b2-a1)*Random()+a1;
                            Mu[j][i]=Mu[i][j];
                        }
                        else //Genero hasta b1
                        {
                            Mu[i][j]=(b1-a1)*Random()+a1;
                            Mu[j][i]=Mu[i][j];
                        }
                    }
            }
            break;
        default://Resto de elementos
            switch (i) {
                case 0:
                    switch (j) {
                        case 117: //Subcaso Mu[0][N-1]
//                            printf("\tResto, Mu[0][N-1]\n");
                            a1=Mu[0][Nmax-2]+Mu[1][Nmax-1]-Mu[1][Nmax-2];
```

```
                        b1=Mu[1][Nmax-1];
                        a2=0.0;
                        b2=Mu[0][Nmax-2];
                        if(a1<a2) //Genero desde a2
                        {
                            if(b2<b1)//Genero hasta b2
                            {
                                Mu[0][Nmax-1]=(b2-a2)*Random()+a2;
                                Mu[Nmax-1][0]=Mu[0][Nmax-1];
                            }
                            else //Genero hasta b1
                            {
                                Mu[0][Nmax-1]=(b1-a2)*Random()+a2;
                                Mu[Nmax-1][0]=Mu[0][Nmax-1];
                            }
                        }
                        else //Genero desde a1
                        {
                            if(b2<b1)//Genero hasta b2
                            {
                                Mu[0][Nmax-1]=(b2-a1)*Random()+a1;
                                Mu[Nmax-1][0]=Mu[0][Nmax-1];
                            }
                            else //Genero hasta b1
                            {
                                Mu[0][Nmax-1]=(b1-a1)*Random()+a1;
                                Mu[Nmax-1][0]=Mu[0][Nmax-1];
                            }
                        }
                        break;
                    default://Resto en la fila 0
//                        printf("\tResto, Mu[0][j]\n");
                        a1=Mu[0][j-1]+Mu[1][j]-Mu[1][j-1];
                        b1=Mu[0][j+1]+Mu[1][j]-Mu[1][j+1];

                        a2=Mu[0][j+1];
                        b2=Mu[0][j-1];
                        if(a1<a2) //Genero desde a2
                        {
                          if(b2<b1)//Genero hasta b2
                          {
                              Mu[0][j]=(b2-a2)*Random()+a2;
                              Mu[j][0]=Mu[0][j];
                          }
                          else //Genero hasta b1
```

```
                                {
                                    Mu[0][j]=(b1-a2)*Random()+a2;
                                    Mu[j][0]=Mu[0][j];
                                }
                            }
                            else //Genero desde a1
                            {
                              if(b2<b1)//Genero hasta b2
                              {
                                  Mu[0][j]=(b2-a1)*Random()+a1;
                                  Mu[j][0]=Mu[0][j];
                              }
                              else //Genero hasta b1
                              {
                                  Mu[0][j]=(b1-a1)*Random()+a1;
                                  Mu[j][0]=Mu[0][j];
                              }
                            }
                        }
                        break;
                    default:
                        switch (j) {
                            case 117: //Subcaso Ultima columna
//                                printf("\tResto, Ultima columna\n");
                            a1=Mu[i-1][Nmax-1];
                            b1=Mu[i][Nmax-2]+Mu[i-1][Nmax-1]-Mu[i-1][Nmax-2];

                            a2=Mu[i][Nmax-2]+Mu[i+1][Nmax-1]-Mu[i+1][Nmax-2];
                            b2=Mu[i+1][Nmax-1];
                            if(a1<a2) //Genero desde a2
                            {
                                if(b2<b1)//Genero hasta b2
                                {
                                    Mu[i][Nmax-1]=(b2-a2)*Random()+a2;
                                    Mu[Nmax-1][i]=Mu[i][Nmax-1];
                                }
                                else //Genero hasta b1
                                {
                                    Mu[i][Nmax-1]=(b1-a2)*Random()+a2;
                                    Mu[Nmax-1][i]=Mu[i][Nmax-1];
                                }
                            }
                            else //Genero desde a1
                            {
                                if(b2<b1)//Genero hasta b2
```

```
                {
                    Mu[i][Nmax-1]=(b2-a1)*Random()+a1;
                    Mu[Nmax-1][i]=Mu[i][Nmax-1];
                }
                else //Genero hasta b1
                {
                    Mu[i][Nmax-1]=(b1-a1)*Random()+a1;
                    Mu[Nmax-1][i]=Mu[i][Nmax-1];
                }
            }
            break;
        default://Elementos restantes
//              printf("\tResto, Ultima columna\n");
            a1=Mu[i][j+1]-Mu[i-1][j+1]+Mu[i-1][j];
            b1=Mu[i][j-1]-Mu[i-1][j-1]+Mu[i-1][j];

            a2=Mu[i][j-1]+Mu[i+1][j]-Mu[i+1][j-1];
            b2=Mu[i][j+1]+Mu[i+1][j]-Mu[i+1][j+1];
            if(a1<a2) //Genero desde a2
            {
                if(b2<b1)//Genero hasta b2
                {
                    Mu[i][j]=(b2-a2)*Random()+a2;
                    Mu[j][i]=Mu[i][j];
                }
                else //Genero hasta b1
                {
                    Mu[i][j]=(b1-a2)*Random()+a2;
                    Mu[j][i]=Mu[i][j];
                }
            }
            else //Genero desde a1
            {
                if(b2<b1)//Genero hasta b2
                {
                    Mu[i][j]=(b2-a1)*Random()+a1;
                    Mu[j][i]=Mu[i][j];
                }
                else //Genero hasta b1
                {
                    Mu[i][j]=(b1-a1)*Random()+a1;
                    Mu[j][i]=Mu[i][j];
                }
            }
    }
```

```
            }
    }
}
void iguala (double **Mu,double **Mu2,int *N)
{
    int i,j,Nmax;
    i=j=Nmax=0;
    Nmax=*N;

    for(i=0;i<Nmax;i++)
    {for(j=i;j<Nmax;j++)
     {
        Mu[i][j]=Mu2[i][j];
        Mu[j][i]=Mu2[i][j];
     }
    }
}
void igualacov (double **C,double **C2,double **Cov,double **Cov2,int *is, int *js,int *N)
{
    int i,j,m,Nmax;
    int ki,kj;
    i=j=m=Nmax=0;
    ki=kj=0;
    Nmax=*N;
    i=*is;
    j=*js;

    if(j>(i+1))
    {
        if(i==0)
        {
            ki=i;
            for(kj=j;kj<Nmax;kj++)
            {
                C[ki][kj]=C2[ki][kj];
                C[kj][ki]=C2[kj][ki];
                Cov[ki][kj]=Cov2[ki][kj];
                Cov[kj][ki]=Cov2[kj][ki];
            }
        }
        else
        {

            if(j==Nmax-1)
            {
```

```
            kj=j;
            for(ki=i;ki>-1;ki--)
            {
                C[ki][kj]=C2[ki][kj];
                C[kj][ki]=C2[kj][ki];
                Cov[ki][kj]=Cov2[ki][kj];
                Cov[kj][ki]=Cov2[kj][ki];
            }
        }
        else
        {
            for(ki=i;ki>-1;ki--)
            {
                for(kj=j;kj<Nmax;kj++)
                {
                    C[ki][kj]=C2[ki][kj];
                    C[kj][ki]=C2[kj][ki];
                    Cov[ki][kj]=Cov2[ki][kj];
                    Cov[kj][ki]=Cov2[kj][ki];
                }
            }
        }
    }
}
else
{
    if(i==j)//Diag. Principal
    {
        C[i][i]=C2[i][i];
        Cov[i][i]=Cov2[i][i];
        if(i<3)
        {
            for(ki=i;ki>-1;ki--)
            {
                for(kj=ki+2;kj<Nmax;kj++)
                {
                    C[ki][kj]=C2[ki][kj];
                    C[kj][ki]=C2[kj][ki];
                    Cov[ki][kj]=Cov2[ki][kj];
                    Cov[kj][ki]=Cov2[kj][ki];
                }
            }
            if(i==0)
            {
                Cov[0][1]=Cov2[0][1];
```

```
            Cov[1][0]=Cov2[1][0];
        }
        else
        {
            Cov[i][j+1]=Cov2[i][j+1];
            Cov[j+1][i]=Cov2[j+1][i];
            Cov[i-1][j]=Cov2[i-1][j];
            Cov[j][i-1]=Cov2[j][i-1];
        }
    }
    else
    {
        for(kj=j;kj<Nmax;kj++)
        {
            for(ki=kj-2;ki>-1;ki--)
            {
                C[ki][kj]=C2[ki][kj];
                C[kj][ki]=C2[kj][ki];
                Cov[ki][kj]=Cov2[ki][kj];
                Cov[kj][ki]=Cov2[kj][ki];
            }
        }
        if(i==Nmax-1)
        {
            Cov[i-1][j]=Cov2[i-1][j];
            Cov[j][i-1]=Cov2[j][i-1];
        }
        else
        {
            Cov[i][j+1]=Cov2[i][j+1];
            Cov[j+1][i]=Cov2[j+1][i];
            Cov[i-1][j]=Cov2[i-1][j];
            Cov[j][i-1]=Cov2[j][i-1];
        }
    }
}
else
{
    if(j<3)
    {
        C[i][j]=C2[i][j];
        C[j][i]=C2[j][i];
        Cov[i][j]=Cov2[i][j];
        Cov[j][i]=Cov2[j][i];
        for(ki=i;ki>-1;ki--)
```

```
                {
                    for(kj=ki+2;kj<Nmax;kj++)
                    {
                        C[ki][kj]=C2[ki][kj];
                        C[kj][ki]=C2[kj][ki];
                        Cov[ki][kj]=Cov2[ki][kj];
                        Cov[kj][ki]=Cov2[kj][ki];
                    }
                }
            }
            else
            {
                for(ki=i;ki>-1;ki--)
                {
                    for(kj=j;kj<Nmax;kj++)
                    {
                        C[ki][kj]=C2[ki][kj];
                        C[kj][ki]=C2[kj][ki];
                        Cov[ki][kj]=Cov2[ki][kj];
                        Cov[kj][ki]=Cov2[kj][ki];
                    }
                }
            }
        }
    }
}

void matrizC (double **C,double **Cov, double **Mu, int *N)
{
    //Variables enteras (int)
        int i,j,m,d,Nmax;//Indice de fila, columna,para el bucle del calculo del sumatorio p
    //Variables decimales (double)
        double sum,num,den;//Variable auxialiar para la suma, numerador y denominador
    //Inicialización
        i=j=m=Nmax=0; sum=num=den=0.0;
    //Asignar a Nmax el valor del puntero N, que contiene la dimension de las matrices
        Nmax=*N;

    for(d=0;d<Nmax;d++)//Bucle de las diagonales
    {for(i=0;i<(Nmax-d);i++)//
        {
            j=i+d;//Asigno la columna
            sum=num=den=0.0;
            if(j>(i+1))//Si es j>i+1 le sumamos la expresión delsumatorio
            {
```

```
                for(m=(i+1);m<j;m++)
                {
                    num=(Mu[i][m-1]-Mu[i][m])*(C[j][j]-C[m][j]);
                    den=(1-C[m][m]);
                    sum+=num/den;
                }
                C[i][j]=(Mu[i][j]+sum);
                Cov[i][j]=C[i][j]-(Mu[i][i]*Mu[j][j]);
                Cov[j][i]=Cov[i][j];
                sum=num=den=0.0;
            }
            else
            {
                C[i][j]=Mu[i][j];
                Cov[i][j]=C[i][j]-(Mu[i][i]*Mu[j][j]);
                Cov[j][i]=Cov[i][j];
            }
        }
    }
}


void pasaacovsred (double **Cold, double **Cnew, double **Cov, double **Mu,double **Munew, i
{
    int i,j,ki,kj,m,Nmax;
    double sum;
    double a,b,c,c2,d;
    i=j=ki=kj=m=Nmax=0;
    sum=0.0;
    a=b=c=c2=d=0.0;
    Nmax=*N;
    i=*is; j=*js;
    double *Gamma;
    Gamma=malloc((Nmax-4)*sizeof(double));

    if(j>(i+1))
    {
        if(i==0)
        {
            if(j==(Nmax-1))//En este caso solo cambia el mismo
            {
                Cnew[0][j]=Cold[0][j]-Mu[0][j]+Munew[0][j];
                Cov[0][j]=Cov[0][j]+Cnew[0][j]-Cold[0][j];
            }
            else//En este caso hay que cambiar columna a columnna hasta la ultima
```

```
    {
        a=Mu[0][j]-Munew[0][j];
        Cnew[0][j]=Cold[0][j]-a;
        Cov[0][j]=Cov[0][j]+Cnew[0][j]-Cold[0][j];
        a= (Mu[0][j]-Munew[0][j])/(1.0-Cold[j][j]);
        b= (Munew[0][j]-Mu[0][j])/(1.0-Cold[j+1][j+1]);

        Cnew[0][j+1]=Cold[0][j+1]+a*(Cold[j+1][j+1]-Cold[j][j+1]);
        Cov[0][j+1]=Cov[0][j+1]+Cnew[0][j+1]-Cold[0][j+1];
        for(kj=j+2;kj<Nmax;kj++)
        {
            Cnew[0][kj]=Cold[0][kj]+a*(Cold[kj][kj]-Cold[j][kj])+b*(Cold[kj][kj]-Col
            Cov[0][kj]=Cov[0][kj]+Cnew[0][kj]-Cold[0][kj];
        }
    }
}
else
{
    if(j==(Nmax-1))
    {
        Cnew[i][j]=Cold[i][j]-Mu[i][j]+Munew[i][j];
        Cov[i][j]=Cov[i][j]+Cnew[i][j]-Cold[i][j];
        for(ki=i-1;ki>-1;ki--)
        {
            for(m=ki+1;m<i+1;m++)
            {
                sum+= ((Mu[ki][m-1]-Mu[ki][m])*(Cold[m][j]-Cnew[m][j]))/(1.0-Cold[m]
            }
            Cnew[ki][j]=Cold[ki][j]+sum;
            Cov[ki][j]=Cov[ki][j]+Cnew[ki][j]-Cold[ki][j];
            sum=0.0;
        }
    }
    else//Quiza se pueda acortar poniendo una condicion a los elementos Mu[1][j] que
    {
        a=Mu[i][j]-Munew[i][j];
        Cnew[i][j]=Cold[i][j]-a;
        Cov[i][j]=Cov[i][j]+Cnew[i][j]-Cold[i][j];
        a=a/(1.0-Cold[j][j]);
        b=(Munew[i][j]-Mu[i][j])/(1.0-Cold[j+1][j+1]);
        //Primero el que esta en la misma fila y una posicion mas a la derecha
        Cnew[i][j+1]=Cold[i][j+1]+a*(Cold[j+1][j+1]-Cold[j][j+1]);
        Cov[i][j+1]=Cov[i][j+1]+Cnew[i][j+1]-Cold[i][j+1];

        for(kj=j+2;kj<Nmax;kj++)
```

```
            {
                Cnew[i][kj]=Cold[i][kj]+a*(Cold[kj][kj]-Cold[j][kj])+b*(Cold[kj][kj]-Col
                Cov[i][kj]=Cov[i][kj]+Cnew[i][kj]-Cold[i][kj];
            }
            //Despues los de filas superiores
            for(ki=i-1;ki>-1;ki--)
            {
                //Generamos los coeficientes Gamma que son comunes para todos los elemer
                for(m=0;m<i-ki;m++)
                {
                    Gamma[m]=(Mu[ki][ki+m]-Mu[ki][ki+m+1])/(1.0-Cold[ki+m+1][ki+m+1]);
                }
                for(kj=j;kj<Nmax;kj++)
                {
                    for(m=0;m<i-ki;m++)
                    {
                        sum+=Gamma[m]*(Cold[ki+m+1][kj]-Cnew[ki+m+1][kj]);
                    }
                    Cnew[ki][kj]=Cold[ki][kj]+sum;
                    Cov[ki][kj]=Cov[ki][kj]+Cnew[ki][kj]-Cold[ki][kj];
                    sum=0.0;
                }
            }
        }
    }
}
else
{
    if(i==j)//Diagonal Principal
    {
        if(i<3)
        {
            a=Munew[i][i]-Mu[i][i];
            Cnew[i][i]=Munew[i][i];
            Cov[i][i]=Cnew[i][i]-(Munew[i][i]*Munew[i][i]);
            switch (i) {
                case 0:
                    Cov[0][1]=Cov[0][1]-(a*Mu[1][1]);
                    a=a/(1.0-Cold[1][1]);
                    for(kj=2;kj<Nmax;kj++)
                    {
                        Cnew[0][kj]=Cold[0][kj]+a*(Cold[kj][kj]-Cold[1][kj]);
                        Cov[0][kj]=Cnew[0][kj]-(Munew[0][0]*Mu[kj][kj]);
                    }
                    break;
```

```
case 1:
    Cov[0][1]=Cov[0][1]-(a*Mu[0][0]);
    Cov[1][2]=Cov[1][2]-(a*Mu[2][2]);
    a=a/(1.0-Cold[2][2]);
    for(kj=3;kj<Nmax;kj++) //Los de tu misma fila
    {
        Cnew[1][kj]=Cold[1][kj]+a*(Cold[kj][kj]-Cold[2][kj]);
        Cov[1][kj]=Cnew[1][kj]-(Munew[1][1]*Mu[kj][kj]);
    }
    a=1.0-Cnew[1][1];
    b=1.0-Cold[1][1];
    c=(Mu[0][0]-Mu[0][1])/(a*b);
    d=b-a;
    Cnew[0][2]=Cold[0][2]+c*(Cold[2][2]-Cold[1][2])*d;
    Cov[0][2]=Cov[0][2]+Cnew[0][2]-Cold[0][2];
    for(kj=3;kj<Nmax;kj++)
    {
        Cnew[0][kj]=Cold[0][kj]+c*(a*Cold[1][kj]-b*Cnew[1][kj]+d*Cold[kj
        Cov[0][kj]=Cov[0][kj]+Cnew[0][kj]-Cold[0][kj];
    }
    break;
case 2:
    Cov[1][2]=Cov[1][2]-a*Mu[1][1];
    Cov[2][3]=Cov[2][3]-a*Mu[3][3];
    a=a/(1.0-Cold[3][3]);
    for(kj=4;kj<Nmax;kj++)//Elementos de la fila 2
    {
        Cnew[2][kj]=Cold[2][kj]+a*(Cold[kj][kj]-Cold[3][kj]);
        Cov[2][kj]=Cnew[2][kj]-(Munew[2][2]*Mu[kj][kj]);
    }
    a=1.0-Cnew[2][2];
    b=1.0-Cold[2][2];
    c=(Mu[1][1]-Mu[1][2])/(a*b);
    d=b-a;
    Cnew[1][3]=Cold[1][3]+c*(Cold[3][3]-Cold[2][3])*d;
    Cov[1][3]=Cov[1][3]-Cold[1][3]+Cnew[1][3];

    for(kj=4;kj<Nmax;kj++)//Elementos de la fila 1
    {
        Cnew[1][kj]=Cold[1][kj]+c*(a*Cold[2][kj]-b*Cnew[2][kj]+d*Cold[kj
        Cov[1][kj]=Cov[1][kj]+Cnew[1][kj]-Cold[1][kj];
    }

    c=(Mu[0][0]-Mu[0][1])/(1.0-Cold[1][1]);
    c2=(Mu[0][1]-Mu[0][2])/(a*b);
```

```
            Cnew[0][2]=Cold[0][2]+c*d;
            Cov[0][2]=Cnew[0][2]-(Munew[2][2]*Mu[0][0]);

            Cnew[0][3]=Cold[0][3]+c*(Cold[1][3]-Cnew[1][3])+c2*d*(Cold[3][3]-Col
            Cov[0][3]=Cov[0][3]+Cnew[0][3]-Cold[0][3];

            for(kj=4;kj<Nmax;kj++)
            {
                Cnew[0][kj]=Cold[0][kj]+c*(Cold[1][kj]-Cnew[1][kj])+c2*(a*Cold[2
                Cov[0][kj]=Cov[0][kj]+Cnew[0][kj]-Cold[0][kj];
            }
            break;
        }
    }
    else//i>=3
    {
        if(i==Nmax-1)
        {
            Cnew[i][i]=Munew[i][i];
            Cov[i][i]=Cnew[i][i]-(Munew[i][i]*Munew[i][i]);
            a=Munew[i][i]-Mu[i][i];
            Cnew[i-2][i]=Cold[i-2][i]+a*((Mu[i-2][i-2]-Mu[i-2][i-1])/(1.0-Cold[i-1]
            Cov[i-2][i]=Cnew[i-2][i]-(Mu[i-2][i-2]*Munew[i][i]);
            for(ki=i-3;ki>-1;ki--)
            {
                for(m=ki+1;m<i-1;m++)
                {
                    sum+=((Mu[ki][m-1]-Mu[ki][m])*(a+Cold[m][i]-Cnew[m][i]))/(1.0-Co
                }
                Cnew[ki][i]=Cold[ki][i]+sum+(((Mu[ki][i-2]-Mu[ki][i-1])*a)/(1.0-Cold
                Cov[ki][i]= Cnew[ki][i]-(Mu[ki][ki]*Munew[i][i]);
                sum=0.0;
            }
            //Ahora calculamos la covarianza que esta una fila superior
            Cov[i-1][i]=Cold[i-1][i]-(Mu[i-1][i-1]*Munew[i][i]);// es Cov[N-2][N-1]
        }
        else
        {
            Cnew[i][i]=Munew[i][i];
            Cov[i][i]=Cnew[i][i]-(Munew[i][i]*Munew[i][i]);
            a=Munew[i][i]-Mu[i][i];
            Cnew[i-2][i]=Cold[i-2][i]+a*((Mu[i-2][i-2]-Mu[i-2][i-1])/(1.0-Cold[i-1]
            Cov[i-2][i]=Cnew[i-2][i]-(Mu[i-2][i-2]*Munew[i][i]);
            for(ki=i-3;ki>-1;ki--)
            {
```

```
    for(m=ki+1;m<i-1;m++)
    {
        sum+=((Mu[ki][m-1]-Mu[ki][m])*(a+Cold[m][i]-Cnew[m][i]))/(1.0-Co
    }
    Cnew[ki][i]=Cold[ki][i]+sum+(((Mu[ki][i-2]-Mu[ki][i-1])*a)/(1.0-Cold
    Cov[ki][i]= Cnew[ki][i]-(Mu[ki][ki]*Munew[i][i]);
    sum=0.0;
}
//Ahora calculamos las covarianzas en L
Cov[i-1][i]=Cold[i-1][i]-(Mu[i-1][i-1]*Munew[i][i]);
Cov[i][i+1]=Cold[i][i+1]-(Munew[i][i]*Mu[i+1][i+1]);

if(i==Nmax-2)//Caso particular porque solo rellena la ultima columna y e
{
    a=1.0-Cnew[i][i];
    b=1.0-Cold[i][i];
    d=(b-a);
    c=((Cold[Nmax-1][Nmax-1]-Cold[Nmax-2][Nmax-1])*d)/(a*b);
    Cnew[Nmax-3][Nmax-1]=Cold[Nmax-3][Nmax-1]+c*(Mu[Nmax-3][Nmax-3]-Mu[N
    Cov[Nmax-3][Nmax-1]=Cov[Nmax-3][Nmax-1]+Cnew[Nmax-3][Nmax-1]-Cold[Nm
    for(ki=i-2;ki>-1;ki--)
    {
        for(m=ki+1;m<i;m++)
        {
            sum+=((Mu[ki][m-1]-Mu[ki][m])*(Cold[m][Nmax-1]-Cnew[m][Nmax-
        }
        Cnew[ki][Nmax-1]=Cold[ki][Nmax-1]+sum+c*(Mu[ki][i-1]-Mu[ki][i]);
        Cov[ki][Nmax-1]=Cov[ki][Nmax-1]+Cnew[ki][Nmax-1]-Cold[ki][Nmax-1
        sum=0.0;
    }
}
else
{
    //Calculamos la columna adyacente
    a=1.0-Cnew[i][i];
    b=1.0-Cold[i][i];
    d=(b-a);
    c=((Cold[i+1][i+1]-Cold[i][i+1])*d)/(a*b);
    Cnew[i-1][i+1]=Cold[i-1][i+1]+c*(Mu[i-1][i-1]-Mu[i-1][i]);
    Cov[i-1][i+1]=Cov[i-1][i+1]+Cnew[i-1][i+1]-Cold[i-1][i+1];
    for(ki=i-2;ki>-1;ki--)
    {
        for(m=ki+1;m<i;m++)
        {
            sum+=((Mu[ki][m-1]-Mu[ki][m])*(Cold[m][i+1]-Cnew[m][i+1]))/(
```

```
                    }
                    Cnew[ki][i+1]=Cold[ki][i+1]+sum+c*(Mu[ki][i-1]-Mu[ki][i]);
                    Cov[ki][i+1]=Cov[ki][i+1]+Cnew[ki][i+1]-Cold[ki][i+1];
                    sum=0.0;
                }
                //Calculamos las columnas restantes hasta la N-1, que empiezan desde
                for(kj=i+2;kj<Nmax;kj++)
                {
                    Cnew[i][kj]=Cold[i][kj]+((Munew[i][i]-Mu[i][i])*(Cold[kj][kj]-Co
                    Cov[i][kj]=Cnew[i][kj]-(Munew[i][i]*Mu[kj][kj]);
                    c=(a*Cold[i][kj]-b*Cnew[i][kj]+d*Cold[kj][kj])/(a*b);
                    Cnew[i-1][kj]=Cnew[i-1][kj] + c*(Mu[i-1][i-1]-Mu[i-1][i]);//Pued
                    Cov[i-1][kj]=Cov[i-1][kj]+Cnew[i-1][kj]-Cold[i-1][kj];
                    for(ki=i-2;ki>-1;ki--)
                    {
                        for(m=ki+1;m<i;m++)
                        {
                            sum+=((Mu[ki][m-1]-Mu[ki][m])*(Cold[m][kj]-Cnew[m][kj]))
                        }
                        Cnew[ki][kj]=Cold[ki][kj]+sum+c*(Mu[ki][i-1]-Mu[ki][i]);
                        Cov[ki][kj]=Cov[ki][kj]+Cnew[ki][kj]-Cold[ki][kj];
                        sum=0.0;
                    }
                }
            }
        }
    }

}
else//Diagonal Secundaria
{
    if(j<3)
    {
        Cnew[i][j]=Munew[i][j];
        Cov[i][j]=Cov[i][j]+Cnew[i][j]-Cold[i][j];
        if(j==1)
        {
            a=(Mu[0][1]-Munew[0][1])/(1.0-Cold[1][1]);
            b=(Munew[0][1]-Mu[0][1])/(1.0-Cold[2][2]);
            Cnew[0][2]=Cold[0][2]+a*(Cold[2][2]-Cold[1][2]);
            Cov[0][2]=Cov[0][2]+Cnew[0][2]-Cold[0][2];
            for(kj=3;kj<Nmax;kj++)
            {
                Cnew[0][kj]=Cold[0][kj]+a*(Cold[kj][kj]-Cold[1][kj])+b*(Cold[kj][kj]
                Cov[0][kj]=Cov[0][kj]+Cnew[0][kj]-Cold[0][kj];
```

```
            }
        }
        else
        {
            a=(Mu[1][2]-Munew[1][2])/(1.0-Cold[2][2]);
            b=(Munew[1][2]-Mu[1][2])/(1.0-Cold[3][3]);
            Cnew[1][3]=Cold[1][3]+a*(Cold[3][3]-Cold[2][3]);
            Cov[1][3]=Cov[1][3]+Cnew[1][3]-Cold[1][3];
            for(kj=4;kj<Nmax;kj++)
            {
                Cnew[1][kj]=Cold[1][kj]+a*(Cold[kj][kj]-Cold[2][kj])+b*(Cold[kj][kj]
                Cov[1][kj]=Cov[1][kj]+Cnew[1][kj]-Cold[1][kj];
            }
            c=(Mu[0][0]-Mu[0][1])/(1.0-Cold[1][1]);
            for(kj=2;kj<Nmax;kj++)
            {
                Cnew[0][kj]=Cold[0][kj]+c*(Cold[1][kj]-Cnew[1][kj]);
                Cov[0][kj]=Cov[0][kj]+Cnew[0][kj]-Cold[0][kj];
            }

        }
    }
    else
    {
        Cnew[i][j]=Munew[i][j];
        Cov[i][j]=Cov[i][j]+Cnew[i][j]-Cold[i][j];
        if(j==Nmax-1)
        {
            for(ki=i-1;ki>-1;ki--)
            {
                for(m=ki+1;m<i+1;m++)
                {
                    sum+=((Mu[ki][m-1]-Mu[ki][m])*(Cold[m][j]-Cnew[m][j]))/(1.0-Cold
                }
                Cnew[ki][j]=Cold[ki][j]+sum;
                Cov[ki][j]=Cov[ki][j]+Cnew[ki][j]-Cold[ki][j];
                sum=0.0;
            }
        }
        else
        {
            a=(Mu[i][j]-Munew[i][j])/(1.0-Cold[j][j]);
            b=(Munew[i][j]-Mu[i][j])/(1.0-Cold[j+1][j+1]);

            Cnew[i][j+1]=Cold[i][j+1]+a*(Cold[j+1][j+1]-Cold[j][j+1]);
```

```
                    Cov[i][j+1]=Cov[i][j+1]+Cnew[i][j+1]-Cold[i][j+1];
                    for(kj=j+2;kj<Nmax;kj++)//Los de su misma fila
                    {
                        Cnew[i][kj]=Cold[i][kj]+a*(Cold[kj][kj]-Cold[j][kj])+b*(Cold[kj][kj]
                        Cov[i][kj]=Cov[i][kj]+Cnew[i][kj]-Cold[i][kj];
                    }
                    //Los de filas superiores
                    for(ki=i-1;ki>-1;ki--)
                    {
                        for(kj=j;kj<Nmax;kj++)
                        {
                            for(m=ki+1;m<i+1;m++)
                            {
                                sum+=((Mu[ki][m-1]-Mu[ki][m])*(Cold[m][kj]-Cnew[m][kj]))/(1.
                            }
                            Cnew[ki][kj]=Cold[ki][kj]+sum;
                            Cov[ki][kj]=Cov[ki][kj]+Cnew[ki][kj]-Cold[ki][kj];
                            sum=0.0;
                        }
                    }
                }
            }
        }
    }
    free(Gamma);
}

void energia (double **Cexp,double **Cold,double *E,int *N)
{
   //Variables (int)
       int i,j,Nmax;//variables de fila, columna,Dim. de la matriz, variables que cuenta los
   //Variables (double)
       double aux,term;//Variables auxiliar para un termino de la energia, variable para cal
   //Inicializacion
       i=j=Nmax=0;aux=term=0.0;
   //Asignación de Nmax el valor del puntero N,dimensión de las matrices
       Nmax=*N;
   //Calculo de la energia, solo miramos los terminos de la parte triangular superior de la
   for(i=0;i<Nmax;i++)
   {
       for(j=i+1;j<Nmax;j++)
       {
           term=fabs(Cold[i][j] - Cexp[i][j]);
           aux+=term*term;
       }
```

```
  }
    *E=(double) aux;
}

void condicioninicial (double **Mu, double **Munew, double **Cold,double **Covold,double **C
{
    FILE *f2;
    FILE *f;
    f2=fopen("/Users/Joaquin/Desktop/desarrolloEmin_versionmodp1.txt","w");
    f=fopen("/Users/Joaquin/Desktop/desarrolloEmin_versionmodp2.txt","w");
    //Variables int
        int iter,itermax;
        int k,kmax;
        int kw,kwmax;
        int ka,kamax;
        int kc, kcmax;
        int i,j,Nmax,M;
        int cont;
        iter=itermax=0; k=kmax=0; kw=kwmax=0; ka=kamax=0; kc=kcmax=0; i=j=Nmax=M=0; cont=0;
        //Evaluacion
        Nmax=*N;
        itermax=10;
        kmax=kcmax=kamax=50;
        kwmax=10;
        M=(Nmax*(Nmax+1))/2;
    //Variables double
        double a,w,c,eps,epsw,cmax;
        double T,beta,alpha,Enew,Emin,Eold;
        double ale,expnew;
        a=w=c=eps=epsw=cmax=0.0; T=beta=alpha=Enew=Emin=Eold=0.0; ale=expnew=0.0;
        //Evaluacion
        w=1.0;
        eps=pow(10,-4);
        cmax=(1.0-4.0*eps)/3.0;
        epsw=0.1;

        T=10.0;
        beta=1.0/T;
        alpha=0.75;

    //Matrices auxiliares
        double **Mumin;
        Mumin=malloc(Nmax*sizeof(double*)); for(i=0;i<Nmax;i++){Mumin[i]=malloc(Nmax*sizeof(

    //Parte 1, optimización de los parametros  (a,w,c) para eps cte,
```

```
Emin=Eold=Enew=1000.0;
for(kw=0;kw<kwmax+1;kw++)
{
    for(iter=0;iter<itermax+1;iter++)//Bucle de temperaturas
    {
        do{
            c=cmax*Random();
        }while(c==cmax);
        for(kc=0;kc<kcmax;kc++)//Bucle de los distintos valores de c
        {
            //Aqui genero a al azar.
            a=0.5*Random();
            for(ka=0;ka<kamax;ka++)//Bucle para tener 50 cambios de a
            {
                for(k=0;k<kmax;k++)//Bucle para realizar cambios en los diagonales sigui
                {
                    matrizporparametros(Munew, &c, &a, &eps, &w, &Nmax); //Creacion de l
                    matrizC(Cnew, Covnew, Munew, &Nmax); //Matriz de covarianza asociada
                    energia(Cexp, Covnew, &Enew, &Nmax); //Energia asociada
                    if(Enew<Eold)
                    {
                        Eold=Enew;
                        iguala(Mu, Munew, &Nmax);
                        printf("%lf %lf %lf %lf %lf %E\n",w,c,a,Emin,Eold,T);
                        if(Enew<Emin)
                        {
                            Emin=Enew;
                            iguala(Mumin, Munew, &Nmax);
                            fprintf(f2,"%d %lf %lf %lf %lf %lf %E\n",cont,w,c,a,Emin,Eol
                            cont++;
                        }
                    }
                    else
                    {
                        ale=Random();
                        expnew=exp(-beta*(Enew-Eold));
                        if(expnew>=ale)
                        {
                            Eold=Enew;
                            iguala(Mu, Munew, &Nmax);
                            printf("%lf %lf %lf %lf %lf %E\n",w,c,a,Emin,Eold,T);
                        }
                        else
                        {
                            iguala(Munew, Mu, &Nmax);
```

```
                            printf("%lf %lf %lf %lf %lf %E\n",w,c,a,Emin,Eold,T);
                        }
                    }

                }
                //Incremento de a y si se excede uso codicion de contorno tipo espejo
                a=a*1.1;
                if(a>1.0)//Le doy un poco de margen
                {
                    a=a-1.0;
                }
            }
            //Incremento c y si se excede uso codicion de contorno tipo espejo
            c=c*1.1;
            if(c>cmax)
            {
                c=c-cmax;
            }
        }
        T=T*alpha;
        beta=1.0/T;
    }
    w=w-epsw;
    T=10.0;
    beta=1.0/T;
    Eold=Enew=1000.0;
}
fclose(f2);
//PARTE 2) Dejar los diagonales libres
    //Mumin es la matriz a batir, pero voy a dejar Muold como está ya que asía podemos r
        matrizC(Cold, Covold, Mu, &Nmax);//como usaremos pasaacovsred debemos guardar en
        iguala(Munew,Mu,&Nmax);
        iguala(Cnew, Cold, &Nmax);
        iguala(Covnew,Covold,&Nmax);
    //Parametros de MC
        T=1.0;
        beta=1.0/T;
        kmax=100*Nmax; //Cien cambios en la diagonal
        itermax=20;//Cambios de temperatura
        cont=0;
for(iter=0;iter<itermax+1;iter++)
{
   for(k=0;k<kmax+1;k++)
   {
       //Primero elegiremos que termino cambiar al azar.
```

```
        do{
            i=floor(Nmax*Random());
        }while(i==Nmax);
        switchcreaunmu(Munew, &i, &i, &Nmax);
        pasaacovsred(Cold, Cnew, Covnew, Mu, Munew, &i, &i, &Nmax);
        energia(Cexp,Covnew,&Enew,&Nmax);
        if(Enew<Eold)
        {
            Eold=Enew;
            Mu[i][i]=Munew[i][i];
            igualacov(Cold, Cnew, Covold, Covnew, &i, &i, &Nmax);
            printf("%d %lf %lf %lf\n",(k+iter*kmax),Eold,Emin,T);
            if(Enew<Emin)
            {
                Emin=Enew;
                Mumin[i][i]=Munew[i][i];
                fprintf(f,"%d %lf %lf %lf %lf %lf %E\n",cont,w,c,a,Emin,Eold,T);
                cont++;
            }
        }
        else
        {
            ale=Random();
            expnew=exp(-beta*(Enew-Eold));
            if(expnew>=ale)
            {
                Eold=Enew;
                Mu[i][i]=Munew[i][i];
                igualacov(Cold, Cnew, Covold, Covnew, &i, &i, &Nmax);
                printf("%d %lf %lf %lf\n",(k+iter*kmax),Eold,Emin,T);
            }
            else
            {
                Munew[i][i]=Mu[i][i];
                igualacov(Cnew, Cold, Covnew, Covold, &i, &i, &Nmax);
                printf("%d %lf %lf %lf\n",(k+iter*kmax),Eold,Emin,T);
            }
        }
    }
    T=T*alpha;
    beta=1.0/T;
}
fclose(f);
for(i=0;i<Nmax;i++)
{
```

```
        for(j=i;j<Nmax;j++)
        {
            Cold[i][j]=0.0;
            Covold[i][j]=0.0;
            Cnew[i][j]=0.0;
            Covnew[i][j]=0.0;
        }
    }
    matrizC(Cold, Covold, Mumin, &Nmax);
    energia(Cexp,Covold,&Emin,&Nmax);
    printf("Energia antes de igualar=%lf\n",Emin);
    iguala(Mu,Mumin,&Nmax);//La matriz de Emin es nuestra condicion inicial
    iguala(Munew,Mumin,&Nmax);

    for(i=0;i<Nmax;i++)
    {
        free(Mumin[i]);
    }
    free(Mumin);

}


void matrizporparametros (double **Mu, double *cs, double *as,double *epsa,double *ws,int *N
{
    int i,j,d, Nmax;
    i=j=d=Nmax=0;
    Nmax=*N;
    double c,a,eps,w;
    double mu1,mu2,muterm;
    double ale;
    c=a=eps=w=0.0;
    mu1=mu2=0.0;
    ale=0.0;
    c=*cs;
    a=*as;
    eps=*epsa;
    w=*ws;
    mu1=2.0*c+eps;
    mu2=((1.0+c)/(2.0))-eps;

    //Elementos Diagonales,
    for(i=0;i<Nmax;i++)
    {
        ale=Random();
```

```
        if(ale<=w)
        {
            Mu[i][i]=mu1;
        }
        else
        {
            Mu[i][i]=mu2;
        }
    }
    //RESTO DE ELEMENTOS, ESTOS LOS CALCULAMOS POR DIAGONALES, YA QUE COINCIDEN
    for(d=1;d<Nmax;d++)
    {
        muterm=c/(pow(d,a));
        for(i=0;i<(Nmax-d);i++)
        {
            j=i+d;
            Mu[i][j]=muterm;
            Mu[j][i]=Mu[i][j];
        }
    }
}

void calculatheta (double **Mu,double **Thetas, int *N)
{
    //Variables int
        int i,j,Nmax;//Indice de fila,columna para los bucles, indice para la dimension y li
    //Variables double
        double aux;
    //Matriz double para introducir los terminos extra que introduce Zamparo
        double **Muaux;

    //Inicializacion de las variables
        i=j=Nmax=0;aux=0.0;
    //Asignar a Nmax el valor del puntero N(dim de la matrices)
        Nmax=*N;
    //Creacion de la matriz Muaux que es (Nmax+2)X(Nmax+2) luego los indices iran de 0,...,N
        Muaux=malloc((Nmax+2)*sizeof(double*)); for(i=0;i<(Nmax+2);i++) Muaux[i]=malloc((Nma
    //Crear la Matriz Muaux apartir de la expresión de Zamparo
        //Primero ponemos que la fila 0 y la columna N+1 sean 0
        for(i=0;i<(Nmax+2);i++)
        {
            Muaux[0][i]=0.0;
            Muaux[i][Nmax+1]=0.0;
        }
        //Segundo asignar que los elementos (i+1,i) como 1, i=0,...,Nmax
```

```
        for(i=0;i<(Nmax+1);i++)
        {
            Muaux[i+1][i]=1.0;
        }
        //Ahora a rellener los otros terminos con los de la matriz Mu
        for(i=1;i<(Nmax+1);i++)
        {for(j=i;j<(Nmax+1);j++)
            {
                Muaux[i][j]=Mu[i-1][j-1];
            }

        }
    //Ya puedo calcular la matriz Theta, donde ahora i empieza en 1 y acaba en Nmax,
    for(i=0;i<Nmax;i++)
    {for(j=i;j<Nmax;j++)
        {
            if(i==j)
            {
                aux=log( (1.0-Muaux[i+1][i+1]) / (1.0-Muaux[i][i]-Muaux[i+1][i+1]+Muaux[i][i
                log( (Muaux[i+1][i+1]-Muaux[i][i+1]-Muaux[i+1][i+2]+Muaux[i][i+2]) / (1.0-Mu
                Thetas[i][i]=aux;
            }
            else
            {
                aux=log( (Muaux[i+2][j]-Muaux[i+1][j]-Muaux[i+2][j+1]+Muaux[i+1][j+1]) /(Mua
                Thetas[i][j]=aux;
                Thetas[j][i]=Thetas[i][j];
            }
        }
    }
}

void guardamu (double **Mu, int *N)
{
    FILE *f1;
    f1=fopen("./Mu_pruebacondini2.txt","w");
    int i,j,Nmax;
    i=j=Nmax=0;
    Nmax=*N;
    for(i=0;i<Nmax;i++)
    {  for(j=i;j<Nmax;j++)
        {
            fprintf(f1,"%d %d %.17E\n",i,j,Mu[i][j]);
            fprintf(f1,"%d %d %.17E\n",j,i,Mu[i][j]);
        }
```

```
    }
    fclose(f1);
}

void guardamuminima (double **Mu,double *E,double *T, int *N)
{
    FILE *f1;
    f1=fopen("./Mu_minima_pruebacondini2.txt","w");
    int i,j,Nmax;
    double Ene,Trelativa;
    i=j=Nmax=0;
    Ene=Trelativa=0.0;
    Nmax=*N;
    Ene=*E;
    Trelativa=*T;
    fprintf(f1,"#E=%lf  T=%E\n",Ene,Trelativa);
    for(i=0;i<Nmax;i++)
    {  for(j=i;j<Nmax;j++)
        {
            fprintf(f1,"%d %d %.17E\n",i,j,Mu[i][j]);
            fprintf(f1,"%d %d %.17E\n",j,i,Mu[i][j]);
        }
    }
    fclose(f1);
}
void guardamuinicial (double **Mu, int *N)
{
    FILE *f1;
    f1=fopen("./Mu_condinicial.txt","w");
    int i,j,Nmax;
    i=j=Nmax=0;
    Nmax=*N;
    for(i=0;i<Nmax;i++)
    {  for(j=i;j<Nmax;j++)
        {
            fprintf(f1,"%d %d %.17E\n",i,j,Mu[i][j]);
            fprintf(f1,"%d %d %.17E\n",j,i,Mu[i][j]);
        }
    }
    fclose(f1);
}


void guardaCovarianzaminima (double **Cov,double *E,double *T, int *N)
{
```

```
    FILE *f1;
    f1=fopen("./Cov_minima_pruebacondini2.txt","w");
    int i,j,Nmax;
    double Ene,Trelativa;
    i=j=Nmax=0;
    Ene=Trelativa=0.0;
    Nmax=*N;
    Ene=*E;
    Trelativa=*T;
    fprintf(f1,"#E=%lf  T=%E\n",Ene,Trelativa);
    for(i=0;i<Nmax;i++)
    {  for(j=i;j<Nmax;j++)
        {
            fprintf(f1,"%d %d %.17E\n",i,j,Cov[i][j]);
            fprintf(f1,"%d %d %.17E\n",j,i,Cov[i][j]);
        }
    }
    fclose(f1);
}

void guardatheta (double **Theta, int *N)
{
    FILE *f2;
    f2=fopen("./Thetaresult_pruebacondini2.txt","w");
    int i,j,Nmax;
    i=j=Nmax=0;
    Nmax=*N;
    for(i=0;i<Nmax;i++)
    {  for(j=i;j<Nmax;j++)
        {
            fprintf(f2,"%d %d %.17E\n",i,j,Theta[i][j]);
            fprintf(f2,"%d %d %.17E\n",j,i,Theta[j][i]);
        }
    }
    fclose(f2);
}
```

5.6. Anexo VI: Código de Poland Scheraga determinista

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

void lectura (double **Old,double **Cov,int *N);
```

```
void generasigma (int *Sigma,int *N);

void nuevatheta (double **Theta, int *Sigma, const double *x,int *N);
void DeThetaaMu (double **Theta,double **Mu, double **Lambda, int *N);

void matrizCorr (double **C,double **Cov, double **Corr,double **Mu, int *N);
double Energia (unsigned n , const double *x, double *grad,void * fdata);
void Energiav2 (unsigned n , const double *x, double *E);
void guarda ( double **Theta, double **Mu, double **Corr, const double *x, double *E);


int main() {

    //Variables enteras
        int i,N;
        i=N=0;
        N = 118;

    //Variables double
        double maxeval, opt_f;
        double E;
        maxeval=opt_f=0.0;
        E = 0.0;

        void *f_data;

    //Vectores
        double *x;
        x = malloc(4*sizeof(double));
    //Creamos el objeto nlopt, el Algoritmo es local derivate Subplex
        nlopt_opt opt = nlopt_create(NLOPT_LN_SBPLX,4);

    //Establecemos el objetivo de miminización
        nlopt_set_min_objective(opt,Energia, f_data );//Creo que de esta manera no hara falt

    //Lanzar la optimización
        nlopt_optimize(opt, x,&opt_f);

    //Destruir el objeto nlopt_opt opt
        nlopt_destroy(opt);

    //Observar los resultados por pantalla
        printf("El algortimo ha terminado\n");
        printf("-------------------------\n");
        printf("\t Valores Optimizados:\n");
```

```
        printf("\t\t Epsilon0 = %lf , Epsilon1 = %lf , q = %lf , alpha = %lf\n", x[0], x[1],
        printf("\t Energia final:\n");
        printf("\t\t E = %lf\n",opt_f);
    //Guardar resultados
         Energiav2(4,x,&E);
    //Liberar la memoria de las matrices utilizadas
        //Vectores
            free(x);
    //C'est fini
    return 0;
}
void lectura (double **Old,double **Cov,int *N)
{
   FILE *f2;
    f2=fopen("./covmatrix.txt","r");
   int i,j,Nmax;
   double aux;
    i=j=Nmax=0;aux=0.0; //Inicializaciones
    Nmax=*N;
    for(i=0;i<Nmax;i++)
    {for(j=0;j<Nmax;j++)
        {
            fscanf(f2,"%lf\t",&aux);
            Cov[i][j]=aux;
        }
        fscanf(f2,"\n");
    }
    fclose(f2);
    //Ahora pasamos a correlaciones
    for(i=0;i<Nmax;i++)
    {for(j=0;j<Nmax;j++)
        {
            Old[i][j]=Cov[i][j]/sqrt(Cov[i][i]*Cov[j][j]);
        }
    }
}

void generasigma (int *Sigma,int *N)
{
    FILE *f;
    f=fopen("./pc_cov.txt", "r");
    int i,Nmax,contpos,contneg;
    i=Nmax=contpos=contneg=0;
    Nmax=*N;
    double aux;
```

```
aux=0.0;
double *Auxiliar;
Auxiliar=malloc(Nmax*sizeof(double));
do{
    fscanf(f,"%d   %lf\n",&i,&aux);
    Auxiliar[i-1]=aux;
}while(feof(f)==0);
fclose(f);
i=0;
for(i=0;i<Nmax;i++)
{
    if(Auxiliar[i] >= 0.0)
    {
        contpos++;
    }
    else
    {
        contneg++;
    }
}
i=0;
if(contpos > contneg)//Guardamos los positivos como Sigma=1 y los negativos como 0
{
    for(i=0;i<Nmax;i++)
    {
        if(Auxiliar[i] >= 0.0)
        {
            Sigma[i]=1;
        }
        else
        {
            Sigma[i]=0;
        }
    }
}
else//Guardamos los positivos como Sigma=0 y los negativos como 1
{
    for(i=0;i<Nmax;i++)
    {
        if(Auxiliar[i] >= 0.0)
        {
            Sigma[i]=0;
        }
        else
        {
```

```
                Sigma[i]=1;
            }
        }
    }
    free(Auxiliar);
}
void nuevatheta (double **Theta, int *Sigma, const double *x,int *N)
{
    int i,j,Nmax;
    int num,dem;
    i=j=Nmax=0;
    num=dem=0;
    double cociente;
    cociente =0.0;
    Nmax=*N;

    for(i=0;i<Nmax;i++)
    {
        for(j=i;j<Nmax;j++)
        {
            if(i==j)//Elementos diagonales
            {
                if(Sigma[i]==1)//en este caso se le suma epsilonvalue1
                {
                    Theta[i][i] = x[1] + x[2] + x[3]*log(2.0);
                }
                else//en este caso se le suma epsilonvalue0
                {
                    Theta[i][i] = x[0] + x[2] + x[3]*log(2.0);
                }
            }
            else//Elementos fuera de la diagonal principal
            {
                num =(j-i+1)*(j-i+1);
                dem =(j-i)*(j-i+2);
                cociente = ((double)num) /((double) dem);
                Theta[i][j] = x[3]*log(cociente);
                Theta[j][i] = Theta[i][j];
            }
        }
    }
}
//Programa nuevo
void DeThetaaMu (double **Theta,double **Mu, double **Lambda, int *N)
{
```

```
int i,j,Nmax;
int h,kh,k;
double termj,termj_h,termj_kh;
double termk,termmu;
i=j=Nmax=0;
h=kh=k=0;
termj=termj_h=termj_kh=0.0;
termk=termmu=0.0;

Nmax=*N;

//Matriz auxiliar debido a las condiciones de contorno
double **Muaux;
Muaux=malloc((Nmax+2)*sizeof(double*)); for(i=0;i<(Nmax+2);i++) { Muaux[i]=malloc((Nmax+

//Calculamos los Lambda
    //Debido a la cod. de contorno Lambda[1][0]=1
        Lambda[1][0] = 1.0;
    //Ahora el resto, cuidado que Lambda[j+1][j]
        for(j=1; j < (Nmax+1); j++)
        {
            //Calculo del termj
                for(h = 1; h < j+1; h++)//Va hasta el h==j, para termj_h
                {
                    for(kh = h; kh < j+1; kh++)//Va hasta el kh==j, para termj_kh
                    {
                        termj_kh+=Theta[kh-1][j-1];
                    }
                    termj_h += Lambda[h][j-1]*exp(termj_kh);
                    termj_kh = 0.0;
                }
                termj = 1.0 / ( 1.0 + termj_h );
                termj_h = 0.0;

            //Calculo de los terminos Lambda de esa columna
                for(i=1; i < (j+2); i++)//Solo llega hasta el i==j+1
                {
                    if(i < (j+1))
                    {
                        for(k=i ; k<(j+1); k++)
                        {
                            termk += Theta[k-1][j-1];
                        }
                        Lambda[i][j] = termj*Lambda[i][j-1]*exp(termk);
                        termk=0.0;
```

```
                    }
                    else//Caso especial i==j+1
                    {
                        Lambda[j+1][j] = termj;
                    }
                }
            }

//Calculo de Muaux
    //1) Condiciones de contorno
        for(j=0; j < (Nmax+2); j++)
        {
            Muaux[0][j] = 0.0;        //Muaux[0][j]
            Muaux[j][Nmax+1] = 0.0; //Muaux[i][N+1]
        }
        for(i=0; i < (Nmax+1); i++)
        {
            Muaux[i+1][i] = 1.0;// Muaux[i+1][i] (corregida)
        }
    //2) Elementos de la diagonal principal
        for(i=Nmax; i>0; i--)//Desde i=N hasta el i==1
        {
            for(j=i; j<(Nmax+1); j++)
            {
                termmu += Lambda[i+1][j] * ( 1.0 - Muaux[j+1][j+1]);
            }
            Muaux[i][i] = 1.0-termmu;
            termmu = 0.0;
        }
    //3) Resto de elemento Muaux
        for(j=Nmax; j>1; j--)//Dese j==N hasta j==2
        {
            for(i=1; i<j; i++)//Desde 1 hasta el j-1
            {
                Muaux[i][j] = Muaux[i-1][j] + Muaux[i][j+1] - Muaux[i-1][j+1] + Lambda[i
            }
        }
//Adaptar los Muaux a la notación de C y guardarlos en Mu
//Entonces los Mus son los de la parte triangular desde i=1,...,N j=i,...,N y habra que
    for(i=1; i < (Nmax+1); i++)//hasta el Nmax
    {
        for(j=i; j < (Nmax+1); j++)//hasta el Nmax
        {
            Mu[i-1][j-1]=Muaux[i][j];
            Mu[j-1][i-1]=Muaux[i][j];
```

```
            }
        }
    //Liberamos la memoria de la matriz Muaux
        for(i=0; i<(Nmax+2); i++)
        {
            free(Muaux[i]);
        }
        free(Muaux);
    //Ya lo tenemos
}
void matrizCorr (double **C,double **Cov, double **Corr,double **Mu, int *N)
{
    //Variables enteras (int)
        int i,j,m,d,Nmax;//Indice de fila, columna,para el bucle del calculo del sumatorio p
    //Variables decimales (double)
        double sum,num,den;//Variable auxialiar para la suma, numerador y denominador
    //Inicialización
        i=j=m=Nmax=0; sum=num=den=0.0;
    //Asignar a Nmax el valor del puntero N, que contiene la dimension de las matrices
        Nmax=*N;

    //Primero calculamos los elementos diagonales
    for(i=0;i<Nmax;i++)
    {
        C[i][i]=Mu[i][i];
        Cov[i][i]=C[i][i]-(Mu[i][i]*Mu[i][i]);
        Corr[i][i]=1.0;
    }

    for(d=1;d<Nmax;d++)//Bucle de las diagonales
    {for(i=0;i<(Nmax-d);i++)//
        {
            j=i+d;//Asigno la columna
            sum=num=den=0.0;
            if(j>(i+1))//Si es j>i+1 le sumamos la expresión delsumatorio
            {
                for(m=(i+1);m<j;m++)
                {
                    num=(Mu[i][m-1]-Mu[i][m])*(C[j][j]-C[m][j]);
                    den=(1-C[m][m]);
                    sum+=num/den;
                }
                C[i][j]=(Mu[i][j]+sum);
                Cov[i][j]=C[i][j]-(Mu[i][i]*Mu[j][j]);
                Corr[i][j]=Cov[i][j]/sqrt(Cov[i][i]*Cov[j][j]);
```

```
                Cov[j][i]=Cov[i][j];
                Corr[j][i]=Corr[i][j];
                sum=num=den=0.0;
            }
            else
            {
                C[i][j]=Mu[i][j];
                Cov[i][j]=C[i][j]-(Mu[i][i]*Mu[j][j]);
                Corr[i][j]=Cov[i][j]/sqrt(Cov[i][i]*Cov[j][j]);
                Cov[j][i]=Cov[i][j];
                Corr[j][i]=Corr[i][j];
            }
        }
    }
}

double Energia (unsigned n , const double *x, double *grad,void * fdata)
{
    int i,j,N;
    i=j=N=0;
    N = 118;

    double term,aux;
    term=aux=0.0;

    double **C,**Cov,**Corr,**Theta,**Mu,**Lambda;
    double **Correxp;

    C = malloc(N*sizeof(double*)); for(i=0;i<N;i++) C[i]=malloc(N*sizeof(double));
    Cov = malloc(N*sizeof(double*)); for(i=0;i<N;i++) Cov[i]=malloc(N*sizeof(double));
    Correxp = malloc(N*sizeof(double*)); for(i=0;i<N;i++) Correxp[i]=malloc(N*sizeof(double)
    Corr = malloc(N*sizeof(double*)); for(i=0;i<N;i++) Corr[i]=malloc(N*sizeof(double));
    Theta = malloc(N*sizeof(double*)); for(i=0;i<N;i++) Theta[i]=malloc(N*sizeof(double));
    Mu = malloc(N*sizeof(double*)); for(i=0;i<N;i++) Mu[i]=malloc(N*sizeof(double));
    Lambda = malloc((N+2)*sizeof(double*)); for(i=0; i< (N+2); i++) { Lambda[i] = malloc((N+

    int *Sigma;
    Sigma=malloc(N*sizeof(int));

    //Lectura de la matriz exponenciala
    lectura(Correxp, Cov, &N);
    generasigma(Sigma, &N);

    //Paso a correlacion
    nuevatheta(Theta, Sigma, x,&N);
```

```
    DeThetaaMu(Theta, Mu, Lambda, &N);
    matrizCorr(C, Cov, Corr, Mu, &N);

    //Calculo de la energia en si
    for(i=0;i<N;i++)
    {
        for(j=i+1;j<N;j++)
        {
            term = (Corr[i][j] - Correxp[i][j]);
            aux += term*term;
        }
    }
    //Liberar la memoria utilizada
    free(Sigma);
    for(i=0;i<N;i++)
    {
        free(Correxp[i]);
        free(C[i]);
        free(Cov[i]);
        free(Corr[i]);
        free(Theta[i]);
        free(Mu[i]);
    }
    for(i=0; i< (N+2); i++)
    {
        free(Lambda[i]);

    }
    free(Correxp);
    free(C);
    free(Cov);
    free(Corr);
    free(Theta);
    free(Mu);
    free(Lambda);

    //Devolvemos el valor de la función
    return aux;
}

void Energiav2 (unsigned n , const double *x, double *E)
{
    int i,j,N;
    i=j=N=0;
    N = 118;
```

```
double term,aux;
term=aux=0.0;

double **C,**Cov,**Corr,**Theta,**Mu,**Lambda;
double **Correxp;

C = malloc(N*sizeof(double*)); for(i=0;i<N;i++) C[i]=malloc(N*sizeof(double));
Cov = malloc(N*sizeof(double*)); for(i=0;i<N;i++) Cov[i]=malloc(N*sizeof(double));
Correxp = malloc(N*sizeof(double*)); for(i=0;i<N;i++) Correxp[i]=malloc(N*sizeof(double)
Corr = malloc(N*sizeof(double*)); for(i=0;i<N;i++) Corr[i]=malloc(N*sizeof(double));
Theta = malloc(N*sizeof(double*)); for(i=0;i<N;i++) Theta[i]=malloc(N*sizeof(double));
Mu = malloc(N*sizeof(double*)); for(i=0;i<N;i++) Mu[i]=malloc(N*sizeof(double));
Lambda = malloc((N+2)*sizeof(double*)); for(i=0; i< (N+2); i++) { Lambda[i] = malloc((N+

int *Sigma;
Sigma=malloc(N*sizeof(int));

//Lectura de la matriz exponenciala
lectura(Correxp, Cov, &N);
generasigma(Sigma, &N);

//Paso a correlacion
nuevatheta(Theta, Sigma, x,&N);
DeThetaaMu(Theta, Mu, Lambda, &N);
matrizCorr(C, Cov, Corr, Mu, &N);

//Calculo de la energia en si
for(i=0;i<N;i++)
{
    for(j=i+1;j<N;j++)
    {
        term = (Corr[i][j] - Correxp[i][j]);
        aux += term*term;
    }
}
*E = aux;
guarda(Theta, Mu, Corr, x, &aux);
//Liberar la memoria utilizada
free(Sigma);
for(i=0;i<N;i++)
{
    free(Correxp[i]);
    free(C[i]);
    free(Cov[i]);
```

```
        free(Corr[i]);
        free(Theta[i]);
        free(Mu[i]);
    }
    for(i=0; i< (N+2); i++)
    {
        free(Lambda[i]);

    }
    free(Correxp);
    free(C);
    free(Cov);
    free(Corr);
    free(Theta);
    free(Mu);
    free(Lambda);
}
void guarda ( double **Theta, double **Mu, double **Corr, const double *x, double *E)
{
    FILE *f,*f1,*f2;
    f=fopen("./ThetaPoland.txt","w");
    f1=fopen("./MuPoland.txt","w");
    f2=fopen("./CorrPoland.txt","w");

    int i,j,Nmax;
    i=j=Nmax=0;
    Nmax=118;
    double Emin;
    Emin=0.0;
    Emin=*E;


    fprintf(f,"#eps0=%lf eps1=%lf q=%lf alpha=%lf, E=%lf\n",x[0],x[1],x[2],x[3],Emin);
    fprintf(f1,"#eps0=%lf eps1=%lf q=%lf alpha=%lf, E=%lf\n",x[0],x[1],x[2],x[3],Emin);
    fprintf(f2,"#eps0=%lf eps1=%lf q=%lf alpha=%lf, E=%lf\n",x[0],x[1],x[2],x[3],Emin);
    for(i=0;i<Nmax;i++)
    {
        for(j=i;j<Nmax;j++)
        {
            fprintf(f,"%d %d %.17E\n",i,j,Theta[i][j]);
            fprintf(f,"%d %d %.17E\n",j,i,Theta[i][j]);
            fprintf(f1,"%d %d %.17E\n",i,j,Mu[i][j]);
            fprintf(f1,"%d %d %.17E\n",j,i,Mu[i][j]);
            fprintf(f2,"%d %d %.17E\n",i,j,Corr[i][j]);
            fprintf(f2,"%d %d %.17E\n",j,i,Corr[i][j]);
```

```
        }
    }
    fclose(f);
    fclose(f1);
    fclose(f2);
}
```

5.7. Anexo VII: Código de WSME determinista

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

void lectura (double **Old,double **Cov,int *N);
void generasigma (int *Sigma,int *N);

void nuevatheta (double **Theta, int *Sigma, const double *x,int *N);
void DeThetaaMu (double **Theta,double **Mu, double **Lambda, int *N);

void matrizCorr (double **C,double **Cov, double **Corr,double **Mu, int *N);
double Energia (unsigned n , const double *x, double *grad,void * fdata);
void guardaminima(double **Thetamin,double **Mu,double **Cov,double *x,double *E,int *N);

int main() {

    //Variables enteras
        int i,N;
        i=N=0;
        N = 118;

    //Variables double
        double maxeval, opt_f;
        maxeval=opt_f=0.0;

    void *f_data;

    //Vectores
        double *x;
        x = malloc(4*sizeof(double));

        int *Sigma;
        Sigma = malloc(N*sizeof(int));
```

```
//Matrices
    double **Theta , **Mu , **C, **Cov, **Corr, **Lambda;
    Theta = malloc(N*sizeof(double*)); for (i=0;i<N;i++){ Theta[i] = malloc(N*sizeof(dou
    Mu = malloc(N*sizeof(double*)); for (i=0;i<N;i++){ Mu[i] = malloc(N*sizeof(double));
    C = malloc(N*sizeof(double*)); for(i=0;i<N;i++) C[i]=malloc(N*sizeof(double));
    Cov = malloc(N*sizeof(double*)); for (i=0;i<N;i++){ Cov[i] = malloc(N*sizeof(double)
    Corr = malloc(N*sizeof(double*)); for(i=0;i<N;i++) Corr[i]=malloc(N*sizeof(double));
    Lambda = malloc((N+2)*sizeof(double*)); for(i=0; i< (N+2); i++) { Lambda[i] = malloc

//Creamos el objeto nlopt, el Algoritmo es local derivate Subplex
    nlopt_opt opt = nlopt_create(NLOPT_LN_SBPLX,4);

//Establecemos el objetivo de miminización
    nlopt_set_min_objective(opt,Energia, f_data );//Creo que de esta manera no hara falt

//Lanzar la optimización
    nlopt_optimize(opt, x,&opt_f);

//Destruir el objeto nlopt_opt opt
    nlopt_destroy(opt);

//Observar los resultados por pantalla
    printf("El algortimo ha terminado\n");
    printf("-------------------------\n");
    printf("\t Valores Optimizados:\n");
    printf("\t\t e11 = %E , e01 = %E , e00 = %E , a = %E\n", x[0], x[1], x[2], x[3]);
    printf("\t Energia final:\n");
    printf("\t\t E = %lf\n",opt_f);

//Guardamos las matrices resultantes.
    generasigma(Sigma,&N);
    nuevatheta(Theta,Sigma,x,&N);
    DeThetaaMu(Theta,Mu,Lambda,&N);
    matrizCorr(C,Cov,Corr,Mu,&N);
    guardaminima(Theta,Mu,Cov,x,&opt_f,&N);
//Liberar la memoria de las matrices utilizadas
    //Vectores
        free(x);
        free(Sigma);
    for(i=0;i<N;i++)
    {
        free(Theta[i]);
        free(Mu[i]);
        free(Cov[i]);
    }
```

```
        free(Theta);
        free(Mu);
        free(Cov);
    //C'est fini
    return 0;
}
void lectura (double **Old,double **Cov,int *N)
{
   FILE *f2;
    f2=fopen("./covmatrix.txt","r");
   int i,j,Nmax;
   double aux;
    i=j=Nmax=0;aux=0.0; //Inicializaciones
    Nmax=*N;
    for(i=0;i<Nmax;i++)
    {for(j=0;j<Nmax;j++)
        {
            fscanf(f2,"%lf\t",&aux);
            Cov[i][j]=aux;
        }
        fscanf(f2,"\n");
    }
    fclose(f2);
    //Ahora pasamos a correlaciones
    for(i=0;i<Nmax;i++)
    {for(j=0;j<Nmax;j++)
        {
            Old[i][j]=Cov[i][j]/sqrt(Cov[i][i]*Cov[j][j]);
        }
    }
}

void generasigma (int *Sigma,int *N)
{
    FILE *f;
    f=fopen("./pc_cov.txt", "r");
    int i,Nmax,contpos,contneg;
    i=Nmax=contpos=contneg=0;
    Nmax=*N;
    double aux;
    aux=0.0;
    double *Auxiliar;
    Auxiliar=malloc(Nmax*sizeof(double));
    do{
        fscanf(f,"%d   %lf\n",&i,&aux);
```

```
    Auxiliar[i-1]=aux;
}while(feof(f)==0);
fclose(f);
i=0;
for(i=0;i<Nmax;i++)
{
    if(Auxiliar[i] >= 0.0)
    {
        contpos++;
    }
    else
    {
        contneg++;
    }
}
i=0;
if(contpos > contneg)//Guardamos los positivos como Sigma=1 y los negativos como 0
{
    for(i=0;i<Nmax;i++)
    {
        if(Auxiliar[i] >= 0.0)
        {
            Sigma[i]=1;
        }
        else
        {
            Sigma[i]=0;
        }
    }
}
else//Guardamos los positivos como Sigma=0 y los negativos como 1
{
    for(i=0;i<Nmax;i++)
    {
        if(Auxiliar[i] >= 0.0)
        {
            Sigma[i]=0;
        }
        else
        {
            Sigma[i]=1;
        }
    }
}
free(Auxiliar);
```

```
}

void nuevatheta (double **Theta, int *Sigma, const double *x,int *N)
{
    int i,j,Nmax;
    i=j=Nmax=0;
    Nmax=*N;

    for(i=0;i<Nmax;i++)
    {
        for(j=i;j<Nmax;j++)
        {
            if(i==j)//Elementos diagonales
            {
                if(Sigma[i]==1)//en este caso se le suma e11
                {
                    Theta[i][i] = x[0];
                }
                else//en este caso e00
                {
                    Theta[i][i] = x[2];
                }

            }
            else//Elementos fuera de la diagonal principal
            {
                if(Sigma[i]==1)
                {
                    if(Sigma[j]==1)//en este caso e11
                    {
                        Theta[i][j] = x[0] * pow( j-i , - x[3] );
                    }
                    else//en este caso e01
                    {
                        Theta[i][j] = x[1] * pow( j-i , -x[3] );
                    }
                }
                else//en este caso se le suma epsilonvalue0
                {
                    if(Sigma[j]==1)//en este caso e01
                    {
                        Theta[i][j] = x[1] * pow( j-i , -x[3] );
                    }
                    else//en este caso e00
                    {
```

```
                        Theta[i][j] = x[2] * pow( j-i , -x[3] );
                    }
                }
            }
        }
    }
}
//Programa nuevo
void DeThetaaMu (double **Theta,double **Mu, double **Lambda, int *N)
{
    int i,j,Nmax;
    int h,kh,k;
    double termj,termj_h,termj_kh;
    double termk,termmu;
    i=j=Nmax=0;
    h=kh=k=0;
    termj=termj_h=termj_kh=0.0;
    termk=termmu=0.0;

    Nmax=*N;

    //Matriz auxiliar debido a las condiciones de contorno
    double **Muaux;
    Muaux=malloc((Nmax+2)*sizeof(double*)); for(i=0;i<(Nmax+2);i++) { Muaux[i]=malloc((Nmax+

    //Calculamos los Lambda
        //Debido a la cod. de contorno Lambda[1][0]=1
            Lambda[1][0] = 1.0;
        //Ahora el resto, cuidado que Lambda[j+1][j]
            for(j=1; j < (Nmax+1); j++)
            {
                //Calculo del termj
                    for(h = 1; h < j+1; h++)//Va hasta el h==j, para termj_h
                    {
                        for(kh = h; kh < j+1; kh++)//Va hasta el kh==j, para termj_kh
                        {
                            termj_kh+=Theta[kh-1][j-1];
                        }
                        termj_h += Lambda[h][j-1]*exp(termj_kh);
                        termj_kh = 0.0;
                    }
                    termj = 1.0 / ( 1.0 + termj_h );
                    termj_h = 0.0;

                //Calculo de los terminos Lambda de esa columna
```

```
                for(i=1; i < (j+2); i++)//Solo llega hasta el i==j+1
                {
                    if(i < (j+1))
                    {
                        for(k=i ; k<(j+1); k++)
                        {
                            termk += Theta[k-1][j-1];
                        }
                        Lambda[i][j] = termj*Lambda[i][j-1]*exp(termk);
                        termk=0.0;
                    }
                    else//Caso especial i==j+1
                    {
                        Lambda[j+1][j] = termj;
                    }
                }
          }

//Calculo de Muaux
    //1) Condiciones de contorno
        for(j=0; j < (Nmax+2); j++)
        {
            Muaux[0][j] = 0.0;       //Muaux[0][j]
            Muaux[j][Nmax+1] = 0.0; //Muaux[i][N+1]
        }
        for(i=0; i < (Nmax+1); i++)
        {
            Muaux[i+1][i] = 1.0;// Muaux[i+1][i] (corregida)
        }
    //2) Elementos de la diagonal principal
        for(i=Nmax; i>0; i--)//Desde i=N hasta el i==1
        {
            for(j=i; j<(Nmax+1); j++)
            {
                termmu += Lambda[i+1][j] * ( 1.0 - Muaux[j+1][j+1]);
            }
            Muaux[i][i] = 1.0-termmu;
            termmu = 0.0;
        }
    //3) Resto de elemento Muaux
        for(j=Nmax; j>1; j--)//Dese j==N hasta j==2
        {
            for(i=1; i<j; i++)//Desde 1 hasta el j-1
            {
                Muaux[i][j] = Muaux[i-1][j] + Muaux[i][j+1] - Muaux[i-1][j+1] + Lambda[i
```

```
                }
            }
    //Adaptar los Muaux a la notación de C y guardarlos en Mu
    //Entonces los Mus son los de la parte triangular desde i=1,...,N j=i,...,N y habra que
        for(i=1; i < (Nmax+1); i++)//hasta el Nmax
        {
            for(j=i; j < (Nmax+1); j++)//hasta el Nmax
            {
                Mu[i-1][j-1]=Muaux[i][j];
                Mu[j-1][i-1]=Muaux[i][j];
            }
        }
    //Liberamos la memoria de la matriz Muaux
        for(i=0; i<(Nmax+2); i++)
        {
            free(Muaux[i]);
        }
        free(Muaux);
    //Ya lo tenemos
}
void matrizCorr (double **C,double **Cov, double **Corr,double **Mu, int *N)
{
    //Variables enteras (int)
        int i,j,m,d,Nmax;//Indice de fila, columna,para el bucle del calculo del sumatorio p
    //Variables decimales (double)
        double sum,num,den;//Variable auxialiar para la suma, numerador y denominador
    //Inicialización
        i=j=m=Nmax=0; sum=num=den=0.0;
    //Asignar a Nmax el valor del puntero N, que contiene la dimension de las matrices
        Nmax=*N;

    //Primero calculamos los elementos diagonales
    for(i=0;i<Nmax;i++)
    {
        C[i][i]=Mu[i][i];
        Cov[i][i]=C[i][i]-(Mu[i][i]*Mu[i][i]);
        Corr[i][i]=1.0;
    }

    for(d=1;d<Nmax;d++)//Bucle de las diagonales
    {for(i=0;i<(Nmax-d);i++)//
        {
            j=i+d;//Asigno la columna
            sum=num=den=0.0;
            if(j>(i+1))//Si es j>i+1 le sumamos la expresión delsumatorio
```

```
            {
                for(m=(i+1);m<j;m++)
                {
                    num=(Mu[i][m-1]-Mu[i][m])*(C[j][j]-C[m][j]);
                    den=(1-C[m][m]);
                    sum+=num/den;
                }
                C[i][j]=(Mu[i][j]+sum);
                Cov[i][j]=C[i][j]-(Mu[i][i]*Mu[j][j]);
                Corr[i][j]=Cov[i][j]/sqrt(Cov[i][i]*Cov[j][j]);
                Cov[j][i]=Cov[i][j];
                Corr[j][i]=Corr[i][j];
                sum=num=den=0.0;
            }
            else
            {
                C[i][j]=Mu[i][j];
                Cov[i][j]=C[i][j]-(Mu[i][i]*Mu[j][j]);
                Corr[i][j]=Cov[i][j]/sqrt(Cov[i][i]*Cov[j][j]);
                Cov[j][i]=Cov[i][j];
                Corr[j][i]=Corr[i][j];
            }
        }
    }
}

double Energia (unsigned n , const double *x, double *grad,void * fdata)
{
    int i,j,N;
    i=j=N=0;
    N = 118;

    double term,aux;
    term=aux=0.0;

    double **C,**Cov,**Corr,**Theta,**Mu,**Lambda;
    double **Correxp;

    C = malloc(N*sizeof(double*)); for(i=0;i<N;i++) C[i]=malloc(N*sizeof(double));
    Cov = malloc(N*sizeof(double*)); for(i=0;i<N;i++) Cov[i]=malloc(N*sizeof(double));
    Correxp = malloc(N*sizeof(double*)); for(i=0;i<N;i++) Correxp[i]=malloc(N*sizeof(double)
    Corr = malloc(N*sizeof(double*)); for(i=0;i<N;i++) Corr[i]=malloc(N*sizeof(double));
    Theta = malloc(N*sizeof(double*)); for(i=0;i<N;i++) Theta[i]=malloc(N*sizeof(double));
    Mu = malloc(N*sizeof(double*)); for(i=0;i<N;i++) Mu[i]=malloc(N*sizeof(double));
    Lambda = malloc((N+2)*sizeof(double*)); for(i=0; i< (N+2); i++) { Lambda[i] = malloc((N+
```

```
int *Sigma;
Sigma=malloc(N*sizeof(int));

//Lectura de la matriz exponenciala
lectura(Correxp, Cov, &N);
generasigma(Sigma, &N);

//Paso a correlacion
nuevatheta(Theta, Sigma, x,&N);
DeThetaaMu(Theta, Mu, Lambda, &N);
matrizCorr(C, Cov, Corr, Mu, &N);

//Calculo de la energia en si
for(i=0;i<N;i++)
{
    for(j=i+1;j<N;j++)
    {
        term = (Corr[i][j] - Correxp[i][j]);
        aux += term*term;
    }
}
//Liberar la memoria utilizada
free(Sigma);
for(i=0;i<N;i++)
{
    free(Correxp[i]);
    free(C[i]);
    free(Cov[i]);
    free(Corr[i]);
    free(Theta[i]);
    free(Mu[i]);
}
for(i=0; i< (N+2); i++)
{
    free(Lambda[i]);

}
free(Correxp);
free(C);
free(Cov);
free(Corr);
free(Theta);
free(Mu);
free(Lambda);
```

```
    //Devolvemos el valor de la función
    return aux;
}

void guardaminima(double **Thetamin,double **Mu,double **Cov,double *x,double *E,int *N)
{
    FILE *f,*f1,*f2;
    f=fopen("./ThetaWSME.txt", "w");
    f1=fopen("./MuWSME.txt", "w");
    f2=fopen("./CovWSME.txt", "w");

    int i,j,Nmax;
    i=j=Nmax=0;
    Nmax=*N;
    double Emin;
    Emin=0.0;
    Emin=*E;

    fprintf(f,"#e11=%E , e01=%E , e00=%E, , a=%E , E=%lf\n",x[0],x[1],x[2],x[3],Emin);
    fprintf(f1,"#e11=%E , e01=%E , e00=%E, a=%E , E=%lf\n",x[0],x[1],x[2],x[3],Emin);
    fprintf(f2,"#e11=%E , e01=%E , e00=%E, a=%E , E=%lf\n",x[0],x[1],x[2],x[3],Emin);
    for(i=0;i<Nmax;i++)
    {
        for(j=i;j<Nmax;j++)
        {
            fprintf(f,"%d %d %.17E\n",i,j,Thetamin[i][j]);
            fprintf(f,"%d %d %.17E\n",j,i,Thetamin[i][j]);
            fprintf(f1,"%d %d %.17E\n",i,j,Mu[i][j]);
            fprintf(f1,"%d %d %.17E\n",j,i,Mu[i][j]);
            fprintf(f2,"%d %d %.17E\n",i,j,Cov[i][j]);
            fprintf(f2,"%d %d %.17E\n",j,i,Cov[i][j]);
        }
    }
    fclose(f1);
    fclose(f2);
    fclose(f);
}
```