Título: Análisis vibracional de proteínas en coordenadas internas mediante el modelo ANM

Autora: Alba Rincón Muñoz

Fecha: 17/12/2014

Director: Víctor Alejandro Gil Sepúlveda

Institución del director: Barcelona Supercomputing Center

Ponente: Rosa Maria Badia Sala

Departamento del ponente: UPC – FIB – Departament d'Arquitectura de Computadors

Titulación: Enginyeria Informàtica

Centre: Facultat d'Informàtica de Barcelona (FIB)

Universitat: Universitat Politècnica de Catalunya (UPC) – BarcelonaTech

Índice

1. Introducción....5
1.1. Descripción y objetivos....5
1.2. Los aminoácidos....7
1.3. Las proteínas....8
1.3.1. Funciones de las proteínas....9
1.3.2. Estructura de las proteínas....10
1.3.3. Plegamiento de las proteínas....13
1.4. Dinámica molecular....14
1.5. PELE....16
2. Análisis vibracional....18
2.1. ANM....19
2.1.1. El ANM en PELE....21
2.2. Implementación en coordenadas cartesianas....23
2.2.1. Aplicación del modelo atómico....23
2.2.2. Cálculo de Hessiano....27
2.2.3. Cálculo de eigenvalues y eigenvectors....27
2.2.4. Normalización....28
2.2.5. Cálculo de coordenadas objetivo....29
2.2.6. Minimización....29
2.2.7. Implementación en PELE....31
2.2.8. Minimización de PELE....33
2.3. Implementación en coordenadas internas....35
2.3.1. Aplicación del modelo atómico....38

2.3.2. Cálculo del Hessiano....40
2.3.3. Cálculo de la matriz K....42
2.3.4. Cálculo de eigenvalues y eigenvectors....42
2.3.5. Normalización....43
2.3.6. Cálculo de ángulos objetivo....44
2.3.7. Minimización....45
2.3.8. Implementación en PELE....45
2.3.9. Minimización en PELE....48
2.3.10. Funciones para el cálculo de H....49
2.3.10.1. calculateDij....49
2.3.10.2. calculateTab....50
2.3.10.3. calculateU....50
2.3.10.4. calculateRab....51
2.3.10.5. calculateHab....52
2.3.10.6. calculateH....53
2.3.11. Funciones para el cálculo de K....54
2.3.11.1. calculateC....54
2.3.11.2. calculateI....55
2.3.11.3. calculateKab....55
2.3.11.4. calculateM....56
2.3.11.5. calculateK....57
2.4. Nuevas restricciones en el diedro....58
2.4.1. Ángulo diedro....59
2.4.2. Energía potencial....61
2.4.3. Gradiente....61

2.4.4. Hessiano....62
2.4.5. Implementación....63
2.4.5.1. Funciones de HarmonicDihedralConstraintFunctions....64
2.4.5.1.1. calculateDihedralAngleWithArcTanFunction....64
2.4.5.1.2. calculateEnergyWithArcTanFunction....64
2.4.5.1.3. derivativeXi....65
2.4.5.1.4. calculateGradient....66
2.4.5.1.5. hessianXiYj....66
2.4.5.1.6. calculateHessian....67
2.4.5.2. Funciones de HarmonicDihedralConstraintTerm....68
2.4.5.2.1. getCurrentAngle....68
2.4.5.2.2. getEnergy....68
2.4.5.2.3. getEnergyGradient....69
2.4.5.2.4. getEnergyGradientHess....69
2.4.5.2.5. getHess....70
3. Testing y validación....71
3.1. Testing....71
3.2. Validación....73
4. Rendimiento....86
5. Planificación y costes....87
5.1. Planificación....87
5.2. Costes....89
6. Conclusiones....91
6.1. Objetivos cubiertos....91
6.2. Discusión y mejoras futuras....92

6.3. Valoración personal.....93
7. Bibliografía.....94
I.Anexo: archivos entregados.....97

1. Introducción

1.1. Descripción y objetivos

En el departamento de *Life Science* del centro de investigación Barcelona Supercomputing Center se está desarrollando **PELE** (*Protein Energy Landscape Exploration*), un software cuyo objetivo es explorar de forma eficiente el espacio conformacional de las proteínas y simular interacciones proteína ligando. Este tipo de simulaciones no sólo permiten ganar conocimiento sobre el funcionamiento de las proteínas si no que también son de gran interés para la industria farmacéutica ya que permiten, por ejemplo, abaratar los costes necesarios para descubrir nuevas medicinas.

PELE es un proceso iterativo que parte de un estado inicial de la proteína y consta de varias fases: perturbación local, muestreo de la cadena lateral y minimización. Este proceso deja la molécula en un estado final que se acepta o se rechaza según su energía. Si se acepta, el estado se utiliza como estructura inicial en el siguiente paso, si no, se descarta el movimiento aplicado.

En la primera fase se genera una **perturbación local** en los átomos de la proteína. Para ello, la proteína se representa con un modelo **ANM** (*Anisotropic Network Model*) a partir del cual se pueden determinar las posibles direcciones de movimiento de sus átomos para una conformación dada.

Actualmente en **PELE**, la implementación de este modelo está basada en la manipulación de las coordenadas cartesianas de la proteína. Este proyecto tiene como objetivo la implementación del modelo **ANM** utilizando las coordenadas internas de la proteína, en concreto los ángulos diedros. Las coordenadas internas permiten

acelerar los cálculos y no alteran la posición ni la orientación global del sistema, sólo las distancias relativas entre los átomos de la molécula.

Este hecho proporciona ventajas respecto a la implementación en coordenadas cartesianas en la que se pueden dar problemas como el "*tip effect*" en el cálculo de algunos modos normales. En esos modos, las magnitudes del desplazamiento de los átomos en las partes no estructuradas de la proteína (los extremos) son mucho mayores que las del resto del sistema. Esto provoca que los demás átomos queden prácticamente inmóviles.

La implementación se llevará a cabo siguiendo la tesis doctoral realizada por José Ramón López Blanco. Gracias a su ayuda hemos podido obtener los juegos de pruebas necesarios para asegurar el correcto funcionamiento de nuestro código.

1.2. Los aminoácidos

Los aminoácidos son moléculas orgánicas simples que contienen un grupo amino y un grupo ácido carboxilo.

Los aminoácidos biológicamente importantes son aquellos que tienen el grupo amino y el carboxilo unidos al mismo átomo de carbono central, denominado carbono alfa (**Cα**). Se denominan alfa-aminoácidos (ver Figura 1).

Existen 20 tipos distintos de estos aminoácidos, todos ellos están formados por:

- Un carbono alfa (**Cα**)
- Un grupo amino (**-NH2**)
- Un grupo ácido carboxilo (**-COOH**)
- Un átomo de hidrógeno (**H**)
- Una cadena lateral (**R**)

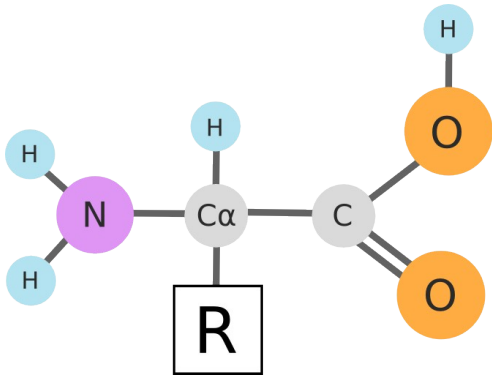


Figura 1 *Representación de un aminoácido*

La función y el comportamiento de cada aminoácido vienen determinados por su cadena lateral o *side-chain* (**R**).

Varios aminoácidos pueden unirse mediante un enlace (denominado enlace peptídico) formando una cadena (péptidos o polipéptidos). Si esta cadena polipeptídica está formada por más de 100 aminoácidos [1] y tiene una estructura tridimensional se considera ya una proteína (ver Figura 2).

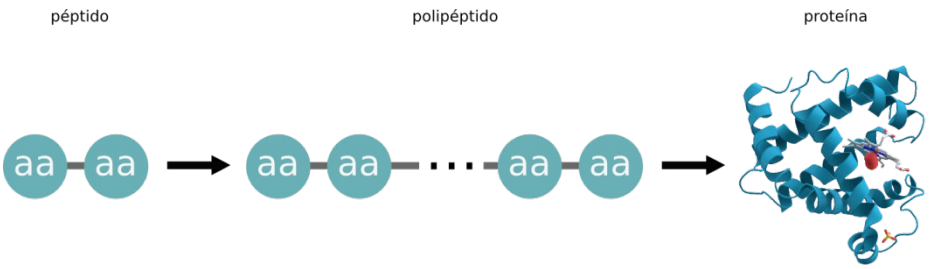


Figura 2 *Una proteína está formada por una cadena polipeptídica compuesta a su vez por aminoácidos enlazados.*

1.3. Las proteínas

Las proteínas participan en gran cantidad de procesos biológicos. Llevan a cabo funciones vitales como el movimiento muscular, la digestión o la defensa contra infecciones. Su mal funcionamiento puede originar enfermedades, por ejemplo enfermedades neurodegenerativas como el *Parkinson* o el *Alzheimer*.

Cada proteína está formada por una secuencia única de aminoácidos. La cantidad, el orden y el tipo de aminoácidos que la componen determinan su estructura y su función.

Una vez unidos en la cadena que conforma la proteína, los átomos de carbono (**C**), nitrógeno (**N**) y oxígeno (**O**) enlazados forman la cadena principal o *backbone* de la proteína. Las cadenas laterales (**R**) de los aminoácidos se enlazan al *backbone* y se denominan residuos (ver Figura 3).

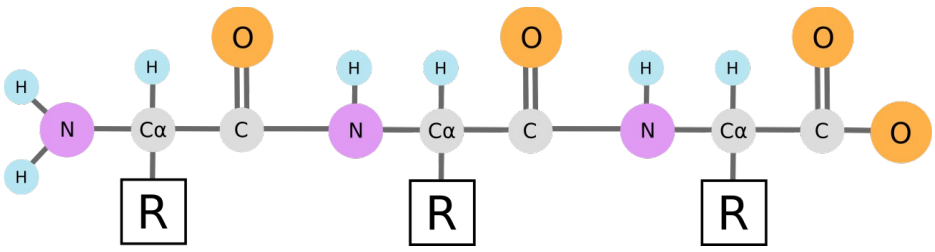


Figura 3 *Esquema de la estructura básica de una proteína*

1.3.1. Funciones de las proteínas

Las proteínas desempeñan una variedad de funciones esenciales que dependen de su estructura. Las principales son:

- Estructural:

  Forman tejidos capaces de soportar gran tensión como huesos, cartílagos o tendones.

- Movimiento y contracción:

  Constituyen estructuras que producen movimientos. Por ejemplo, la contracción muscular es resultado de la interacción entre dos proteínas: la actina y la miosina.

- Transporte:

  Tienen la capacidad de transportar sustancias como oxígeno, lípidos o electrones. Proteínas como la hemoglobina y la mioglobina transportan el oxígeno en la sangre en los organismos vertebrados.

- Reserva energética:

  Pueden acumular y producir energía y aportarla al organismo. Por ejemplo, la ovoalbúmina, principal proteína de la clara de huevo, funciona como una reserva de aminoácidos para el desarrollo del ave.

- Homeostática:

  Regulan y mantienen el equilibrio en diferentes medios.

- Defensiva:

  Las proteínas crean los anticuerpos que ayudan al sistema inmunitario a proteger el organismo de agentes externos.

- Hormonal:

  Funcionan como mensajeros de señales hormonales y facilitan la comunicación entre células.

- Enzimática:

  Actúan como biocatalizadoras para acelerar las reacciones químicas del metabolismo.

1.3.2. Estructura de las proteínas

Las proteínas se pliegan formando estructuras tridimensionales. Existen cuatro niveles estructurales [21]:

- Estructura primaria:

  Corresponde con la secuencia de aminoácidos que conforman la cadena de la proteína y el orden lineal en que se encuentran.

  El primer y el último aminoácido de la cadena se denominan n-terminal y c-terminal respectivamente.

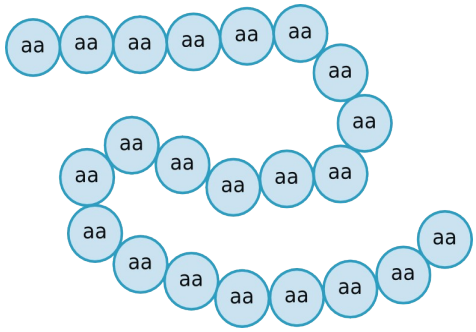


Figura 4 *Estructura primaria*

– Estructura secundaria:

Es la manera en que se pliega sobre sí misma la proteína formando una estructura tridimensional. Esta estructura viene determinada por los enlaces entre los átomos de hidrógeno.

Las estructuras más comunes son las denominadas hélice alfa y hoja beta.

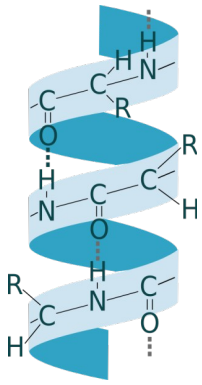


Figura 5 *Estructura secundaria*

– Estructura terciaria:

Corresponde con la distribución tridimensional en el espacio de los átomos que componen la proteína. Esta configuración está determinada por las interacciones entre las cadenas laterales de los aminoácidos.

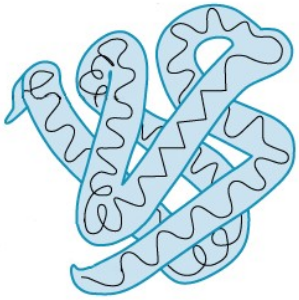


Figura 6*: Estructura terciaria*

– Estructura cuaternaria:

Describe la forma en que se enlazan e interaccionan las múltiples cadenas polipeptídicas de la proteína.

Sólo las proteínas que están formadas por más de una cadena tienen estructura cuaternaria.

Figura 7 *Estructura cuaternaria*

1.3.3. Plegamiento de las proteínas

Las proteínas están formadas por cadenas lineales de aminoácidos y sólo cuando estas cadenas se pliegan formando una estructura tridimensional (estructura terciaria) son capaces de desarrollar su función biológica (ver Figura 8).

La secuencia de aminoácidos que las componen es la que determina la forma que tendrá la proteína cuando se pliegue. Sin embargo, predecir esta estructura terciaria es un problema muy complejo, sólo se puede determinar mediante complicados análisis experimentales [4][5].

El incorrecto plegamiento de las proteínas provoca que no sean funcionales, es decir, que no sean capaces de cumplir correctamente con su función biológica. Este hecho puede derivar en enfermedades como son el *Alzheimer*, la encefalopatía espongiforme, la enfermedad de *Huntington* o algunas formas de cáncer [3].

La predicción de estas estructuras es clave para entender su funcionamiento y poder corregirlo en caso de ser incorrecto. Actualmente es uno de los principales y más importantes objetos de estudio en la bioquímica y la bioinformática.

Existen técnicas de simulación, como la dinámica molecular [2], que permiten calcular y visualizar el movimiento de las proteínas. El principal problema de estas técnicas es la gran cantidad de recursos que necesitan para conseguir simulaciones de tan solo unos pocos nanosegundos, ya que la velocidad a la que se pueden calcular depende del tamaño del sistema (del número de átomos de la cadena).

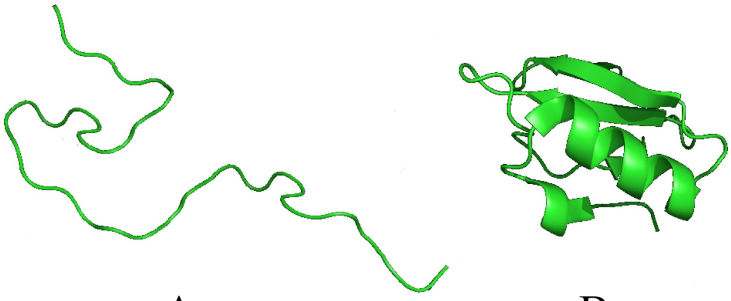


***Figura 8**: En la ilustración se puede apreciar una proteína antes (A) y después (B) del plegamiento*

1.4. Dinámica molecular

La dinámica molecular es un método de simulación por ordenador capaz de predecir el movimiento de átomos y moléculas. Esta técnica describe el comportamiento de un sistema de partículas (la molécula) a lo largo del tiempo mediante la integración de las ecuaciones del movimiento de Newton [17][18][26]. El algoritmo llevado a cabo en la simulación de dinámica molecular puede resumirse en los siguientes pasos [7]:

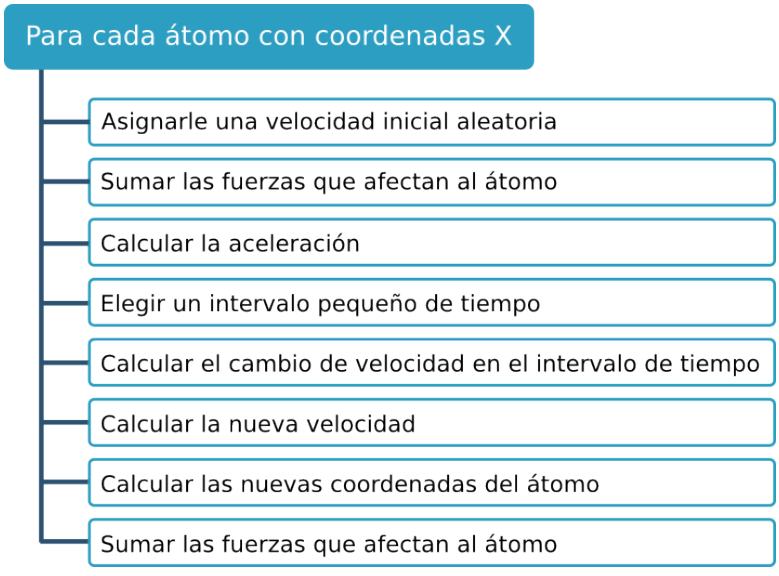


***Figura 9** Pasos del algoritmo de simulación de la dinámica molecular*

Para llevar a cabo esta simulación es necesario también tener en cuenta las interacciones existentes entre las partículas.

Los *force fields* (campos de fuerza) [8][27] son modelos empíricos que se utilizan para describir la energía potencial de un sistema de partículas, en este caso de átomos (ecuación 1). Estas ecuaciones tienen en cuenta la energía debida a los enlaces covalentes (*bonded*) y los no covalentes (*non-bonded*) [6][10]:

$$V(R) = E_{bonded} + E_{non-bonded} \quad (1)$$

- Fuerzas intramoleculares (bonded):

$$E_{bonded} = E_{bond-stretch} + E_{angle-bend} + E_{rotate-along-bond} \quad (2)$$

  - Fuerzas para describir los enlaces entre átomos (*bond-stretch*).
  - Fuerzas que mantienen el ángulo entre dos enlaces consecutivos (*angle-bend*).
  - Fuerzas que mantienen el ángulo diedro que forman cuatro átomos unidos por tres enlaces (*rotate-along-bond*).

- Fuerzas intermoleculares (non-bonded):

$$E_{non-bonded} = E_{van-der-Waals} + E_{electrostatic} \quad (3)$$

  - Fuerzas de Van der Waals: fuerzas de estabilización que mantienen la estructura de la proteína evitando colisiones estéricas [9].
  - Fuerzas electrostáticas: fuerzas de atracción y repulsión que dependen del signo de las cargas de los átomos.

El inconveniente de la dinámica molecular es que, al tener que usar pasos de simulación muy cortos para evitar los errores de integración, necesita una gran cantidad de recursos computacionales para conseguir un intervalo de tiempo muy pequeño de simulación.

1.5. PELE

Con el objetivo de minimizar el coste computacional de la dinámica molecular, se han creado técnicas más eficientes para explorar el espacio conformacional de las proteínas.

Este es el caso de **PELE** (*Protein Energy Landscape Exploration*) [16], un software que combina algoritmos de predicción de la estructura de las proteínas y el método de Monte Carlo para conseguir un nuevo sistema de exploración mucho más eficiente (ver Figura 10).

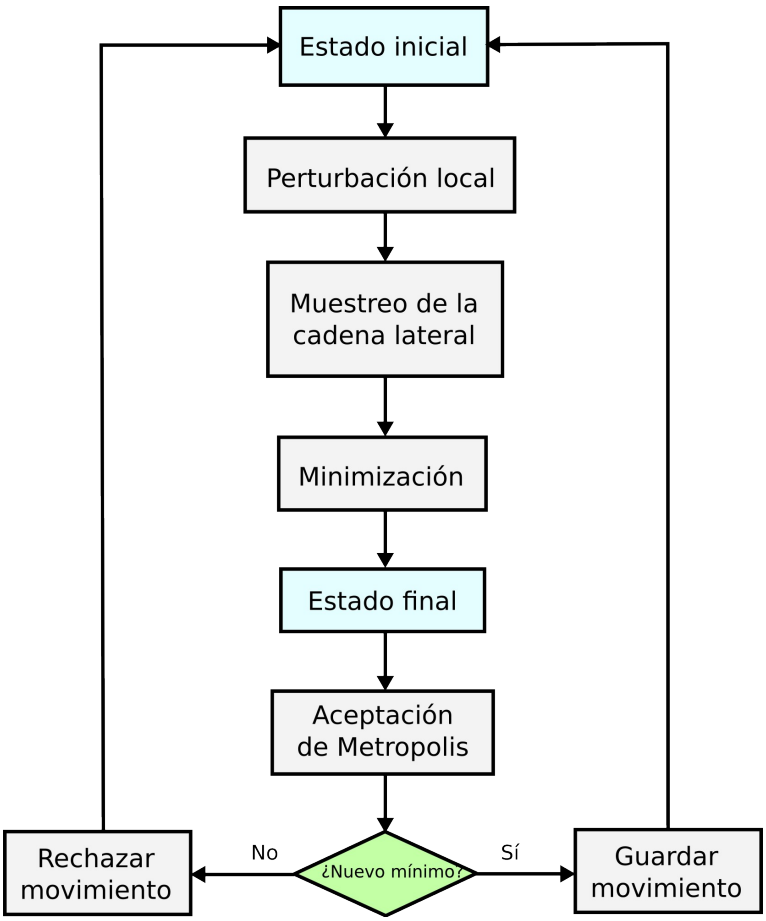


Figura 10 *Diagrama de flujo de PELE*

El algoritmo parte de un **estado inicial** de la proteína (una conformación y una energía inicial) que es objeto de estudio.

El primer paso es la generación de una **perturbación local** (es decir, un movimiento) al ligando y a la proteína. La nueva conformación de la proteína se obtiene mediante el cálculo de modos normales de vibración usando **ANM**.

A continuación, se realiza un **muestreo de las cadenas laterales**. El algoritmo continúa reposicionando de manera óptima todas las cadenas laterales de los átomos afectados por la perturbación del paso anterior con el objetivo de evitar contactos entre ellos y conseguir la menor energía posible.

La siguiente es la fase de **minimización**. Consiste en generar una respuesta del *backbone* a las restricciones introducidas por el movimiento causado por la perturbación local. Se consigue minimizar la energía total de la proteína aplicando el algoritmo de truncado de Newton.

Estos tres pasos principales constituyen un movimiento que deja la proteína en un **estado final** que es aceptado o rechazado según un **criterio de aceptación de Metropolis** que determina si se ha alcanzado un nuevo mínimo de energía. En caso de que así sea se guarda el movimiento y se usa como estructura inicial del siguiente paso. Si no lo es, se rechaza y se vuelve a iniciar nuevamente el algoritmo con la estructura inicial de la proteína.

2. Análisis vibracional

Para la generación de la perturbación local, PELE realiza un análisis vibracional de la proteína. En general, el análisis vibracional de un cuerpo permite determinar su movimiento y su evolución temporal en el espacio. Aplicado a las proteínas, permite conocer las direcciones de movimiento de los átomos que las conforman.

Estas posibles direcciones de movimiento se conocen como modos normales de vibración (ver Figura 11) y son un conjunto de vectores independientes que representan todos los patrones o formas posibles en que oscilará la estructura de la proteína alrededor de su conformación de equilibrio (mínimo local de energía) al ser perturbada. La deformación de una proteína se puede expresar como una combinación lineal de varios de estos modos calculados.

Cada uno de los modos se representa mediante un autovector (*eigenvector*) y un autovalor (*eigenvalue*). El *eigenvector* contiene la información sobre la amplitud y la dirección de movimiento de cada átomo y el *eigenvalue* contiene su frecuencia. De esta manera, se pueden determinar los movimientos más probables.

En las proteínas, los modos con frecuencias más altas representan movimientos muy localizados en los que intervienen un número bajo de átomos (como por ejemplo en la vibración de los átomos de hidrógeno). Por otra parte, los modos con frecuencias más bajas corresponden a movimientos deslocalizados o colectivos en los que participan una gran cantidad de átomos que oscilan con una amplitud considerable. Precisamente son estos modos los más relevantes ya que están relacionados con las propiedades funcionales de las proteínas.

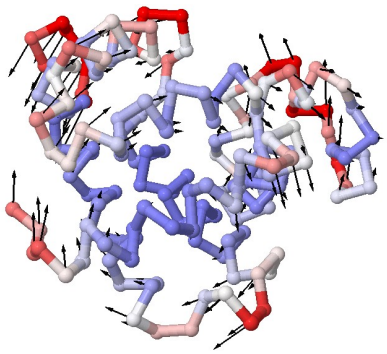


Figura 11 *En la proteína de la figura las flechas representan las direcciones de movimiento de cada uno de los átomos obtenidas mediante el ANM.*

2.1. ANM

En PELE, para el análisis de los modos normales de vibración (*Normal Mode Analisis*, NMA [12][20]) se utiliza el ANM (*Anisotropic Network Model*) [11]. En este método se representa la proteína como una red elástica formada por nodos conectados entre sí a través de muelles. Cada uno de los nodos de la red está centrado en el átomo de Cα del residuo de la proteína correspondiente al que representa. Por su parte, los resortes simulan las fuerzas que interactúan entre los nodos (ver Figura 12).

Los muelles de la red representan dos tipos distintos de fuerzas:

- Debidas a enlaces covalentes: la fuerza existente entre cada par de nodos unidos directamente mediante un enlace covalente.
- Debidas a enlaces no covalentes: la fuerza que actúa entre cada par de nodos que se encuentren dentro de una rango de distancia denominada distancia de *cut-off*.

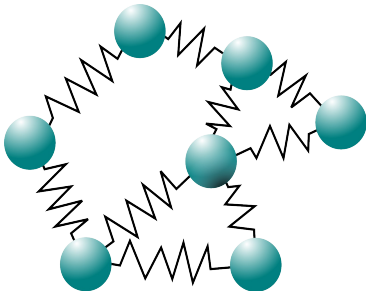


Figura 12 *La proteína se representa como una red formada por nodos y muelles*

El movimiento oscilatorio del sistema puede ser descrito mediante los modos normales de vibración. Para encontrar todas esas posibles direcciones de movimiento de los átomos de la molécula es necesario resolver el siguiente problema de valores propios:

$$HV = \Lambda KV \tag{4}$$

donde **H** es la matriz Hessiana, **K** es la matriz de energía cinética, **V** es el conjunto de vectores propios (*eigenvectors*) que representan los N modos normales asociados al sistema y **Λ** es la matriz diagonal de valores propios (*eigenvalues*).

2.1.1. El ANM en PELE

En la fase de la generación de la perturbación local es donde PELE utiliza los modos normales de vibración calculados con el ANM para reposicionar los átomos del *backbone*.

Actualmente PELE lleva a cabo estos cálculos utilizando un sistema de referencia en coordenadas cartesianas, es decir, basado en la posición concreta de cada átomo de la proteína en el espacio.

Un modo normal de vibración se representa con un vector para cada uno de los **N** átomos del sistema que indica cómo debe moverse. Estos vectores sólo indican la dirección del movimiento, no el sentido ni la magnitud del desplazamiento (ver Figura 13).

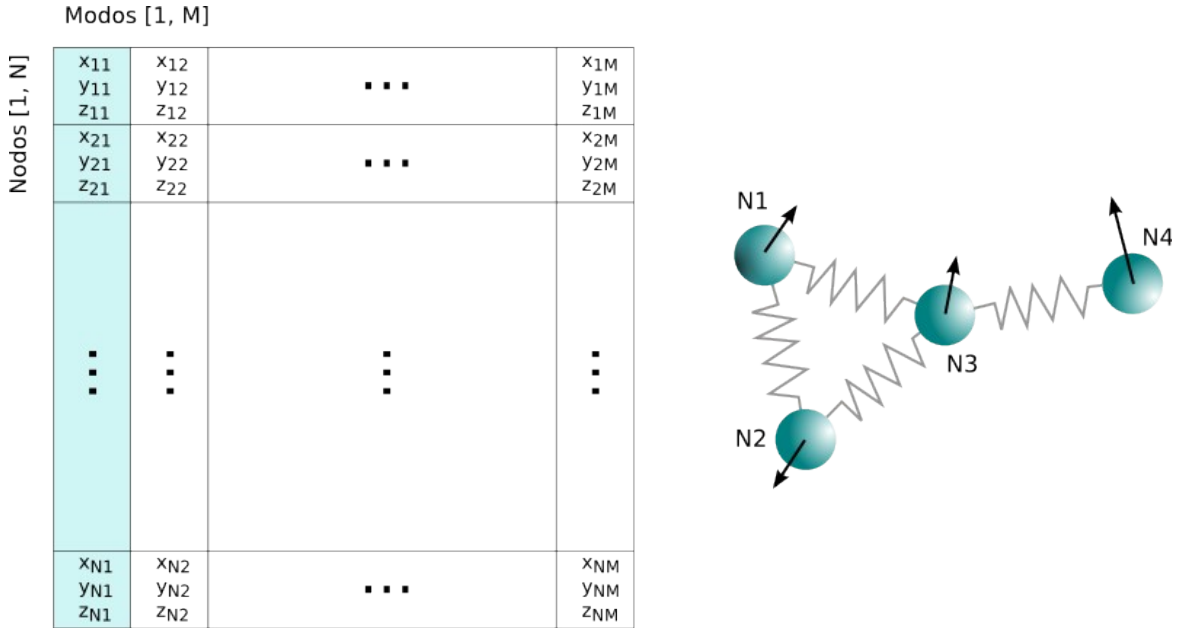


Figura 13 *A la izquierda se puede ver la representación de un conjunto de M modos normales. Cada modo se compone de N vectores, correspondientes a cada uno de los nodos de la red. A la derecha, se muestra una red de cuatro nodos donde las flechas representan los vectores del modo normal elegido.*

Para conseguir calcular los *eigenvectors* y los *eigenvalues* se debe calcular la matriz de energía cinética K y la matriz hessiana H.

Los elementos de K y H se obtienen mediante el cálculo de las segundas derivadas parciales de la energía cinética y de la energía potencial respectivamente:

$$K_{\alpha\beta} = \frac{\partial K}{\partial \dot{q}_\alpha \partial \dot{q}_\beta} \tag{5}$$

$$H_{\alpha\beta} = \frac{\partial V}{\partial q_\alpha \partial q_\beta} \tag{6}$$

En coordenadas cartesianas es posible reducir los cálculos de H y K usando coordenadas pesadas por las masas. De esta forma, ya no es necesario tener en cuenta la matriz de energía cinética K.

La ecuación a solucionar queda:

$$\hat{H}\hat{V} = \hat{\Lambda}\hat{V} \tag{7}$$

donde $\hat{}$ significa que las matrices tienen en cuenta las masas de los átomos.

2.2. Implementación en coordenadas cartesianas

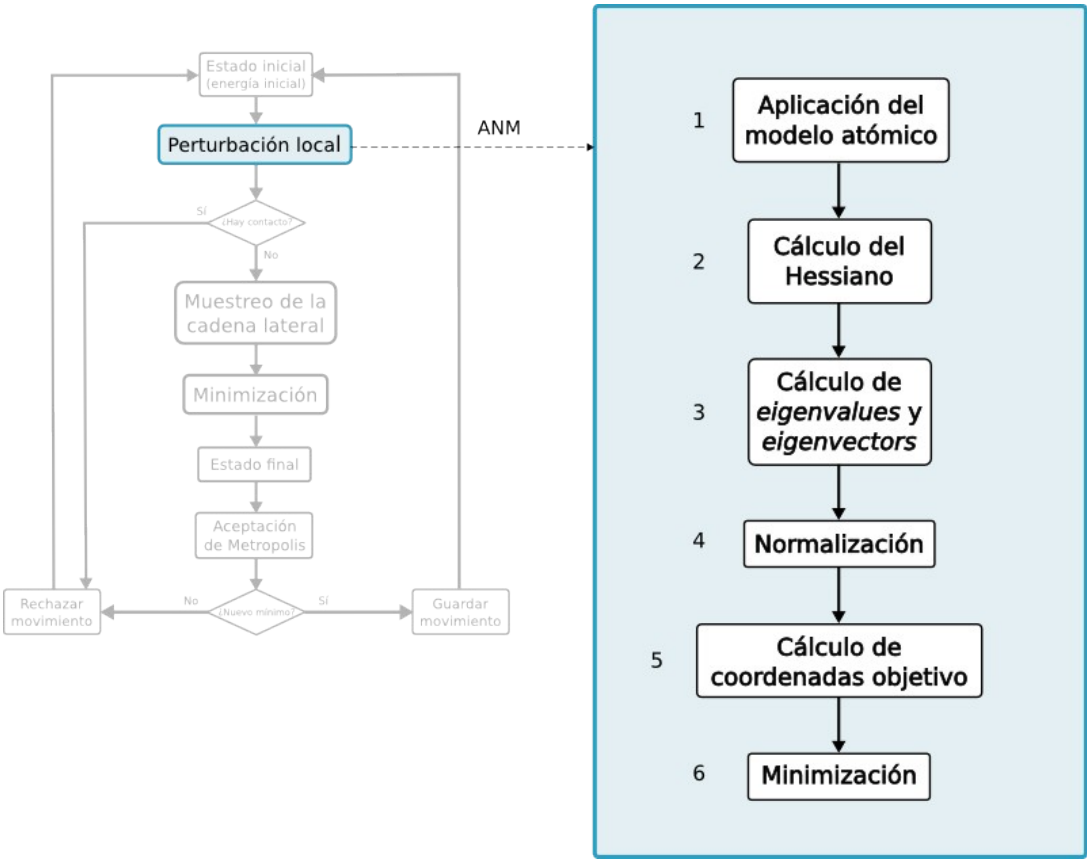


Figura 14 *Esquema de funcionamiento de la perturbación local en coordenadas cartesianas*

2.2.1. Aplicación del modelo atómico

El coste computacional de las simulaciones con proteínas es muy elevado y depende del nivel de resolución o detalle de la estructura de la molécula analizada, es decir, de la cantidad de átomos que se utilicen como entrada.

Es posible realizar simulaciones representando las proteínas con todos sus átomos con la ayuda de supercomputadores, pero están limitadas por su elevado coste a tiempos del orden de nanosegundos [28].

Gracias a la utilización de modelos atómicos reduccionistas o *coarse-grained* es posible disminuir en gran medida los recursos necesarios para calcular los modos normales [22]. Estos modelos permiten representar la estructura de las proteínas manteniendo su forma original pero reduciendo considerablemente el número de átomos usados como entrada.

En el modelo utilizado en el caso de PELE cada aminoácido de la cadena se representa mediante un único nodo cuyo centro corresponde con la posición del átomo de carbono alfa (ver Figuras 15 y 16). Como los átomos de Cα forman parte del *backbone* de la cadena, es posible reconstruir toda estructura inicial de la proteína a partir de su modelo *coarse-grained*.

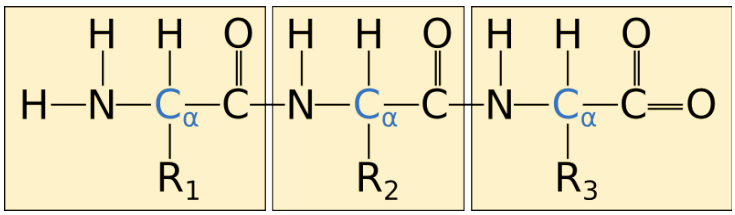


Figura 15 *Modelo inicial – En el esquema se puede apreciar una cadena compuesta de tres aminoácidos enlazados con los átomos de Cα destacados*

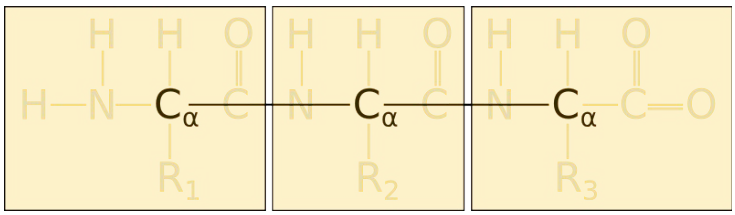


Figura 16 *Modelo coarse-grained Cα – Después de aplicar el modelo atómico Cα, cada uno de los aminoácidos se representa mediante un sólo átomo de carbono centrado en el Cα*

Como se ha explicado anteriormente, en el ANM la proteína se representa mediante nodos conectados a través de muelles elásticos que expresan las fuerzas existentes

entre los átomos (ver Figura 17 y 18). Después del proceso de aplicación del modelo atómico y del ANM la cadena quedaría de la siguiente forma:

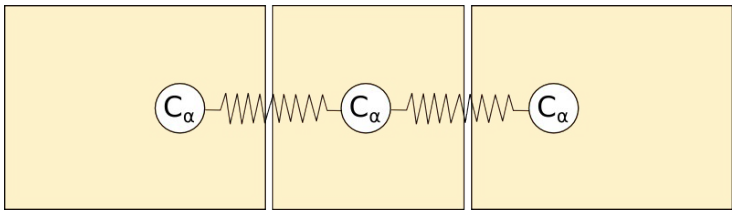


Figura 17 *Modelo de nodos y muelles – En la figura sólo se representan las interacciones debidas a los enlaces covalentes*

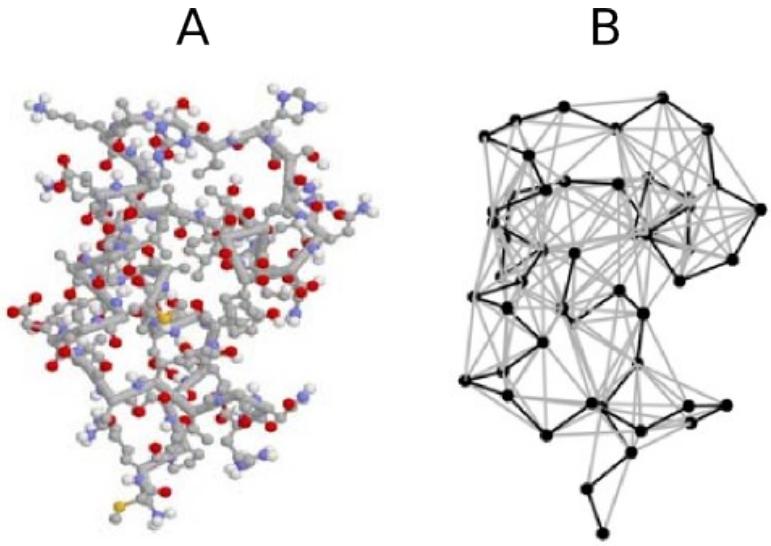


Figura 18 *En la ilustración A se puede observar la representación de la proteína con todos sus átomos, mientras que en B se representan sólo los átomos Cα y sus interacciones. [29]*

En PELE, ***NodeListGeneratorBuilder::createFromConfiguration*** es la función encargada de crear una instancia de ***NodeListGenerator*** e inicializar sus atributos de tipo ***Selector*** a partir de la configuración indicada en un archivo en formato *JSON* (*JavaScript Object Notation*) mediante las funciones ***createNodeSelector*** y ***createLinksSelector***.

Finalmente, para generar la lista de nodos se llama a la función ***NodeListGenerator::generateNodeList*** que selecciona los enlaces (***computeLinksList*** y ***deleteOmitingLinksInVector***) y los nodos a incluir en el modelo (***selectNodes***).

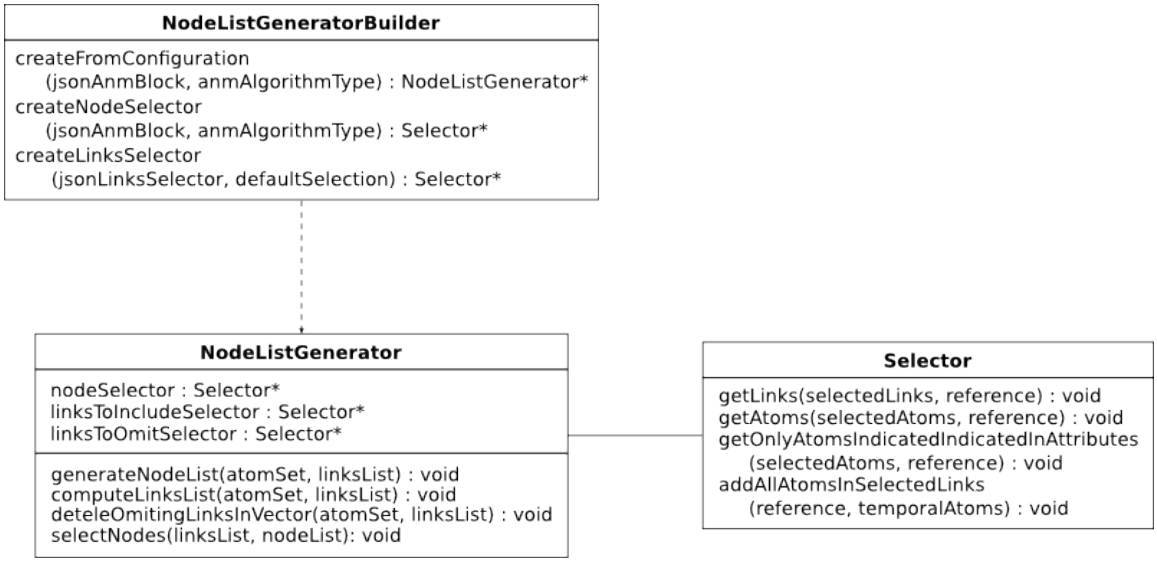


Figura 19 *Estructura de clases de PELE donde se construye el modelo atómico*

2.2.2. Cálculo de Hessiano

Una vez decidido el modelo *coarse-grained* se puede calcular la matriz hessiana para poder resolver el problema planteado.

Cada elemento del Hessiano se calcula como:

$$H_{xy} = \frac{f_{xy}}{|r_{xy}|^2} r_x r_y \qquad si\ x \neq y$$
$$H_{xy} = \sum_{z,z\neq x}^{3n} \frac{f_{xz}}{|r_{xz}|^2} r_x r_z \qquad si\ x = y \tag{8}$$

donde r_{xy} es el vector desplazamiento entre las posiciones de equilibrio de los átomos x e y, f_{xy} es la constante de fuerza del muelle entre los átomos x e y, y r_x y r_y son las coordenadas del punto r_{xy}.

2.2.3. Cálculo de *eigenvalues* y *eigenvectors*

Una vez calculado el Hessiano se puede resolver la ecuación:

$$\hat{H}\,\hat{V} = \hat{\Lambda}\,\hat{V} \tag{9}$$

donde $\hat{H}$ es la matriz de energía potencial, $\hat{\Lambda}$ es la matriz diagonal de *eigenvalues* y $\hat{V}$ es el conjunto de *eigenvectors*. Por lo tanto, $\hat{V}$ es una matriz que contiene los N eigenvectors que representan los N modos normales asociados cada nodo del sistema (ver Figura 20).

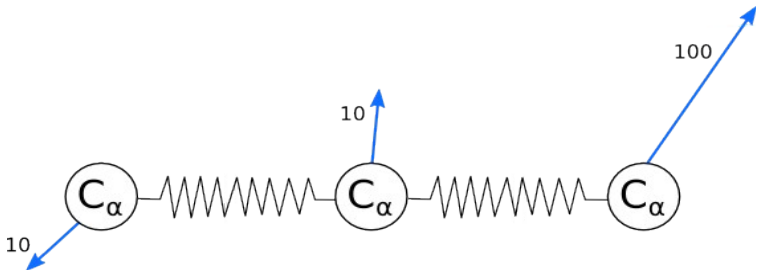


Figura 20 *A cada nodo se le aplica uno de los modos normales. El valor indicado en los vectores representa la magnitud del eigenvector para el nodo correspondiente*

2.2.4. Normalización

Una vez obtenidos los *eigenvalues* y los *eigenvectors*, es necesario normalizarlos: la magnitud total de los vectores calculados no es relevante en la metodología del ANM, sólo es importante su dirección y magnitud relativa.

La normalización se lleva a cabo partiendo del vector con mayor magnitud de entre los calculados. Este vector pasa a tener módulo 1 y todos los demás se dividen entre su norma (ver Figura 21).

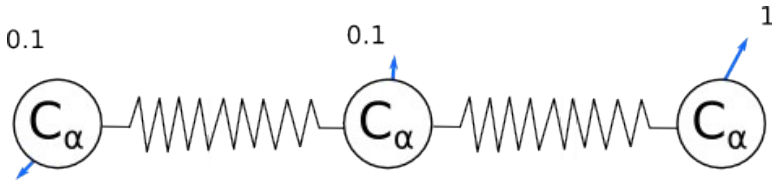


Figura 21 *El valor indicado en cada vector representa la magnitud del eigenvector después de normalizarlo.*

En este proceso pueden aparecer problemas derivados del ANM como el "*tip effect*": esta situación se produce en moléculas que tiene partes no estructuradas, como pueden ser los extremos de la proteína (*tips*). Debido a la existencia de estas zonas, algunos modos normales pueden estar sobredimensionados. En estos modos, la magnitud de los vectores de movimiento en las zonas menos estructuradas es mucho más grande que en el resto de sistema. Al normalizar los vectores a partir del vector de magnitud mayor se obtienen unos desplazamientos muy elevados en los extremos y otros muy pequeños en los demás átomos, lo que provoca que el resto de la proteína se mantenga prácticamente estática.

Para evitar este hecho, se puede llevar a cabo el cálculo del ANM sin incluir las zonas no estructuradas de la molécula o bien utilizando desplazamientos relativos, es decir, movimientos o giros que conserven la estructura original de la proteína.

2.2.5. Cálculo de coordenadas objetivo

A continuación, se determina la posición final para cada uno de los átomos del sistema (ver Figura 22).

El desplazamiento se calcula como una combinación lineal de uno o más modos normales. Como los *eigenvectors* están normalizados es necesario también decidir una magnitud que se usará para calcular el desplazamiento máximo de los átomos. En PELE esta magnitud es un parámetro que el usuario puede elegir y modificar.

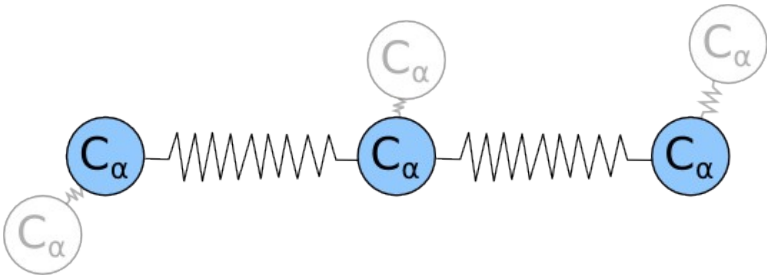


Figura 22 *En la ilustración se pueden apreciar en azul los nodos en su posición inicial y en gris la nueva posición objetivo calculada*

Al fijar la posición final de cada uno de los átomos después del desplazamiento, se añaden nuevas restricciones de movimiento a la estructura de la proteína que se suman a las que ya tenía originalmente.

2.2.6. Minimización

Al haber añadido nuevas restricciones en el paso anterior es necesario que la minimización intente situar cada nodo en su posición final calculada (ver Figura 23). Su objetivo es conseguir una conformación tal que su energía potencial sea mínima y se cumplan todas las restricciones impuestas por la perturbación local a la vez que se cumplen también las restricciones iniciales de la proteína (de distancia entre átomos,

de ángulo y de ángulo diedro). De este modo los nodos se moverán a una posición cercana a la calculada en el paso anterior.

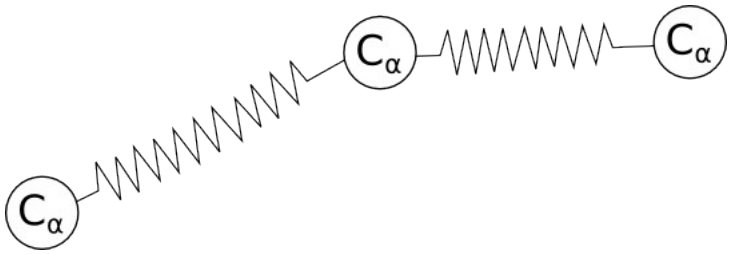


Figura 23 *Posición final de los nodos del modelo después del cálculo de los puntos finales y la minimización*

2.2.7. Implementación en PELE

A continuación se muestra el diagrama de funcionamiento de la implementación del ANM en coordenadas cartesianas.

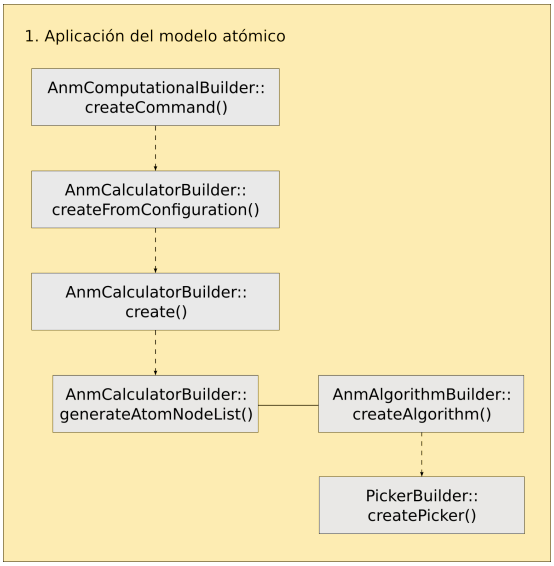


Figura 24 *Diagrama de funciones que se encargan de la aplicación del modelo atómico en PELE*

Los parámetros de configuración de una ejecución de PELE (como son el número de modos o la distancia de *cut-off*) se obtienen mediante un archivo en formato *JSON*. La clase ***AnmCalculatorBuilder*** se encarga de crear un objeto de la clase ***AnmCalculator*** que cumpla con todos los requisitos especificados en dicho archivo de configuración. Es en este proceso donde se produce la aplicación del modelo atómico (paso 1) a través de la función ***AnmCalculatorBuilder:: generateAtomNodeList()*** que construye la lista final de nodos partiendo del conjunto de átomos de la proteína.

A partir de aquí se empieza la ejecución de un paso de PELE. La perturbación local se inicia con la llamada a la función ***PeleStep::perturb()***. Primero es necesario

calcular los modos normales. Se calcula el Hessiano (paso 2) mediante la función ***LapackFormatHessianGenerator::computeUpperMatrix()***, luego los *eigenvalues* y *eigenvectors* (paso 3) con ***LapackCartesianModesCalculator::calculateEigenVectorsAndEigenValues()*** y finalmente se normalizan (paso 4) a través de ***LapackCartesianModesCalculator::normalizeEigenVectors()***.

Por último, se obtienen las coordenadas objetivo (paso 5) de cada nodo con la función ***AnmAlgorithm::calculateTargetCoords()*** que se encarga de calcular los vectores de desplazamiento y actualizar las coordenadas y se lleva a cabo la minimización (paso 6) en ***AnmCalculator::minimize()***.

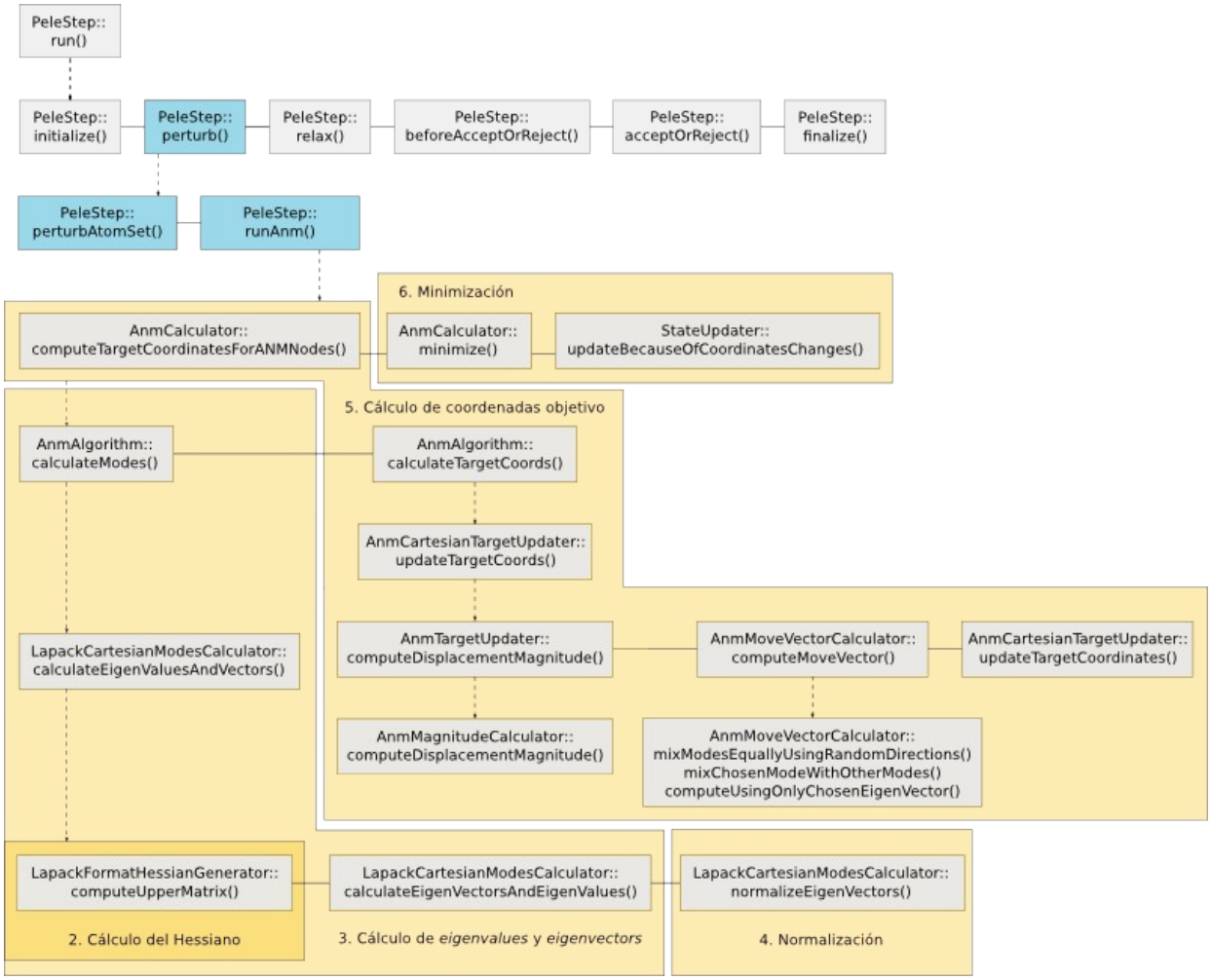


Figura 25 *Diagrama donde se muestran las clases y funciones que utiliza PELE para cada uno de los pasos del algoritmo en la etapa de la perturbación local*

2.2.8. Minimización de PELE

La última etapa de PELE es la minimización (ver Figura 26) y su objetivo es disminuir la energía de la proteína después de las perturbaciones sufridas en las etapas anteriores.

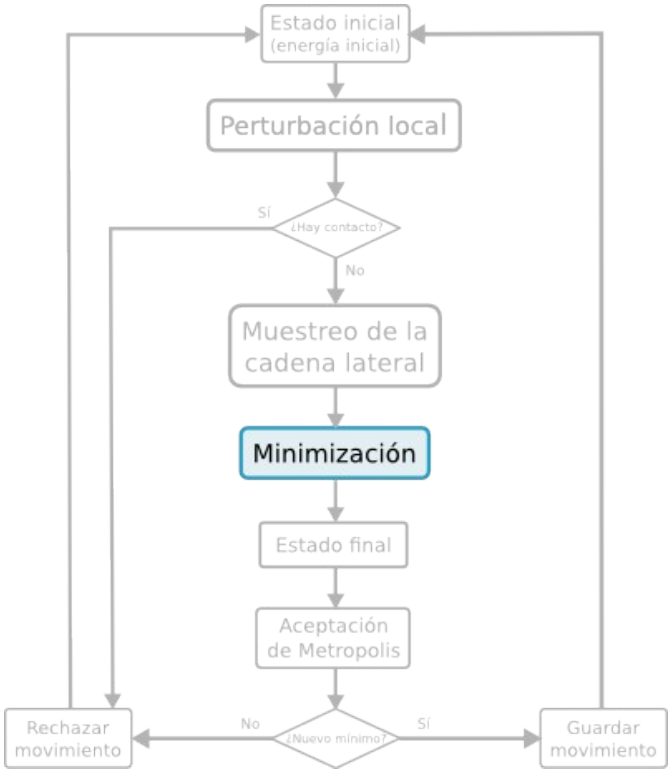


Figura 26 *Esquema de PELE donde se observa la etapa de minimización*

La tendencia del minimizador será devolver al sistema a su estado inicial previo al desplazamiento producido por la perturbación local. Para evitarlo, se deben añadir nuevas restricciones en los nodos de Cα (en forma de muelles con constantes elásticas altas) que los fijarán en su posición actual.

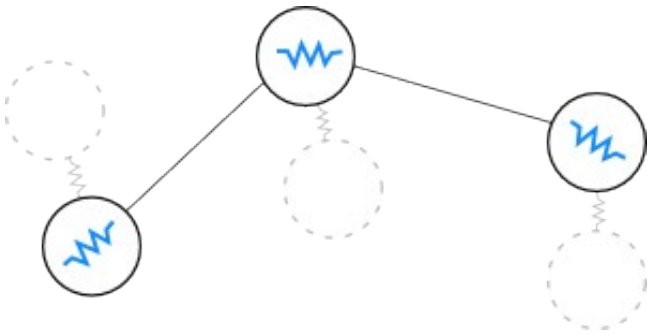


Figura 27 *En gris se observan las posiciones iniciales de los nodos, para evitar que vuelvan a ellas se añaden restricciones (muelles, en azul) en las posiciones actuales*

El minimizador reducirá la energía de la proteína respetando todas las restricciones en la medida de lo posible y sin deshacer el movimiento provocado por el ANM.

2.3. Implementación en coordenadas internas

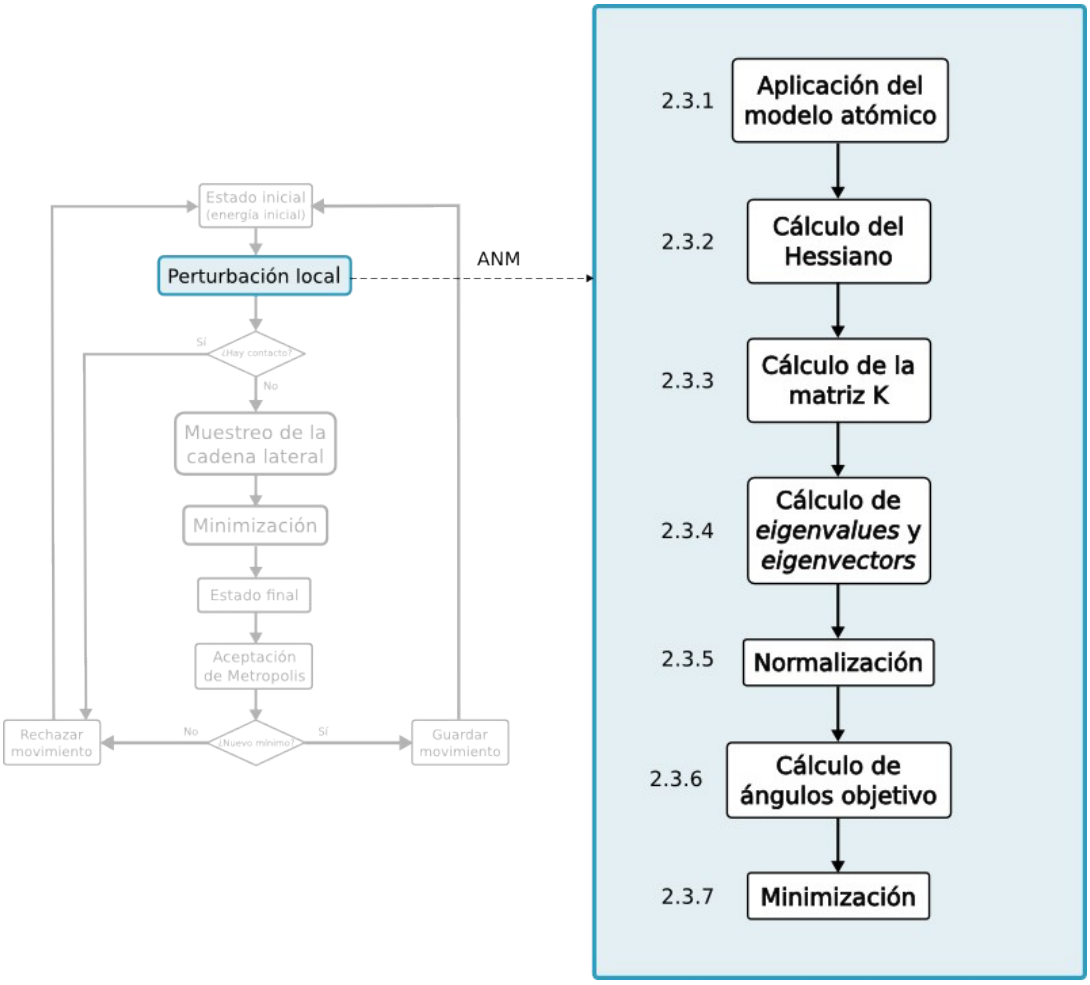


Figura 28 *Esquema de funcionamiento de la perturbación local en coordenadas cartesianas*

El diseño anterior de PELE no preveía la introducción del módulo ANM en coordenadas internas. Debido a esto muchas clases y funciones, partían de la precondición que los *eigenvectors* de entrada provenían de la diagonalización del Hessiano en coordenadas cartesianas y, por tanto, representaban desplazamientos y tenían dimensión 3N. Además, el modelo reduccionista estaba implícito en el diseño.

Para poder integrar el nuevo modelo de ANM en coordenadas internas se ha tenido que refactorizar el código.

Se han creado clases genéricas y sus respectivas subclases para coordenadas cartesianas e internas (ver Figura 29). Un ejemplo es la clase ***AnmInternals***, que hereda de ***AnmAlgorithm***. En esta clase se han modificado las funciones de normalización y de creación de restricciones en los giros. Otro ejemplo es la clase ***UpdateTargetCoordinates***, se ha cambiado totalmente la forma de calcular las coordenadas objetivo ya que en internas se tienen desplazamientos angulares y no cartesianos.

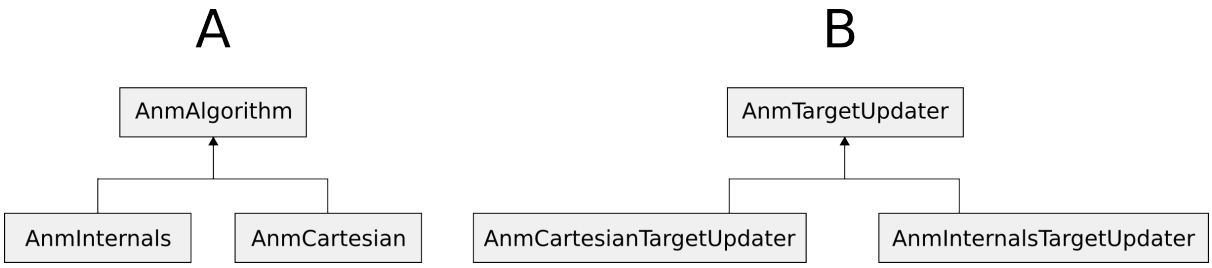


Figura 29 *Se han creado clases diferenciadas para internas y cartesianas. En* ***A*** *se puede ver que* ***AnmInternals*** *y* ***AnmCartesian*** *heredan ambas de* ***AnmAlgorithm***. *En* ***B*** *se pueden ver las subclases de* ***AnmTargetUpdater*** *para coordenadas internas y cartesianas.*

También se ha creado la clase ***AnmNodeList*** para generalizar el modelo reduccionista (ver Figura 30). La implementación original esperaba como entrada una lista de átomos, mientras que en el modelo reduccionista en coordenadas internas se usan unidades (implementadas en la clase ***Unit***), que constan de varios átomos, los diedros de los que forman parte e información extra que se necesitará en el cálculo de las diversas matrices. Al crear esta clase genérica se consigue evitar el efecto cascada que provocaría tener que adaptar cada una de las funciones y clases que utilizaban esta lista de átomos (se tendrían que haber duplicado muchas clases sólo por cambiar la signatura de la función).

Además, se ha creado una nueva función ***generateUnitNodeList*** en la clase ***AnmCalculatorBuilder*** que se encarga de generar el listado de unidades para el

cálculo en coordenadas internas.

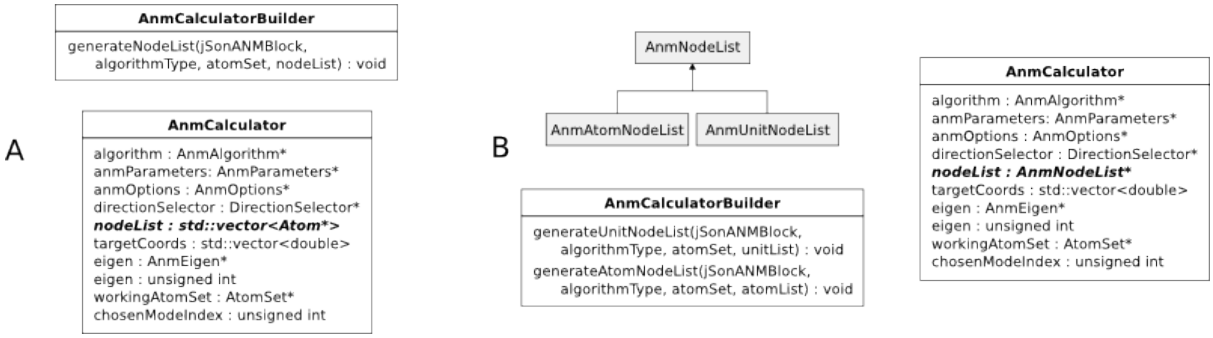


Figura 30 *En la implementación original (**A**) existía la función **generateNodeList** y la clase **AnmCalculator** tenía como atributo un vector de átomos **nodeList**. En la nueva implementación (**B**), se ha creado la clase **AnmNodeList** con una subclase para cada uno de los tipo de lista (de átomos y de unidades) y se han implementado las dos nuevas funciones para generar las listas correspondientes. Además, el atributo **nodeList** de **AnmCalculator** ahora es del tipo **AnmNodeList**.*

Con el nuevo diseño se ha conseguido reducir significativamente la complejidad del código y mejorar su legibilidad. Se ha logrado tener dos clases con implementaciones distintas para el paso de la aplicación del modelo atómico según el algoritmo de PELE utilizado. Disminuye también el acoplamiento entre clases ya que el modelo reduccionista deja de estar implícito en el diseño. Esto permitirá extender o modificar fácilmente el funcionamiento con nuevas implementaciones futuras si fuera necesario.

2.3.1. Aplicación del modelo atómico

Como en el ANM en coordenadas cartesianas, en el caso de las coordenadas internas es igualmente necesario aplicar una simplificación a la proteína.

Se representa la proteína como una sucesión de unidades rígidas unidas entre sí a través de los enlaces que forman los ángulos diedros ψ y φ. De esta forma, cada una de las unidades contiene un conjunto de átomos cuya posición relativa no varía durante el movimiento.

El ángulo diedro ψ_i lo forman la sucesión de cuatro átomos $N_i - C_i^{\alpha} - C_i - N_{i+1}$ mientras que el ángulo φ_i lo forman $C_{i-1} - N_i - C_i^{\alpha} - C_i$. Existe otro tipo tipo de ángulo diedro, el ángulo ω, formado por $C_i^{\alpha} - C_i - N_{i+1} - C_i^{\alpha}$, pero en este caso no se va a utilizar en el cálculo ya que no proporciona información relevante.

Partiendo de la siguiente cadena de ejemplo,

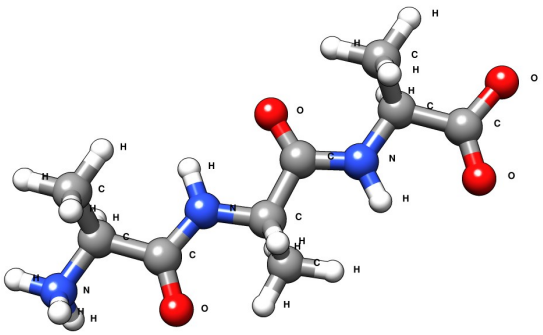


Figura 31 *Representación inicial de la proteína con todos sus átomos*

se representa simplificada sin tener en cuenta los átomos de hidrógeno, es decir, considerando solamente los átomos más pesados de la proteína (oxígeno, nitrógeno y carbono).

Se pueden ver marcados los ángulos diedros ψ y φ mencionados sobre la cadena de ejemplo:

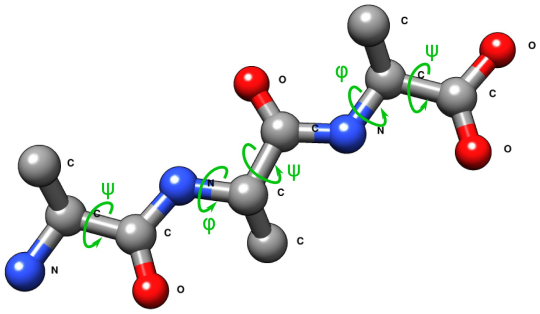


Figura 32 *Representación de la cadena sin los átomos de hidrógeno. En verde se muestran los ángulos diedros.*

Con lo cual la subdivisión en unidades aplicada a la cadena de ejemplo quedaría:

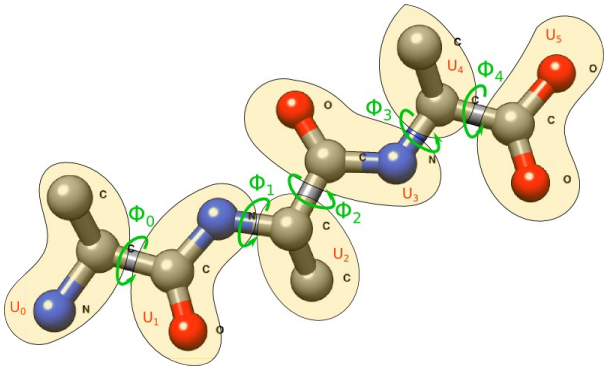


Figura 33 *Representación de la cadena una vez dividida en 6 unidades independientes.*

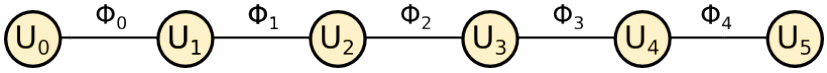


Figura 34 *Representación esquemática de las unidades*

2.3.2. Cálculo del Hessiano

El siguiente paso es el cálculo del Hessiano H que corresponde a las segundas derivadas parciales de la energía potencial. Para realizar este cálculo de forma eficiente se utiliza el método rápido de Go [15].

En este método los átomos localizados a un lado de los enlaces α y β se mantienen fijos y son sólo los de los otros lados los afectados por el movimiento de rotación del ángulo diedro. De esta forma se define una parte fija y una móvil para cada enlace α y β.

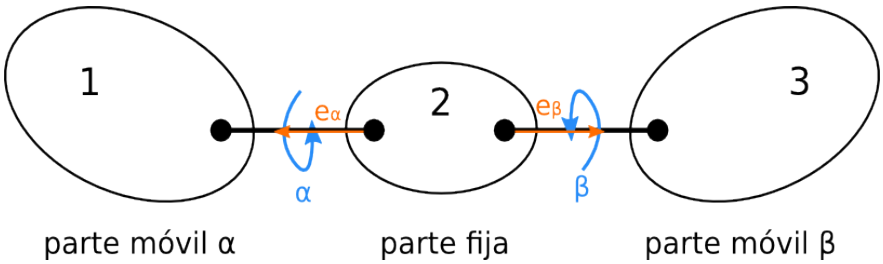


Figura 35 *La molécula se representa como dos partes móviles (1 y 3) enlazadas a la parte fija (2) por medio de los enlaces α y β*

Cada elemento de la matriz H se calcula como:

$$H_{\alpha\beta}=(e_\alpha, e_\alpha\times r_\alpha)R_{\alpha\beta}\begin{pmatrix} e_\beta \\ e_\beta\times r_\beta \end{pmatrix} \tag{10}$$

donde:

$$R_{\alpha\beta}=\sum_{i\subset\alpha}\sum_{j\subset\beta} D_{ij} \tag{11}$$

$$D_{ij}=\frac{f_{ij}}{|r_{ij}|^2}\begin{pmatrix} r_i\times r_j \\ r_{ij} \end{pmatrix}(r_i\times r_j, r_{ij}) \tag{12}$$

e_α, e_β son vectores unitarios que determinan la dirección de cada enlace, r_α, r_β representan la posición de un punto situado en el enlace, r_i, r_j son las posiciones de los

átomos i, j y f_{ij} es la constante de fuerza del muelle que se encuentra entre los átomos i, j.

La matriz $R_{\alpha\beta}$ guarda la información de las interacciones entre los átomos que intervienen en las rotaciones de los enlaces α y β.

Gracias al modelo reduccionista aplicado se consigue agrupar en una sola matriz T_{ab} toda la información sobre las interacciones entre cada uno de los átomos de las unidades a y b:

$$T_{ab}=\sum_{i\in a}\sum_{j\in b} D_{ij} \tag{13}$$

Se puede redefinir el cálculo de la matriz $R_{\alpha\beta}$ como:

$$R_{\alpha\beta}=\sum_{a\subset\alpha}\sum_{b\subset\beta} T_{ab} \tag{14}$$

Para optimizar más el cálculo de $R_{\alpha\beta}$, los sumatorios se puede hacer recursivamente utilizando las unidades u definidas anteriormente. De esta forma no es necesario guardar todos los elementos T_{ab} en memoria:

$$U_{ab}=U_{a,b+1}+U_{a-1,b}-U_{a-1,b}+T_{ab} \tag{15}$$

Finalmente, para conseguir $R_{\alpha\beta}$ a partir de los elementos U_{ab} tan sólo hay que relacionar sus índices. Los índices a y b se refieren a unidades y los α y β se refieren a ángulos diedros. Por lo tanto, la relación es la siguiente:

$$a=\alpha \quad \text{y} \quad b=\beta+1 \tag{16}$$

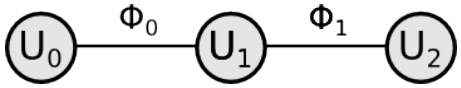


Figura 36 *Por ejemplo, en el calculo de $R_{0,1}$ las unidades afectadas son la 0 y 2, es decir, $U_{0,2}$*

Así pues:

$$R_{\alpha\beta}=U_{\alpha,\beta+1} \tag{17}$$

2.3.3. Cálculo de la matriz K

El siguiente paso es el cálculo de la matriz K que corresponde con las segundas derivadas parciales de la energía cinética:

$$K_{\alpha\beta}=\frac{M_1 M_3}{M}[e_\alpha\times(r_\alpha-r_1^0)][e_\beta\times(r_\beta-r_3^0)] \\ +[M_1 r_1^0\times(e_\alpha\times r_\alpha)-I_1 e_\alpha]I^{-1}[M_3 r_3^0\times(e_\beta\times r_\beta)-I_3 e_\beta] \quad (18)$$

Los valores de M_1 y M_3 corresponden a la suma de las masas de las partes rígidas situadas entre el inicio de la cadena y el enlace α, y el enlace β y el final de la cadena. M es la masa de todo el sistema.

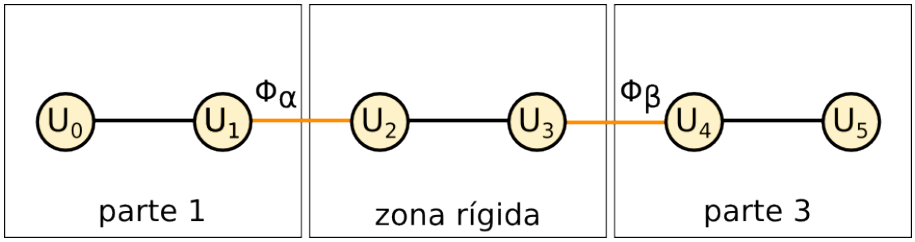


Figura 37 *En la ilustración se puede ver la división de la cadena de unidades en 3 partes rígidas enlazadas por los enlaces α y β.*

r_1^0 y r_3^0 son las posiciones del centro de masas de las partes 1 y 3 respectivamente.

e_α y e_β son vectores unitarios que especifican la dirección de los enlaces α y β.

Por último, la matriz I se corresponde con el tensor de inercia (matriz simétrica que caracteriza la distribución de masas del sistema a medida que éste gira).

2.3.4. Cálculo de *eigenvalues* y *eigenvectors*

Una vez se han calculado la matrices H y K se puede resolver la ecuación:

$$HV=\Lambda KV \quad (19)$$

Y así obtener la matriz **V** que contiene los N modos normales (V = v_1, …, v_N) y **Λ**, la matriz diagonal de *eigenvalues*.

Los *eigenvectors* resultantes (ver Figura 38) son ángulos infinitesimales que definen el desplazamiento angular de cada diedro de la cadena.

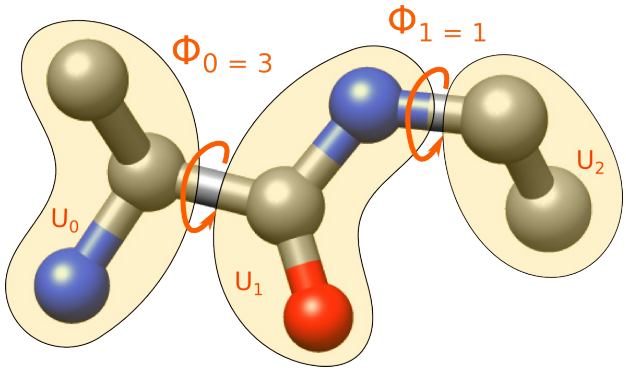


Figura 38 *En este ejemplo se pueden ver tres unidades unidas por dos ángulos diedros. En* $\boldsymbol{\Phi}_0$ *y en* $\boldsymbol{\Phi}_1$ *los eigenvector calculados tiene valores de magnitud 3 y 1 respectivamente.*

2.3.5. Normalización

De la misma forma que en coordenadas cartesianas, en este caso también se lleva a cabo la normalización de los *eigenvectors* obtenidos (ver Figura 39). Se toma como referencia el mayor de los ángulos calculados y se dividen todos los demás entre su magnitud.

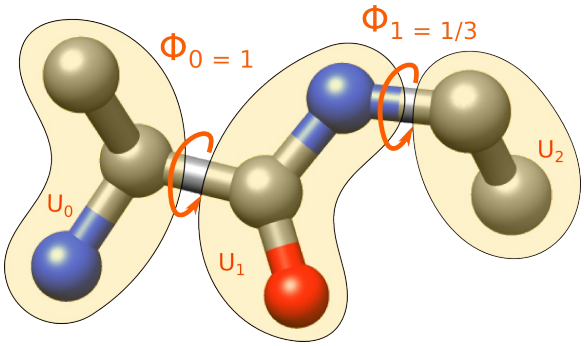


Figura 39 *El valor indicado en cada ángulo representa la proporción del giro después de normalizarlo.*

2.3.6. Cálculo de ángulos objetivo

En este paso se va a determinar la posición final de los átomos (ver Figura 40 y 41). El desplazamiento angular objetivo se obtiene combinando los modos normalizados en el paso anterior, normalizándolos de nuevo y asignándoles una magnitud máxima elegida por el usuario. A continuación, se aplican los desplazamientos a cada ángulo diedro utilizando cuaterniones (operaciones geométricas de rotación) [32].

Se conserva el nombre de la función para mantener el mismo esquema original de PELE, pero se realizan dos pasos al mismo tiempo: el cálculo de los desplazamientos y la aplicación del movimiento (equivalente al paso de la minimización en coordenadas cartesianas).

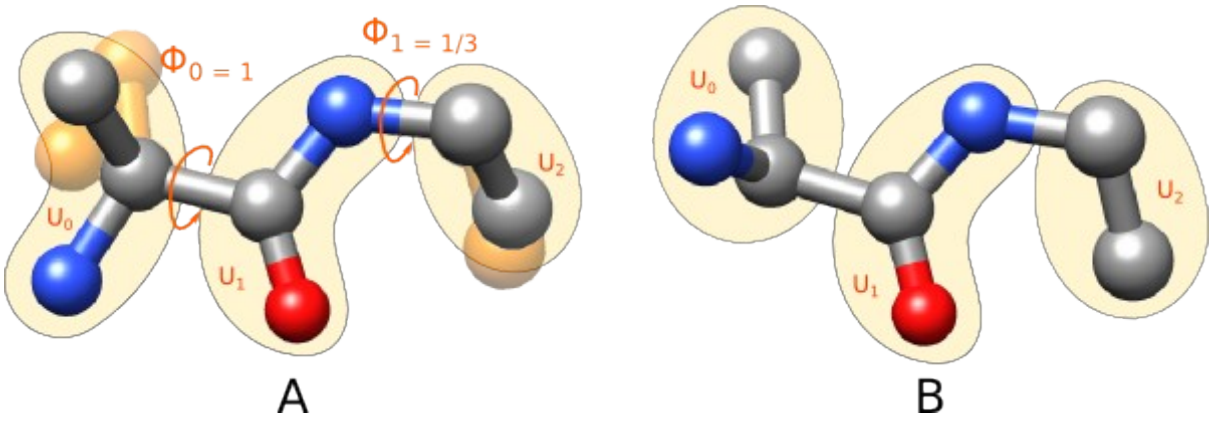


Figura 40 *En* ***A*** *se puede ver en color naranja transparente la posición final objetivo de los átomos. En* ***B*** *observa la posición final de los átomos de la cadena después del desplazamiento.*

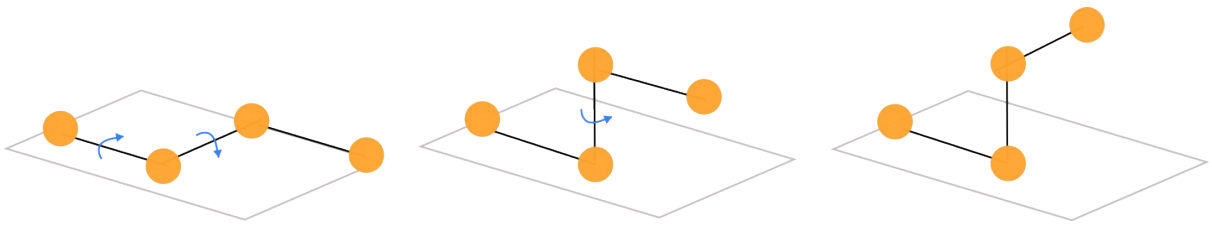


Figura 41 *Aplicación de rotaciones sucesivas en cada uno de los ángulos diedros.*

2.3.7. Minimización

Debido a la forma de mover los átomos en el algoritmo en coordenadas internas, ya no es necesario el paso de la minimización con el ahorro computacional que esto conlleva.

2.3.8. Implementación en PELE

Igual que en la implementación en coordenadas cartesianas, PELE crea un objeto ***AnmCalculator*** con los parámetros indicados en el archivo JSON de configuración.
En este caso, la aplicación del modelo atómico (paso 1) se realiza mediante la función ***AnmCalculatorBuilder::generateUnitNodeList()*** que crea el conjunto de unidades a partir de los átomos de la proteína. A continuación ***AnmAlgorithmBuilder::createAlgorithm()*** crea un objeto de la clase ***AnmAlgorithm*** del tipo ***AnmInternals*** configurado con un objeto ***InternalModesCalculator***, un ***AnmInternalsTargetUpdater*** y un ***Picker***.
El ***Picker*** se encarga de seleccionar los modos normales que se van a utilizar durante el cálculo de la perturbación, se genera con la función ***PickerBuilder:: createPicker()*** y se elige su tipo según el archivo de configuración.

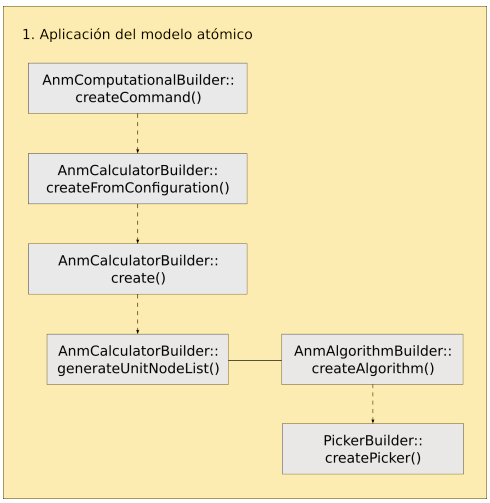


Figura 42 *Esquema de las funciones encargadas de aplicar el modelo atómico.*

Durante el cálculo de los modos normales ***AnmAlgorithm::calculateModes()*** utiliza la función ***InternalModesCalculator::calculateEigenValues AndVectors()*** para obtener los modos (paso 4). Además, para calcularlos es necesaria la matriz H (paso 2) matriz K que se consigue mediante la función ***ANMICKineticMatrixCalculator::calculateK()*** (paso 3). Esta matriz no era necesaria en el caso de coordenadas internas, ya que se utilizaban coordenadas pesadas por las masas.

Para calcular los ángulos objetivo (paso 6) en este caso se utiliza la función ***AnmInternalsTargetUpdater::updateTargetCoordinates()*** que se encarga de obtener los vectores de desplazamiento, normalizarlos (paso 5) y actualizar las coordenadas de las unidades.

Finalmente ***AnmCalculator::minimize()*** llevar a cabo la minimización (paso 7).

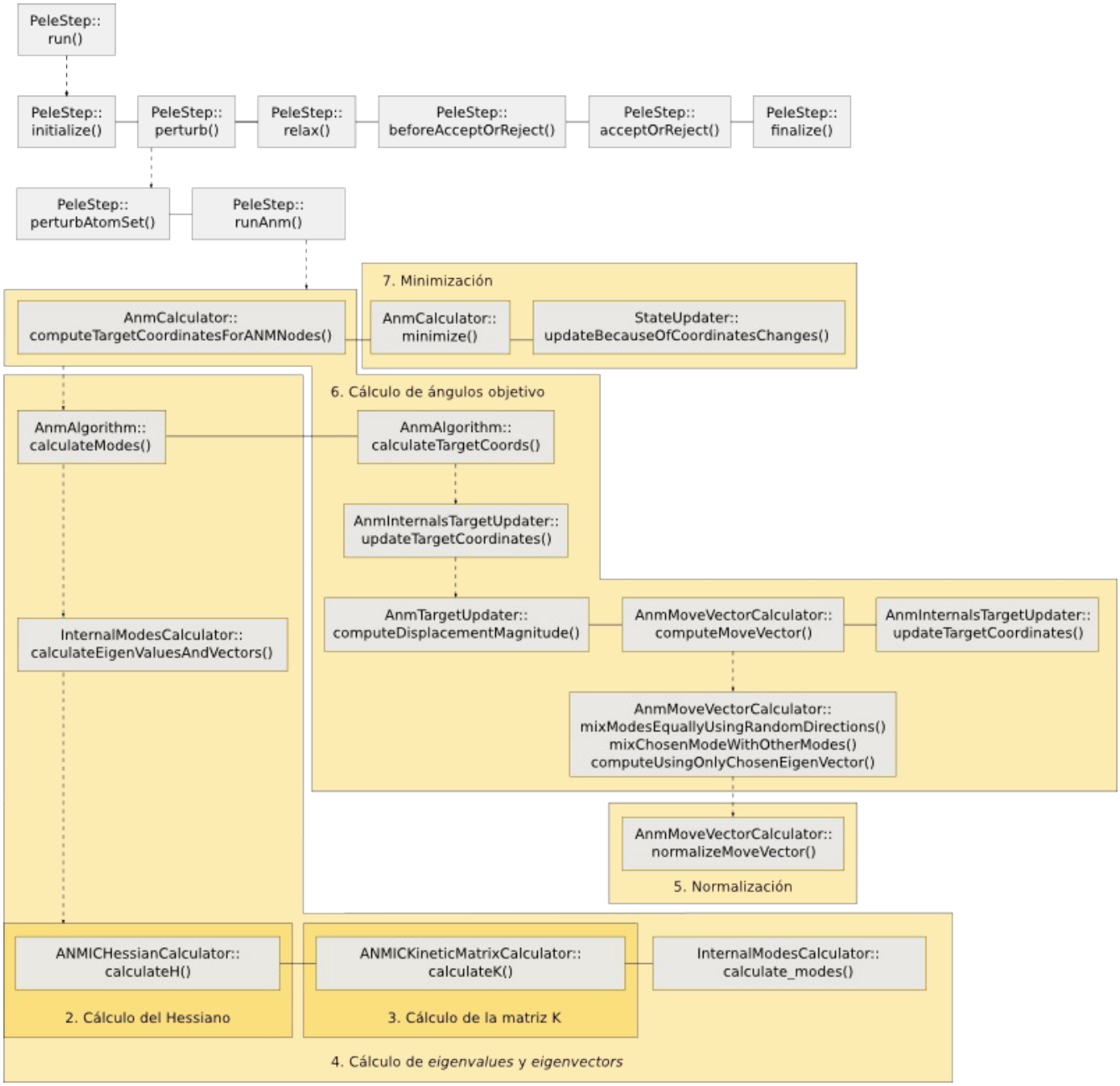


Figura 43 *Esquema de clases y funciones para cada paso de la nueva implementación en coordenadas internas de la etapa de la perturbación local.*

2.3.9. Minimización en PELE

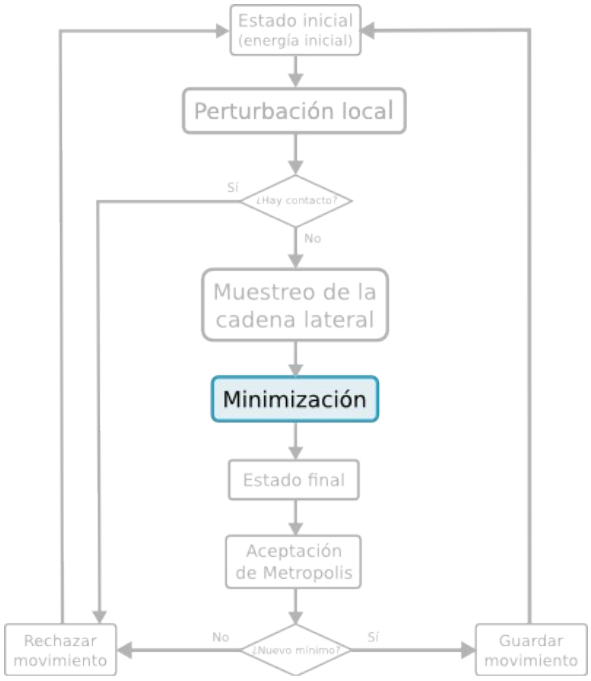


Figura 44 *Esquema de PELE donde se observa la etapa de minimización*

Igual que en coordenadas cartesianas, la etapa de la minimización desplazará los átomos de la proteína para intentar reducir su energía total.

En este caso, se deberá evitar que los ángulos diedros vuelvan a su posición original previa a la perturbación local añadiéndoles nuevas restricciones. Para ello, los ángulos actuales pasan a ser los ángulos de equilibrio y se les asignará una constante elástica alta para fijarlos en su posición y evitar que se desplacen a la ubicación inicial (ver apartado 2.4).

2.3.10. Funciones para el cálculo de H

Para conseguir la matriz hessiana de manera eficiente se deben calcular previamente los elementos de la matriz R a partir de la matriz U.

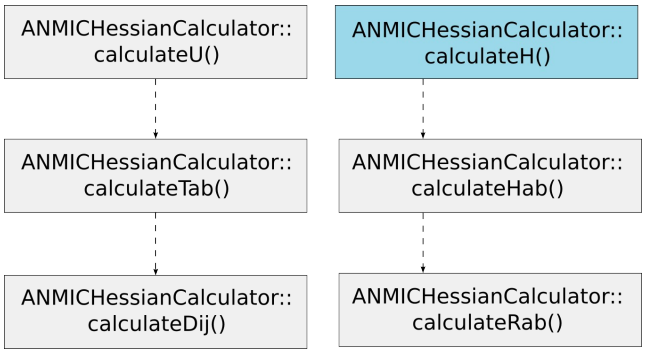
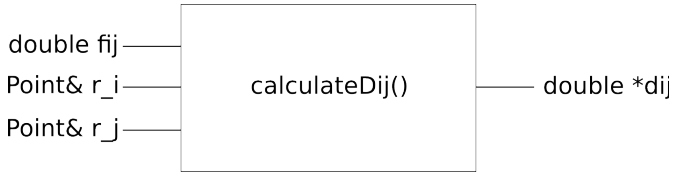


Figura 45 *Esquema de funciones de la implementación en coordenadas internas para el calculo del hessiana*

2.3.10.1. calculateDij



La función calcula la matriz D_{ij} que contiene la información sobre la interacción del par de átomos i y j.

$$D_{ij} = \frac{f_{ij}}{|r_{ij}|^2} \begin{pmatrix} r_i \times r_j \\ r_{ij} \end{pmatrix} (r_i \times r_j , r_{ij}) \tag{20}$$

Parámetros:

fij: constante de fuerza del muelle situado entre los átomos i y j.

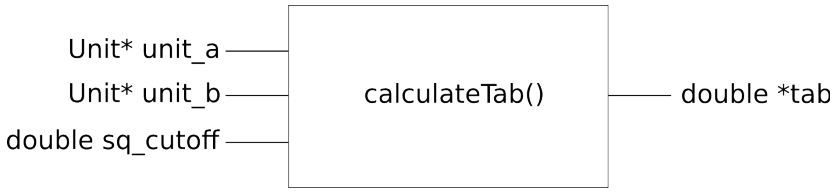
r_i: punto del espacio donde se encuentra el átomo i.

r_j: punto del espacio donde se encuentra el átomo j.

Salida:

Matriz D_{ij} de 6x6 elementos.

2.3.10.2. calculateTab



Calcula la matriz T_{ab} que almacena las interacciones existentes entre las unidades a y b. T_{ab} se calcula como la el sumatorio de las matrices de interacción D_{ij} entre cada uno de los átomos que componen las unidades a y b.

Utiliza la función ***calculateDij***.

$$T_{ab}=\sum_{i\in a}\sum_{j\in b} D_{ij} \tag{21}$$

Parámetros:

unit_a: unidad a.

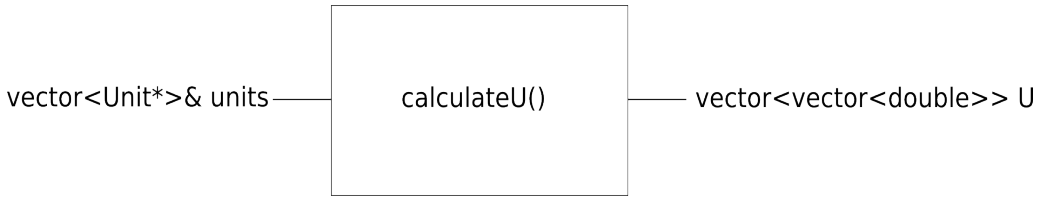
unit_b: unidad b.

sq_cutoff: distancia de *cut-off* al cuadrado.

Salida:

Matriz T_{ab} de 6x6 elementos.

2.3.10.3. calculateU



Calcula la matriz U que contiene la información de interacción de todas las unidades de la cadena. Cada elemento U_{ab} de la matriz guarda la información sobre la interacción entre las unidades a y b donde $a\in[0,a]$ y $b\in[b,M-1]$ (siendo M el número total de unidades).

Este cálculo se lleva a cabo de forma recursiva mediante la siguiente fórmula:

$$U_{ab} = U_{a,b+1} + U_{a-1,b} - U_{a-1,b} + T_{ab} \tag{22}$$

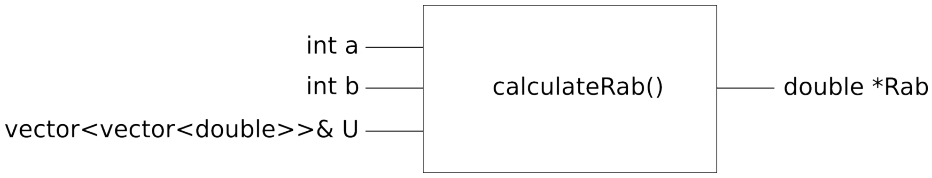
Parámetros:

units: vector que contiene todas las unidades de la proteína.

Salida:

Matriz simétrica U de (6*M)x(6*M) elementos.

2.3.10.4. calculateRab



Devuelve la matriz $R_{\alpha\beta}$ que contiene la información sobre las interacciones entre las unidades de la parte móvil de los enlaces α y β.

$$R_{\alpha\beta} = U_{a,\beta+1} \tag{23}$$

Parámetros:

a: índice de la unidad a (entre 0 y M-1).

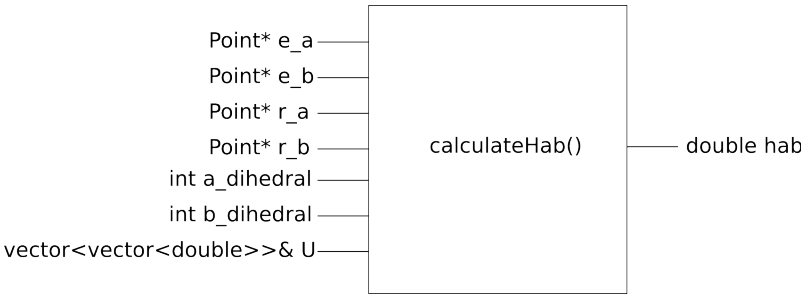
b: índice de la unidad b (entre 0 y M-1).

U: matriz U.

Salida:

Matriz $R_{\alpha\beta}$ de 6x6 elementos.

2.3.10.5. calculateHab



Esta función calcula el elemento de la matriz hessiana correspondiente a los ángulos diedros α y β. Utiliza la función ***calculateRab***.

$$H_{\alpha\beta}=(e_\alpha, e_\alpha \times r_\alpha) R_{\alpha\beta} \begin{pmatrix} e_\beta \\ e_\beta \times r_\beta \end{pmatrix} \tag{24}$$

Parámetros:

e_a: vector unitario que indica la dirección del enlace α.

e_b: vector unitario que indica la dirección del enlace β.

r_a: posición del átomo situado en el enlace α.

r_b: posición del átomo situado en el enlace β.

a_dihedral: índice correspondiente al ángulo diedro α.

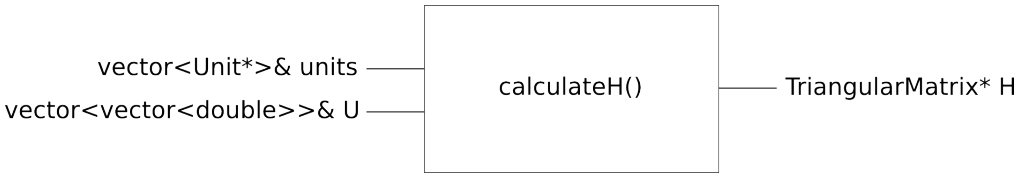
b_dihedral: índice correspondiente al ángulo diedro β.

U: matriz U.

Salida:

Elemento $H_{\alpha\beta}$.

2.3.10.6. calculateH



Esta función calcula todos los elementos del Hessiano a partir de la función ***calculateHab***.

Parámetros:

units: vector que contiene todas las unidades de la proteína.

U: matriz U.

Salida:

Matriz hessiana *H*.

2.3.11. Funciones para el cálculo de K

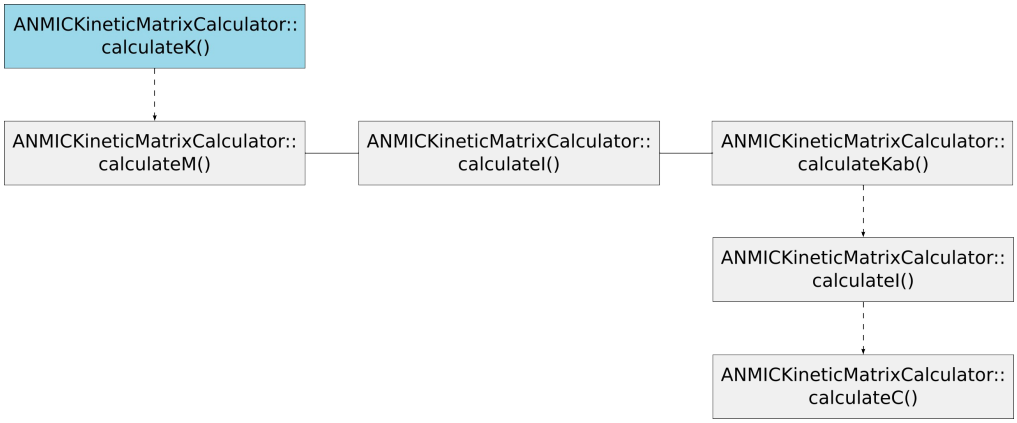
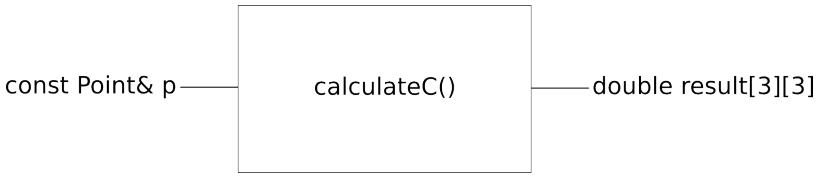


Figura 46 *Diagrama de funciones para el cálculo de la matriz K*

2.3.11.1. calculateC



Calcula la matriz C correspondiente al punto indicado como parámetro. [31]

$$C=\begin{bmatrix} p_y{}^2+p_z{}^2 & -p_y p_x & -p_z p_x \\ -p_x p_y & p_x{}^2+p_z{}^2 & -p_z p_y \\ -p_x p_z & -p_y p_z & p_x{}^2+p_y{}^2 \end{bmatrix} \tag{25}$$

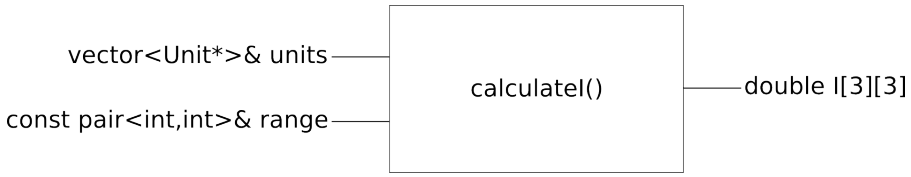
Parámetros:

p: posición del átomo en el espacio.

Salida:

Matriz C de 3x3 elementos.

2.3.11.2. calculateI



Esta función calcula el tensor de inercia correspondiente a las unidades indicadas por el rango. Realiza un sumatorio para cada átomo que se encuentre en el rango indicado, multiplicando la masa del átomo por su matriz C correspondiente. [31]

$$\sum_{i}^{N} m_i C_i \tag{26}$$

Parámetros:

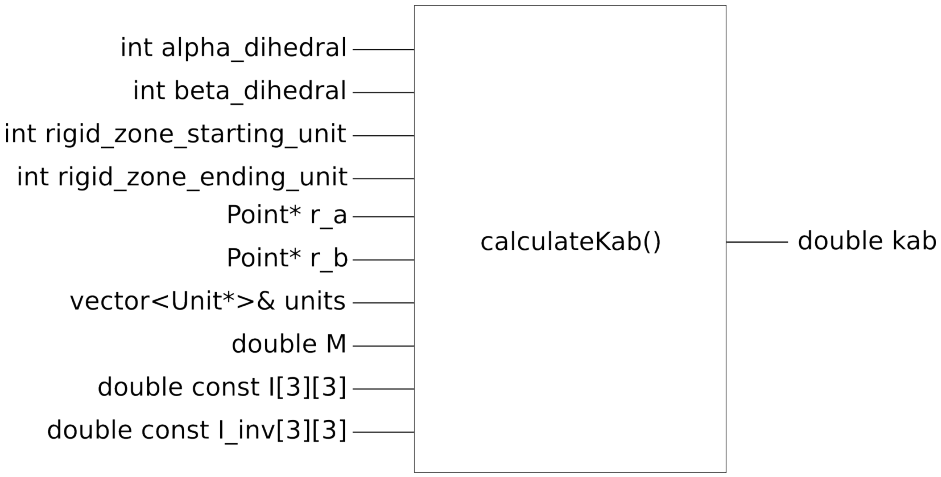
units: vector que contiene todas las unidades de la proteína.

range: enteros que indican los índices de las unidades de inicio y final a tener en cuenta en el cálculo de I.

Salida:

Matriz I de 3x3 elementos.

2.3.11.3. calculateKab



Esta función calcula el elemento de la matriz cinética correspondiente a los ángulos diedros α y β.

$$K_{\alpha\beta}=\frac{M_1 M_3}{M}[e_\alpha\times(r_\alpha-r_1^0)][e_\beta\times(r_\beta-r_3^0)] \\ +[M_1 r_1^0\times(e_\alpha\times r_\alpha)-I_1 e_\alpha]I^{-1}[M_3 r_3^0\times(e_\beta\times r_\beta)-I_3 e_\beta] \tag{27}$$

Parámetros:

alpha_dihedral: índice del ángulo diedro α.

beta_dihedral: índice del ángulo diedro β.

rigid_zone_starting_unit: índice de la primera unidad de la parte fija.

rigid_zone_ending_unit: índice de la última unidad de la parte fija.

r_a: posición del átomo situado en el enlace α.

r_b: posición del átomo situado en el enlace β.

units: vector que contiene todas las unidades de la proteína.

M: masa total del sistema (suma de las masas de cada unidad).

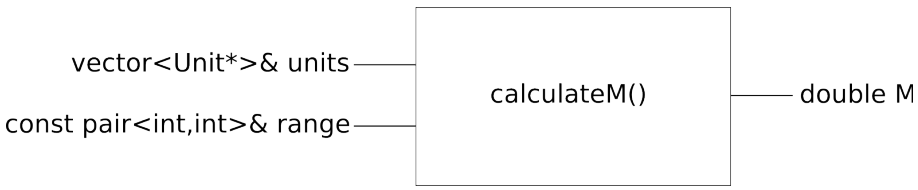
I: matriz tensor de inercia del sistema.

I_inv: matriz I invertida.

Salida:

Elemento $K_{\alpha\beta}$.

2.3.11.4. calculateM



La función calcula la suma de las masas de la unidades comprendidas entre las posiciones inicial y final indicadas por la variable de entrada **range**.

Parámetros:

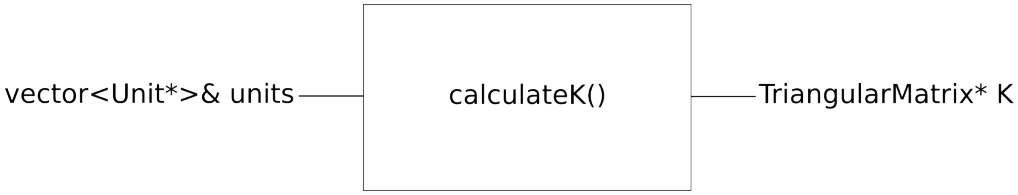
units: vector que contiene todas las unidades de la proteína.

range: enteros que indican los índices de las unidades de inicio y final a tener en cuenta en el cálculo.

Salida:

Masa total M.

2.3.11.5. calculateK



Esta función calcula todos los elementos de la matriz de energía cinética K a partir de la función ***calculateKab***.

Parámetros:

units: vector que contiene todas las unidades de la proteína.

Salida:

Matriz K.

2.4. Nuevas restricciones en el diedro

Antes del paso de aceptación, PELE intenta relajar el estado del sistema con una última minimización. En cualquiera de las minimizaciones se intenta cumplir un conjunto de restricciones (ver Figura 47) dadas por el campo de fuerza:

1. De distancia entre átomos (cada 2 átomos)
2. De ángulo (cada 3 átomos)
3. De ángulo diedro (cada 4 átomos)
4. Electrostáticas y de Lennard-Jones

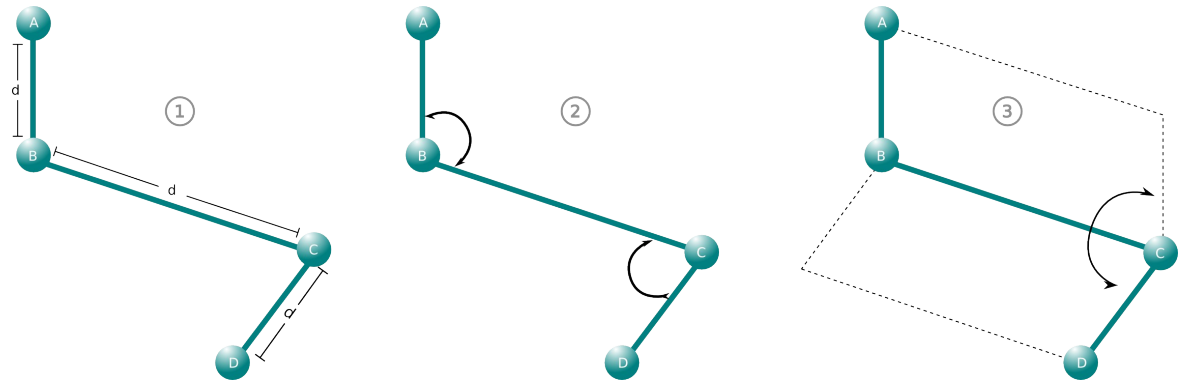


Figura 47 *Restricciones iniciales de la molécula*

En el paso final se han tenido que añadir además las nuevas restricciones en los diedros introducidas por el ANM. Su función es fijar los átomos de la proteína en sus nuevas posiciones para que la minimización no revierta los cambios.

Se espera que el proceso de minimización deje a la proteína en un estado tal que su energía potencial esté cerca de un mínimo y se respeten simultáneamente las nuevas restricciones y las iniciales (en la medida de lo posible).

El minimizador utiliza un algoritmo de truncado de Newton que requiere el cálculo de diversas funciones para poder añadir una nueva restricción:

1. **Energía potencial** (dados cuatro átomos que forman un ángulo diedro)
2. **Gradiente** (primera derivada de la energía)
3. **Hessiano** (matriz cuadrada de las segundas derivadas parciales de energía)

El código para calcular estas tres funciones y los juegos de prueba para los *tests* se han generado con la ayuda de *wxMaxima*, un software de cálculo simbólico. Aunque estos cálculos no son muy complejos, realizarlos de forma manual es costoso y podría introducir errores, por eso se decidió automatizar el proceso. *WxMaxima* ha permitido generar las expresiones de las fórmulas, calcular las derivadas y transformarlas a código C++ para integrarlo en la estructura de PELE.

2.4.1. Ángulo diedro

En las fórmulas para el cálculo de la energía, el gradiente y el Hessiano, interviene el ángulo diedro, por lo tanto, el primer paso es programar una función para calcularlo. Un ángulo diedro [13] corresponde con el espacio limitado por la intersección de dos planos. Dados cuatro puntos, en este caso cuatro átomos de la proteína, se puede ver cómo forman los dos planos que delimitan el espacio del ángulo (ver Figura 48).

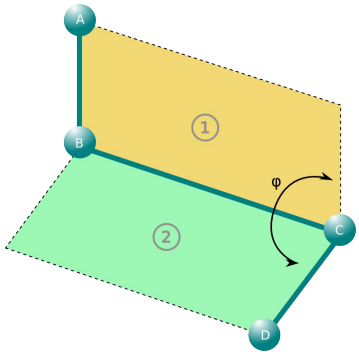


Figura 48 *El ángulo diedro es el delimitado entre los planos 1 y 2*

Puesto que un plano se puede definir de diferentes maneras, existen varios métodos para calcular el ángulo diedro que forman.

1. Se puede calcular como el ángulo que forman los dos vectores perpendiculares a cada uno de los planos, es decir:

$$\varphi = \arccos\left(\frac{\vec{u}\cdot\vec{v}}{\|\vec{u}\|\cdot\|\vec{v}\|}\right) \tag{28}$$

2. También se puede calcular mediante los tres vectores que definen los enlaces [14].

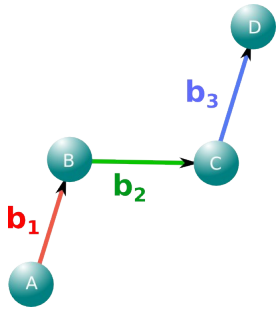


Figura 49 *b_1 y b_2 definen el primer plano y b_2 y b_3 definen el segundo.*

$$n_1 = \frac{b_1 \times b_2}{\|b_1 \times b_2\|} \qquad n_2 = \frac{b_2 \times b_3}{\|b_2 \times b_3\|} \qquad m_1 = n_1 \times \left(\frac{b_2}{\|b_2\|}\right) \qquad x = n_1 \cdot n_2 \qquad y = m_1 \cdot n_2 \tag{29}$$

$$\varphi = arctan2(y, x) \tag{30}$$

El método utilizado en el cálculo del ángulo diedro es el segundo, dado que en el primero el ángulo queda indefinido cuando es cercano en 0º o 180º. Se ha expresado la fórmula en *wxMaxima* para poder generar el código en C++ de la función.

2.4.2. Energía potencial

La energía potencial depende de la posición de los átomos de la proteína y se calcula mediante la fórmula:

$$E = k(\Theta - \Theta_{eq})^2 \tag{31}$$

Donde k es la constante de fuerza definida por los átomos, Θ_{eq} el ángulo de equilibrio y Θ es el ángulo diedro formado por cuatro nodos consecutivos.

La función que calcula la energía tendrá como parámetros k, las coordenadas (x, y, z) de cada uno de los átomos y el ángulo de equilibrio Θ_{eq}.

Para el cálculo con *wxMaxima* de la fórmula de la energía se utiliza la segunda función del ángulo diedro explicada anteriormente.

2.4.3. Gradiente

El vector gradiente se calcula mediante las primeras derivadas parciales de la energía potencial en función de cada una de las coordenadas de (x, y, z) de los átomos:

$$G = \left(\frac{\partial E}{\partial x_a}, \frac{\partial E}{\partial y_a}, \frac{\partial E}{\partial z_a}, \frac{\partial E}{\partial x_b}, \frac{\partial E}{\partial y_b}, \frac{\partial E}{\partial z_b}, \frac{\partial E}{\partial x_c}, \frac{\partial E}{\partial y_c}, \frac{\partial E}{\partial z_c}, \frac{\partial E}{\partial x_d}, \frac{\partial E}{\partial y_d}, \frac{\partial E}{\partial z_d}\right) \tag{32}$$

Con *wxMaxima*, se calculan cada una de las 12 funciones necesarias para obtener el vector gradiente (una por cada coordenada de los cuatro átomos) y se convierten a código C++.

2.4.4. Hessiano

El Hessiano o matriz hessiana corresponde con la matriz simétrica cuadrada de las segundas derivadas parciales de la función energía potencial.

Con *wxMaxima* se calculan cada una de las derivadas que forman la matriz:

$$H_{ij}=\frac{\partial^2 E}{\partial i\,\partial j} \tag{33}$$

Donde los índices *i*, *j* corresponden a cada uno de las coordenadas (x, y, z) de las posiciones de los cuatro átomos.

Como la matriz es simétrica, sólo hará falta calcular la matriz triangular superior. Por tanto, el Hessiano contendrá 78 valores:

$$H=\begin{bmatrix} \frac{\partial^2 E}{\partial^2 x_a} & \frac{\partial^2 E}{\partial x_a \partial y_a} & \frac{\partial^2 E}{\partial x_a \partial z_a} & \cdots & \frac{\partial^2 E}{\partial x_a \partial z_d} \\ 0 & \frac{\partial^2 E}{\partial^2 x_b} & \frac{\partial^2 E}{\partial x_b \partial z_a} & \cdots & \frac{\partial^2 E}{\partial x_b \partial z_d} \\ 0 & 0 & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \frac{\partial^2 E}{\partial^2 z_d} \end{bmatrix} \tag{34}$$

2.4.5. Implementación

Para el cálculo de la energía, el gradiente y el hessiano con las nuevas restricciones introducidas se ha desarrollado en PELE la clase **HarmonicDihedral ConstraintTerm**.

Cada cuatro átomos consecutivos de la cadena de la proteína forman uno de estos términos y delimitan un ángulo diedro. De cada término se puede obtener el ángulo, la energía, el gradiente y el hessiano a partir de las coordenadas de los átomos que lo forman.

La clase ***HarmonicDihedralConstraintTerm*** utiliza la clase ***Harmonic DihedralConstraintFunctions*** que es la que contiene las funciones básicas generadas a partir del software *wxMaxima*.

2.4.5.1. Funciones de HarmonicDihedralConstraintFunctions

2.4.5.1.1. calculateDihedralAngleWithArcTanFunction

La función calcula el ángulo diedro formado por los cuatro puntos del espacio donde están situados los átomos *(a, b, c, d)* utilizando la segunda fórmula explicada en el apartado 2.4.1.

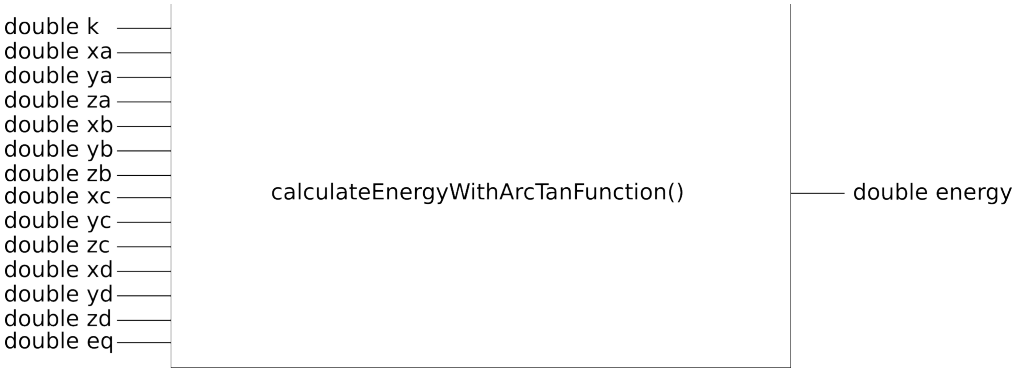
Parámetros:

xa, ..., zd: coordenadas *(x, y, z)* de cada uno de los 4 átomos *(a, b, c, d)* del término.

Salida:

Valor del ángulo diedro.

2.4.5.1.2. calculateEnergyWithArcTanFunction



Calcula la energía potencial del término compuesto por los cuatro átomos *(a, b, c, d)* basándose en el cálculo del ángulo del apartado 2.4.1.

Parámetros:

k: constante de fuerza del muelle.

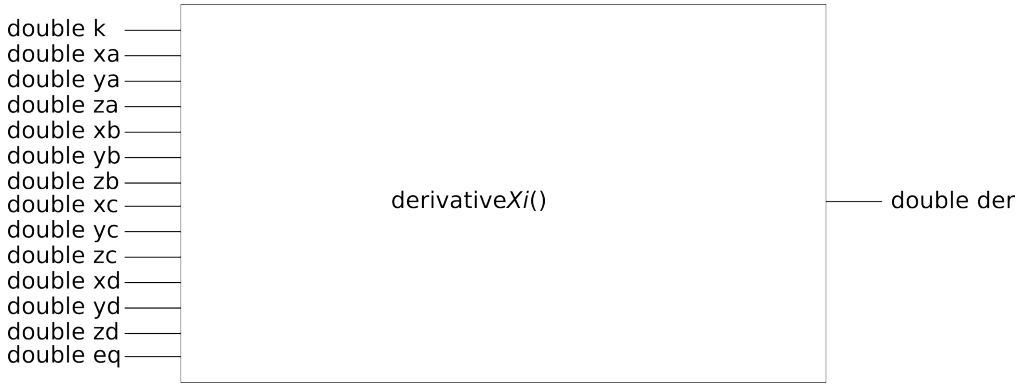
xa,…,zd: coordenadas *(x, y, z)* de cada uno de los átomos *(a, b, c, d)* del término.

eq: ángulo de equilibrio.

Salida:

Valor de la energía.

2.4.5.1.3. derivative*Xi*



Calcula la primera derivada parcial de la energía en función de la coordenada *Xi*, donde $X \in \{x, y, z\}$ y $y \in \{a, b, c, d\}$

Existe una de estas funciones para cada una de las coordenadas: *derivativeXa, derivativeYa, …, derivativeZd.*

Parámetros:

k: constante de fuerza del muelle.

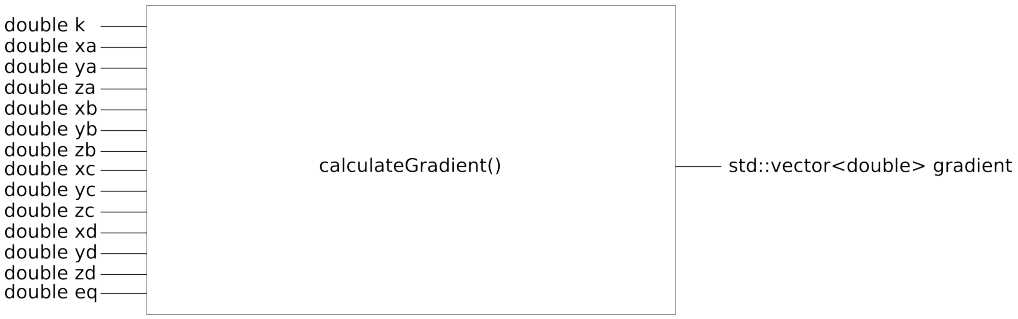
xa, …, zd: coordenadas *(x, y, z)* de cada uno de los 4 átomos *(a, b, c, d)* del término.

eq: ángulo de equilibrio.

Salida:

Valor de la derivada.

2.4.5.1.4. calculateGradient



Calcula el gradiente del término utilizando las funciones ***derivativeXa***, ***derivativeYa***, …, ***derivativeZd***.

Parámetros:

k: constante de fuerza del muelle.

xa, …, *zd*: coordenadas *(x, y, z)* de cada uno de los 4 átomos *(a, b, c, d)* del término.

eq: ángulo de equilibrio.

Salida:

Vector gradiente de 12 valores.

2.4.5.1.5. hessian***XiYj***

Calcula la segunda derivada parcial de la energía en función de las coordenadas *Xi* y

Yj, donde $X, Y \in \{x, y, z\}$ y $i, j \in \{a, b, c, d\}$

Existe una de estas funciones para cada par de coordenadas: ***hessianXaXa***, ***hessianXaYa***, *…*, ***hessianZdZd***.

Parámetros:

k: constante de fuerza del muelle.

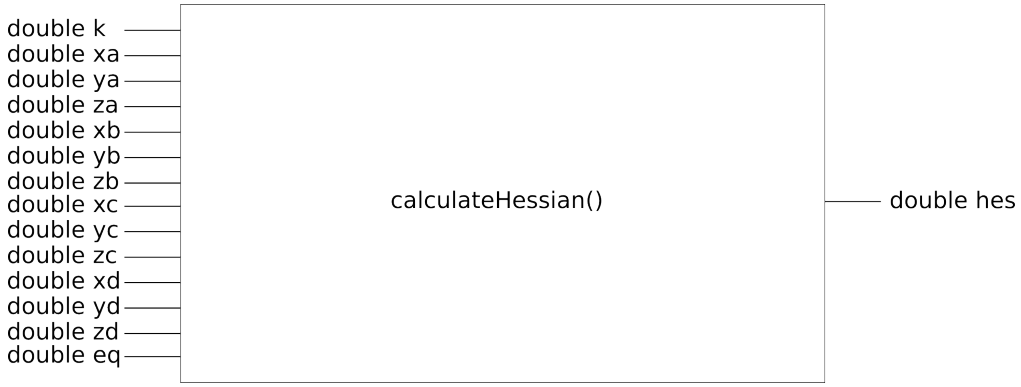
xa, …, zd: coordenadas *(x, y, z)* de cada uno de los 4 átomos *(a, b, c, d)* del término.

eq: ángulo de equilibrio.

Salida:

Valor de la segunda derivada.

2.4.5.1.6. calculateHessian



Calcula el hessiano del término utilizando las funciones *hessianXaXa, hessianXaYa, …, hessianZdZd*.

Parámetros:

k: constante de fuerza del muelle.

xa, …, zd: coordenadas *(x, y, z)* de cada uno de los 4 átomos *(a, b, c, d)* del término.

eq: ángulo de equilibrio.

Salida:

Matriz hessiana de 12x12 elementos.

2.4.5.2. Funciones de HarmonicDihedralConstraintTerm

2.4.5.2.1. getCurrentAngle

Calcula el ángulo formado por cuatro átomos consecutivos usando la función ***calculateDihedralAngleWithArcTanFunction***.

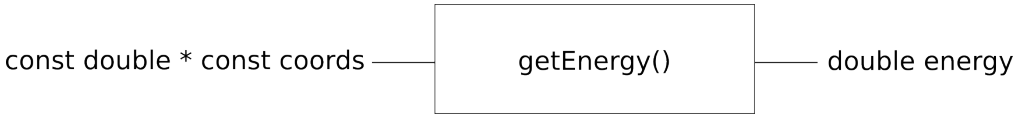
Parámetros:

coords: vector con las coordenadas en el espacio de los átomos que forman el término.

Salida:

Ángulo diedro.

2.4.5.2.2. getEnergy



Retorna la energía del término formado por los cuatro átomos del término usando la función ***calculateEnergyWithArcTanFunction***.

Parámetros:

coords: vector con las coordenadas en el espacio de los átomos del término.

Salida:

Energía potencial.

2.4.5.2.3. getEnergyGradient

Calcula el gradiente utilizando la función ***calculateGradient*** y lo actualiza en el vector ***grad*** mediante la función ***updateGradient***.

Parámetros:

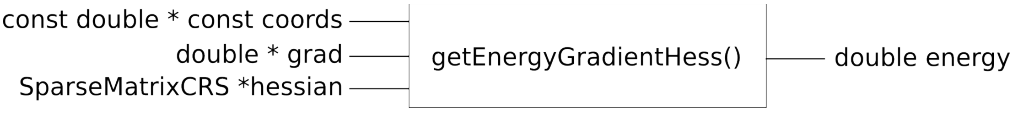
coords: vector con las coordenadas en el espacio de los átomos del término.

grad: vector donde se almacena el valor del gradiente.

Salida:

Valor de la energía.

2.4.5.2.4. getEnergyGradientHess



Calcula y actualiza el hessiano utilizando la función ***getHess***.

Calcula y actualiza el gradiente mediante la función ***getEnergyGradient***.

Parámetros:

coords: vector con las coordenadas en el espacio de los átomos del término.

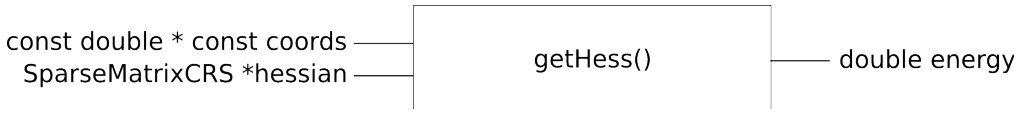
grad: vector donde se almacena el valor del gradiente.

hessian: matriz donde se almacena el valor del hessiano.

Salida:

Valor de la energía.

2.4.5.2.5. getHess



Calcula la matriz hessiana a través de la función ***calculateHessian*** y modifica su valor. Esta matriz tiene un tamaño muy grande, pero también tiene muchos ceros, por lo que se guarda en formato comprimido *CRS* (*Compressed Sparse Row*). Realizar cambios en la matriz no es trivial, y se debe hacer por bloques usando la función ***updateHessian***.

Parámetros:

coords: vector con las coordenadas en el espacio de los átomos del término.

hessian: matriz donde se almacena el valor del hessiano.

Salida:

Valor de la energía.

3. Testing y validación

3.1. Testing

En paralelo al desarrollo del proyecto se han implementado un conjunto de métodos para realizar pruebas unitarias para garantizar el correcto funcionamiento del código. Cada clase tiene asociada su propia clase de *testing* encargada de validar todas sus funciones.

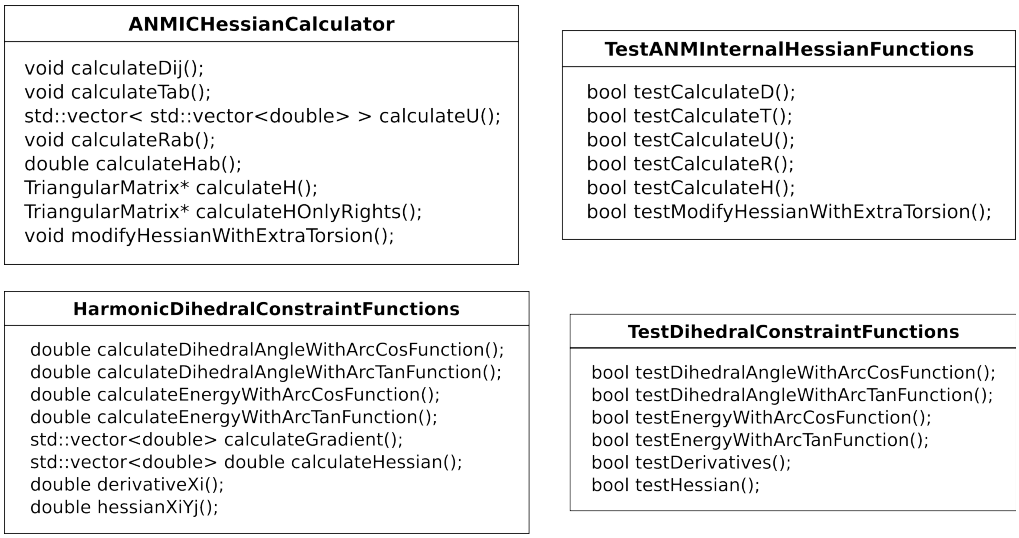


Figura 50 *Las clases ANMICHessianCalculator y HarmonicDihedral ConstraintFunctions tienen asociadas sus clases de testing TestANMICHessianCalculator y TestDihedralConstraintFunction respectivamente.*

Las clases de *testing* que se han implementado se integran con el *framework* de *tests* que utiliza PELE. Para ello es necesario que hereden de la clase **Test** e implementen tres funciones para cada conjunto de pruebas: ***init***, ***run*** y ***finish***.

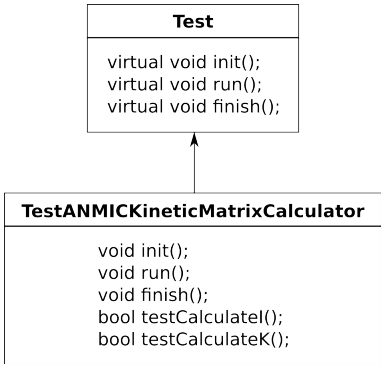


Figura 51 *La clase TestANMICKineticMatrixCalculator, igual que el resto de clases de test, hereda de la clase padre Test.*

La función ***init*** es la primera en ejecutarse y se encarga de configurar los *tests*: inicializaciones, importaciones de datos, etc.

A continuación, se ejecuta la función ***run*** que llama a cada una de las funciones de *test* de la clase pasando como parámetro los juegos de pruebas correspondientes. Cada función comprueba que los resultados son correctos mediante el uso de aserciones.

Por último, se llama a la función ***finish*** que libera los recursos utilizados (si fuera necesario).

Los juegos de pruebas se han obtenido gracias a la implementación que realizó José Ramón López Blanco para su tesis [15].

3.2. Validación

Para verificar la corrección de los modos normales se ha realizado una comparación del conjunto de los modos calculado en coordenadas cartesianas con el de los modos en coordenadas internas. La validación consta de dos pasos:

1. Transformación de los modos en coordenadas internas a coordenadas cartesianas.
2. Comparación de modos normales calculados en coordenadas cartesianas usando PELE con los modos transformados.

Transformación de los modos en coordenadas internas a coordenadas cartesianas

Para poder realizar esta transformación se ha desarrollado una función que está integrada en la clase ***InternalModesCalculator*** (ver Figura 49). La función toma como parámetros el vector de unidades de la proteína ***units*** y un objeto de la clase ***AnmEigen*** que contiene los *eigenvalues* y *eigenvectors* calculados en coordenadas internas. Después de la transformación retornará otro objeto ***AnmEigen*** con los *eigenvectors* y *eigenvalues* expresados en coordenadas cartesianas.

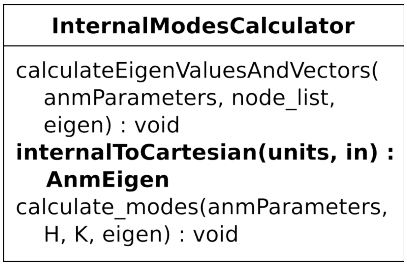


Figura 52 *La nueva función de validación* ***internalToCartesian*** *realiza la transformación de internas a cartesianas.*

La transformación de cada vector que representa uno de los modos normales se lleva a cabo mediante las siguientes fórmulas:

$$V_{ki}^{C}=\sum_{\alpha}^{N}\frac{\delta r_i}{\delta q_\alpha}V_{ak}^{I}$$

$$\frac{\delta r_1}{\delta q_\alpha}=e_\alpha\times\{\frac{M_2}{M}r_\alpha+\frac{M_1}{M}r_1^0\}-\{[M_1 r_1^0\times(e_\alpha\times r_\alpha)+I_2 e_\alpha]I^{-1}\}\times r_1 \qquad (35)$$

$$\frac{\delta r_2}{\delta q_\alpha}=-e_\alpha\times\{\frac{M_1}{M}r_\alpha+\frac{M_2}{M}r_2^0\}-\{[M_2 r_2^0\times(e_\alpha\times r_\alpha)+I_1 e_\alpha]I^{-1}\}\times r_2$$

donde V_{ki}^{C} es el vector i-ésimo del modo normal k , e_α es el vector unitario que especifica la dirección del enlace, r_α es la posición de un punto cualquiera localizado en el enlace α y V_{ak}^{I} representa el ángulo a del modo normal k.

Los puntos r_0 representan las posiciones de los centros de masas, las matrices I son los tensores de inercia y M la masa del cuerpo correspondiente. Los subíndices 1 y 2 se refieren a la partes fijas a cada lado del enlace α .

Comparación de modos normales calculados en coordenadas cartesianas usando PELE con los modos transformados

Se comprobará que los *eigenvectors* calculados usando el algoritmo en coordenadas cartesianas y en internas son equivalentes. Esto garantizará que el movimiento de la proteína es el mismo.

Ejecutando el algoritmo en coordenadas cartesianas se obtiene un *eigenvector* por cada nodo (Cα, ver Figura 53). En cambio, con el algoritmo en coordenadas internas se obtiene un vector que representa rotaciones en los diedros ψ y φ del *backbone* de la proteína. Al expresar este último vector en cartesianas mediante las fórmulas (35), se consigue un vector de desplazamiento por cada átomo de la proteína (ver Figura 54).

Para poder comparar ambos resultados se deben seleccionar, en el vector transformado, solamente las tripletas asociadas a los átomos Cα.

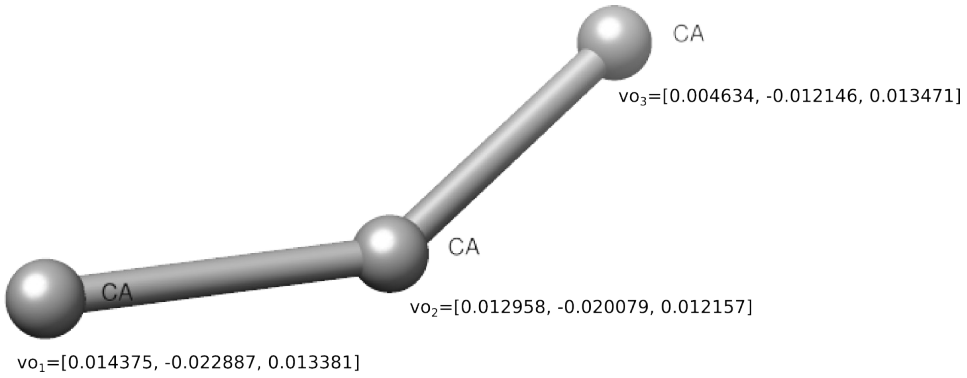


Figura 53 *Cadena de ejemplo con 3 nodos centrados en el átomo de Cα. Los vectores* vo_1, vo_2, vo_3 *componen el modo normal original elegido.* $mno=[vo_1, vo_2, vo_3]$

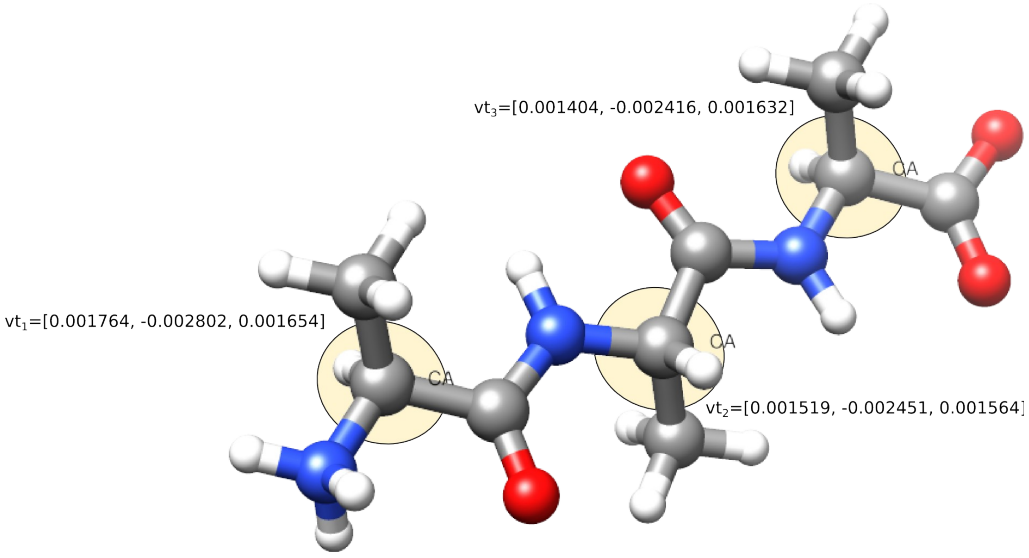


Figura 54 *Cadena de ejemplo con todos los átomos, se seleccionan tan solamente los vectores de los átomos de Cα.*

Los vectores vt_1, vt_2, vt_3 *forman el modo normal transformado a validar.* $mnt=[vt_1, vt_2, vt_3]$

A continuación, se transforma cada *eigenvector* en un vector unitario y se comparan utilizando el producto escalar; si dos vectores son paralelos (es decir, el ángulo que forman es de 0 grados), el valor de su producto escalar es igual al producto de sus

módulos, ya que el coseno del ángulo que forman es 1 (ver Fórmula 36).

$$A \cdot B = |A||B|\cos\Theta \qquad \Theta = 1 \Rightarrow \cos\Theta = 1$$
$$A \cdot B = |A||B| \tag{36}$$

Por lo tanto, dos modos normales expresan el mismo movimiento en la proteína si el producto de los módulos de cada uno de sus *eigenvectors* tiene un valor cercano a 1.

En la cadena y el modo del ejemplo anterior, el proceso de comparación quedaría de la siguiente forma:

1. <u>Normalización</u>

   Modo normal en coordenadas cartesianas (vo_i es el vector original y $\widehat{vo}_i$ es el normalizado).

$vo_1 = [0.014375, -0.022887, 0.013381]$ $\quad \hat{vo}_1 = [0.47665586, -0.75890245, 0.44369614]$

$vo_2 = [0.012958, -0.020079, 0.012157]$ $\quad \hat{vo}_2 = [0.483296, -0.74888875, 0.45342101]$

$vo_3 = [0.004634, -0.012146, 0.013471]$ $\quad \hat{vo}_3 = [0.24753259, -0.64879821, 0.71957522]$

   Modo normal transformado del cálculo en internas (vt_i es el vector original y $\widehat{vt}_i$ es el normalizado).

$vt_1 = [0.001764, -0.002802, 0.001654]$ $\quad \hat{vt}_1 = [0.4766068, -0.7570591, 0.44688642]$

$vt_2 = [0.001519, -0.002451, 0.001564]$ $\quad \hat{vt}_2 = [0.46305686, -0.74717075, 0.47677481]$

$vt_3 = [0.001519, -0.002451, 0.001564]$ $\quad \hat{vt}_3 = [0.43386897, -0.74660074, 0.50432633]$

2. <u>Producto escalar</u>

   $|vo_1||vt_1| = 0.999993210898$

   $|vo_2||vt_2| = 0.99952101301$

   $|vo_3||vt_3| = 0.954690665873$

   En este caso, todos los productos escalares de los vectores comparados tienen un resultado muy próximo a 1, así pues, ambos modos son equivalentes y el modo normal calculado en coordenadas internas es correcto.

Se han usado cinco proteínas distintas en el proceso de validación. Las proteínas se representan en archivos de texto en un formato estándar llamado PDB y se han descargado del Protein Data Bank [31], un repositorio de acceso libre de estructuras atómicas de proteínas.

Se han seleccionado las proteínas **4AKE**, **1SU4**, **1GGG**, **1EX6** y **1DDT** en las cuales se producen cambios conformacionales amplios. Antes de iniciar el proceso, se han eliminado de cada proteína las cadenas duplicadas y los átomos que no forman parte de la cadena principal.

Para todas ellas se ha calculado un conjunto de 20 modos normales usando el algoritmo en coordenadas internas y se han transformado los modos resultantes (que representan amplitudes de giros) a coordenadas cartesianas (representando direcciones de traslación).

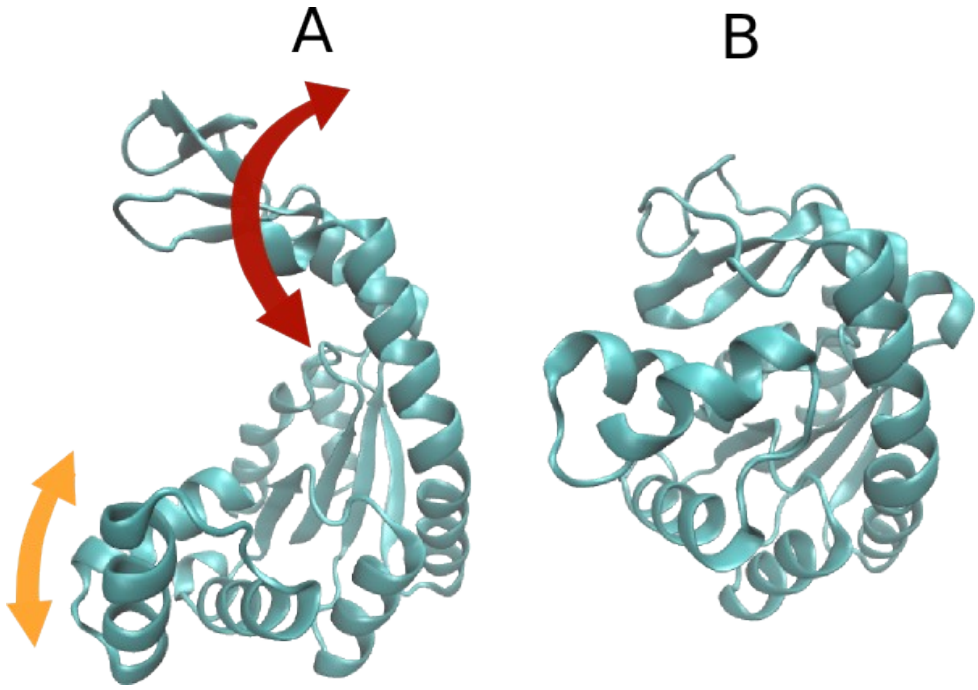


Figura 55 *Representación del primer modo normal en la proteína* ***4AKE****. En la imagen* ***A*** *se puede ver la cadena en la posición inicial abierta. En el movimiento de la cadena se diferencian dos dominios que se acercan y se alejan entre ellos hasta llegar a la posición final* ***B*** *siguiendo la trayectoria que marcan las flechas de la figura.*

Se han considerado equivalentes aquellos modos cuya media de los valores del producto escalar sea mayor que 0.9 o menor que -0.9 (180º, vectores en sentido contrario). Puesto que sólo son importantes la dirección y la magnitud relativa de los vectores, el sentido no es relevante.

Para las proteínas elegidas los resultados han sido los siguientes:

4AKE	
Modo	**Media**
1	0.979
2	-0.922
3	0.814
4	0.662
5	-0.759

1SU4	
Modo	**Media**
1	-0.994
2	0.884
3	-0.797
4	0.322
5	-0.736

1EX6	
Modo	**Media**
1	0.842
2	0.820
3	0.779
4	0.386
5	-0.425

1GGG	
Modo	**Media**
1	-0.875
2	-0.806
3	0.961
4	0.761
5	0.801

1DDT	
Modo	**Media**
1	-0.119
2	-0.227
3	0.940
4	0.879
5	0.940

Tabla 1 *Para cada proteína, en la columna izquierda se indica el modo y en la derecha la media del producto escalar. Se puede observar que los modos más coincidentes son normalmente los primeros, es decir, los que corresponden a las frecuencias más bajas y, por lo tanto, los movimientos más amplios y relevantes (son los que proporcionan más información sobre la función de la proteína).*

Como se puede observar en las tablas no todos los modos normales se pueden considerar equivalentes. Para entender las causas se van a comparar las tendencias de movimiento mediante unos gráficos y para cada residuo se analizará:

- Que las direcciones de los vectores de desplazamiento sean similares. En los gráficos se mostrará el producto escalar que toma valores entre 1 y -1.
- Que las magnitudes de estos desplazamientos sean parecidas. Para visualizarlas en las gráficas se han normalizado las tripletas de traslación de cada residuo: se han calculado las normas de todos los vectores y se han dividido entre la norma del vector más grande, normalizándolos de este modo en el rango [0, 1].

En color verde se muestra la tendencia de movimiento (magnitud relativa) de los vectores en coordenadas cartesianas, en rojo la de los vectores en coordenadas internas transformados y en azul el resultado del producto escalar de ambos vectores.

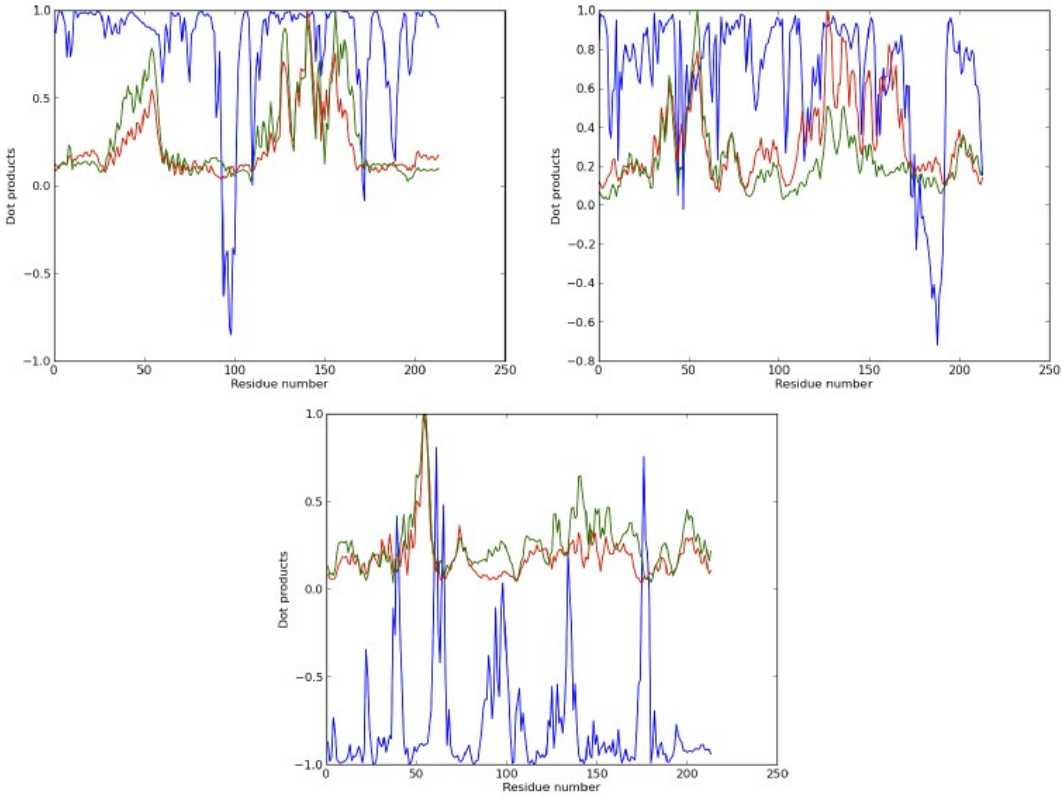


Figura 56 *4AKE: gráficos de los modos 3, 4 y 5*

En los gráficos de los modos 3 y 4 de la proteína 4AKE se observa que las magnitudes de los *eigenvectors* siguen un patrón muy parecido en la mayoría de residuos. En las zonas donde el valor del producto escalar (línea azul) es muy cercano a -1 (alrededor del residuo 100 en el modo 3 y del 190 en el modo 4) se aprecia que las tendencias de las normas y la dirección de los vectores se invierten.

En el modo 5, las partes con tendencias más parecidas se encuentran en los primeros 60 residuos y en los situados alrededor del 150. Hay zonas menos coincidentes especialmente donde el producto escalar tiene un valor cercano a 0 (residuos cerca del 100).

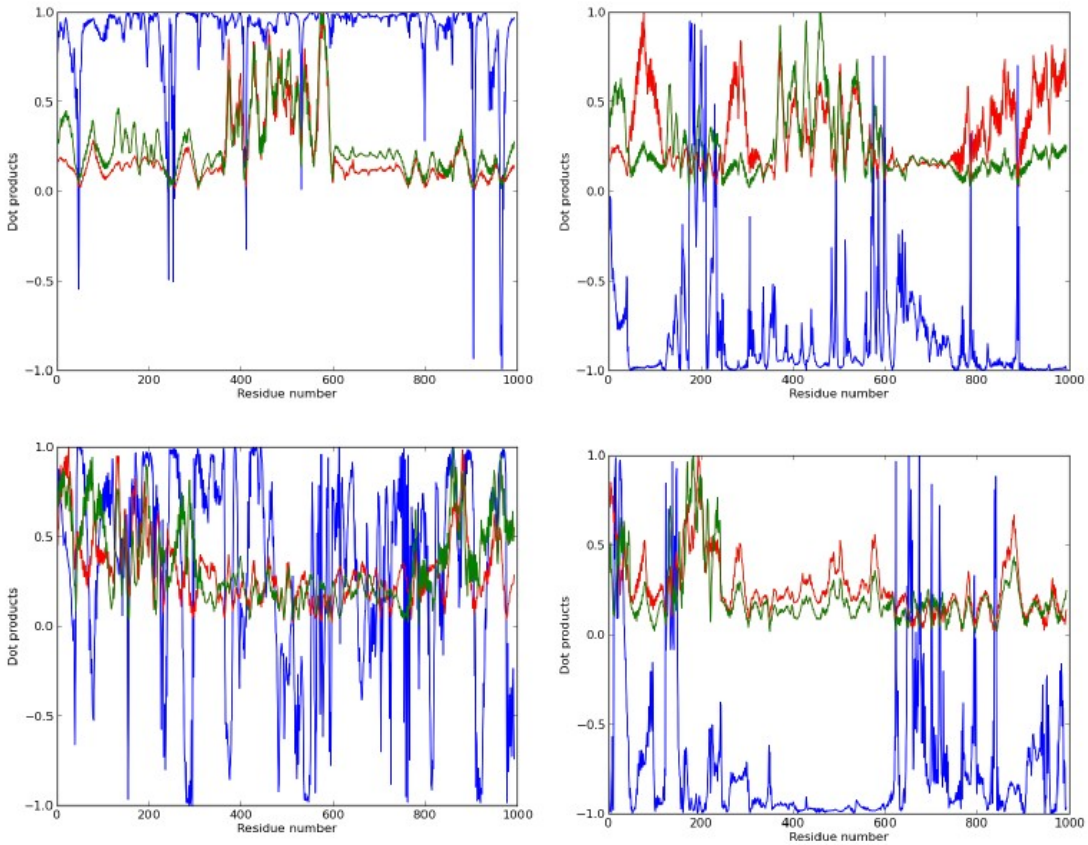


Figura 57 *1SU4: gráficos de los modos 2, 3, 4 y 5*

En los modos 2, 3 y 5 se observa un tendencia muy similar en toda la cadena, sobre todo en la zona entre los residuos 400 y 600 donde el producto escalar se mantiene bastante constante (excepto algunas oscilaciones puntuales).

El modo 4 tiene un valor de media muy bajo (0.322), a lo largo de toda la proteína las tendencias están invertidas y el valor del producto escalar oscila constantemente entre 1 y -1.

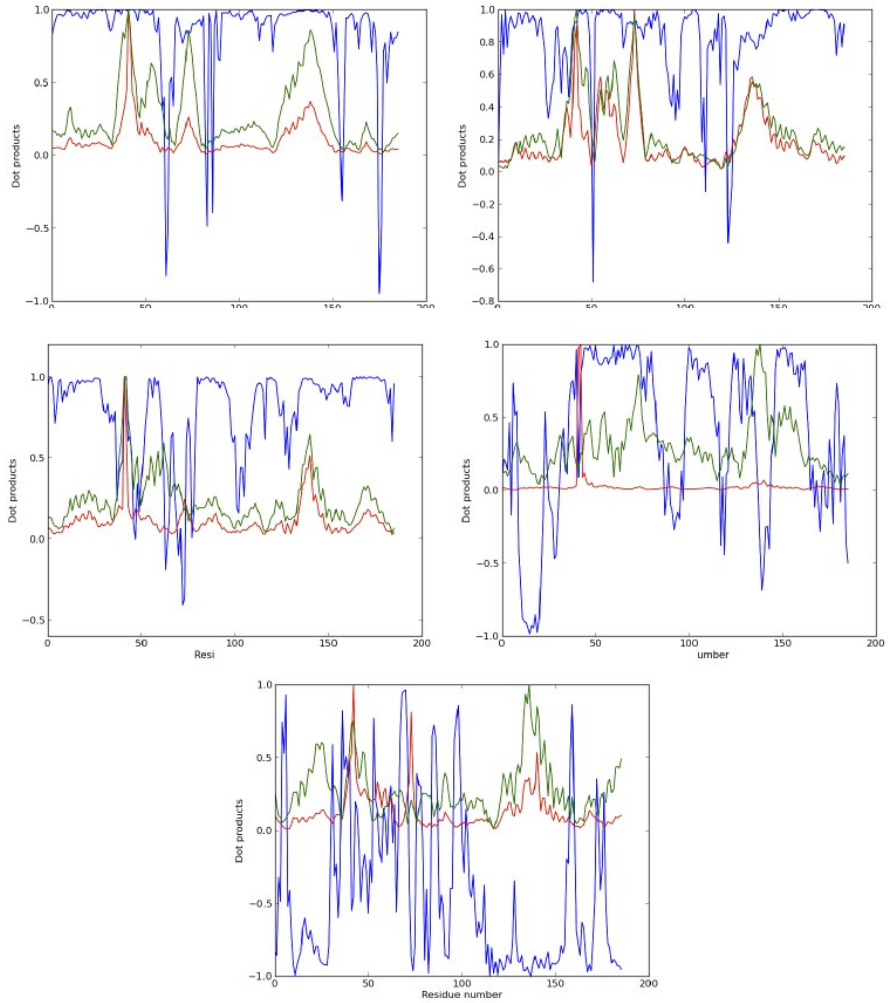


Figura 58 *1EX6: gráficos de los modos 1, 2, 3, 4 y 5*

En este caso ninguna de las medias de los cinco modos es superior a 0.9. Los tres primeros son los que tienen medias más altas y se puede ver en el gráfico que son los más similares a pesar de que hay algunas diferencias de magnitud. Alrededor del residuo 50, el valor de producto escalar es cercano a -1 y, por tanto, las tendencias se invierten.

Los modos 4 y 5 tienen las medias más bajas y especialmente en el modo 4 se observa como las magnitudes son muy diferentes a lo largo de toda la cadena.

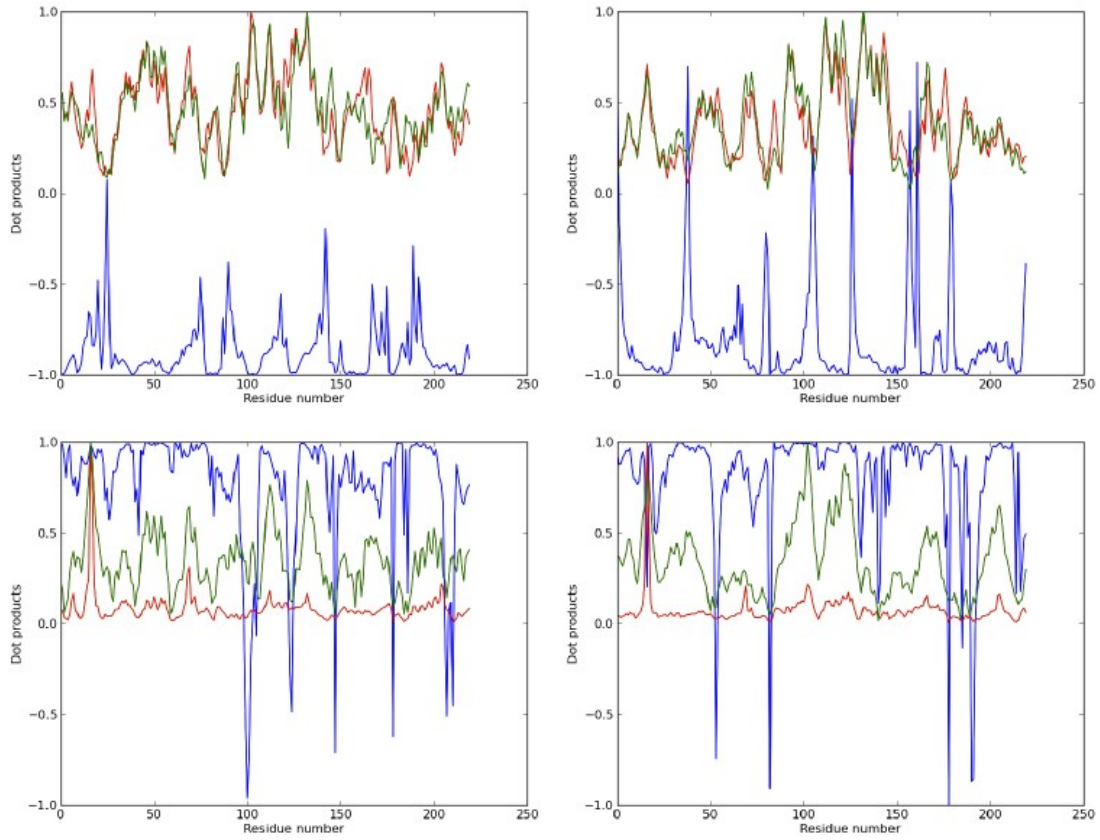


***Figura 59**: 1GGG: gráficos de los modos 1, 2, 4, 5*

En los cuatro modos de la proteína 1GGG el producto escalar se mantiene bastante constante en valores cercanos a -1 (en los dos primeros modos) y a 1 (en los dos últimos), por eso sus medias son altas aunque no lo suficiente para considerarlos totalmente equivalentes. En los dos primeros gráficos se ve una coincidencia casi total en la magnitud de todos los residuos. En los modos 4 y 5 la tendencia es parecida, pero las magnitudes en coordenadas cartesianas (línea verde) son mucho mayores.

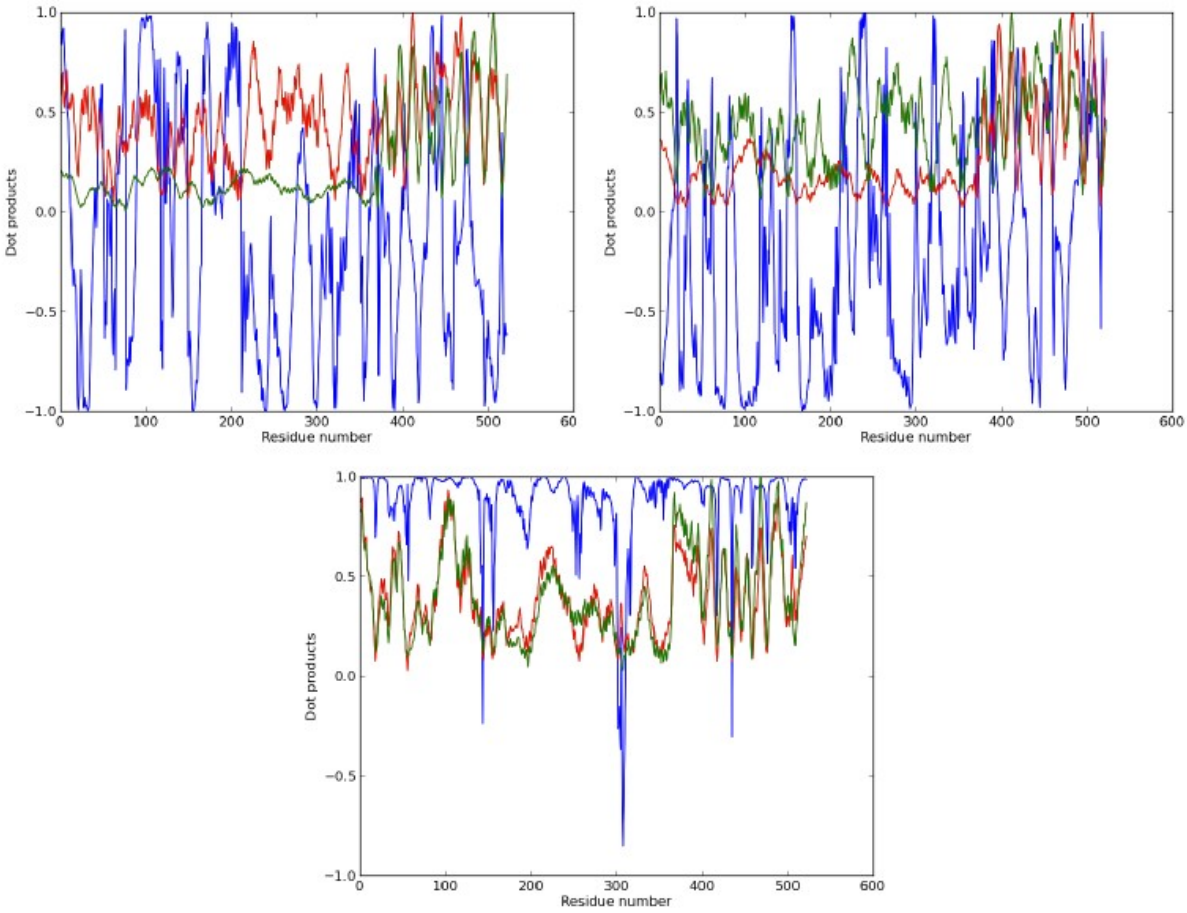


***Figura 60**: 1DDT: gráficos de los modos 1, 2 y 4*

En 1DDT, las medias de los productos escalares de los modos 1 y 2 tienen valores muy bajos (-0.119 y -0.227 respectivamente) y esto queda reflejado claramente en los gráficos. Se observan tendencias opuestas y una oscilación continua del valor del producto escalar entre 1 y -1. Por otro lado, la media del modo 4, aunque no llega a 0.9 tiene un valor muy alto (0.879) y se puede constatar en el gráfico que las magnitudes de ambos modos están casi totalmente solapadas.

A pesar del valor de la media, se puede ver como, en general, las tendencias de los *eigenvectors* son muy similares en la mayoría de casos y, por tanto, el movimiento que provocarán en la proteína será muy parecido.

Tres posibles causas para explicar la falta de coincidencia entre los modos son:

- Los modelos atómicos aplicados en coordenadas cartesianas e internas son diferentes. Los *eigenvectors* en cartesianas se calculan para cada Cα, en internas para las unidades compuestas por varios átomos y para la comparación se escogen tan solo los asociados a los Cα.
- En el modelo en coordenadas internas se tienen en cuenta las masas y el centro de masas de cada unidad. En coordenadas cartesianas no se diferencia la masa para cada nodo centrado en el Cα.
- Finalmente, los algoritmos que se usan para conseguir los eigenvectors no garantizan el orden de los modos resultantes. Esto significa que el i-ésimo modo obtenido con un algoritmo no es forzosamente el mismo que el modo i-ésimo obtenido con el otro. Por ejemplo, los modos 1 y 2 de la proteína 1DDT parecen estar invertidos: las magnitudes en internas del primero coinciden con las de cartesianas del segundo y viceversa.

  Es posible que algunos de los modos que son muy diferentes tengan correspondencia con otros modos fuera del rango del 1 al 5 (por ejemplo, el modo 4 de 1SU4).

4. Rendimiento

La hipótesis inicial era que la implementación del ANM en coordenadas internas mejoraría el rendimiento del paso de PELE (especialmente por el ahorro que conlleva no hacer la minimización en la etapa del ANM). Para comprobarlo se ha realizado una ejecución de PELE utilizando ambos métodos sobre una proteína tipo: 4AKE. No se ha hecho un estudio del rendimiento en mayor profundidad (repetición de la prueba, uso de distintas proteínas, uso de herramientas de *profiling*) porque la eficiencia no era uno de los objetivos de este proyecto.

El *script* de PELE que se ha usado tiene las siguientes características:

- Se calculan los eigenvectors una sola vez
- Se realizan 10 pasos de PELE

Los resultados han sido:

Tipo de algoritmo	**Tiempo**
ANM en coordenadas cartesianas	9m 36s
ANM en coordenadas internas	2m 2s

Tabla 2 *Tiempos de ejecución de PELE con los dos algoritmos*

Se puede observar en la tabla 2 que el rendimiento del ANM en coordenadas internas es unas cuatro veces más rápido que el ANM en coordenadas cartesianas lo cual supone una notable mejora.

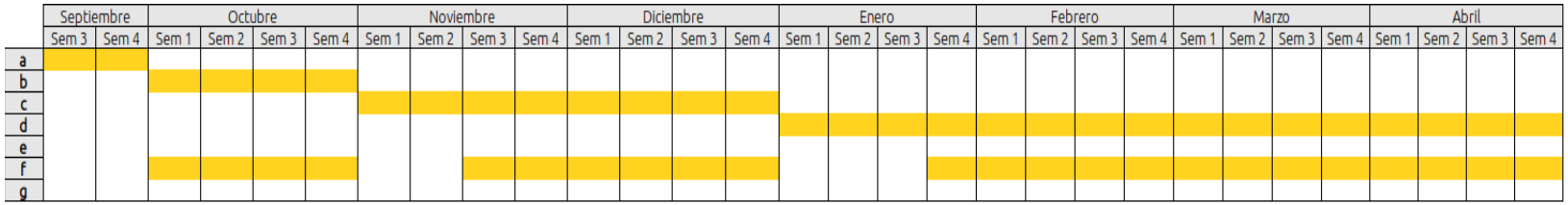
5. Planificación y costes

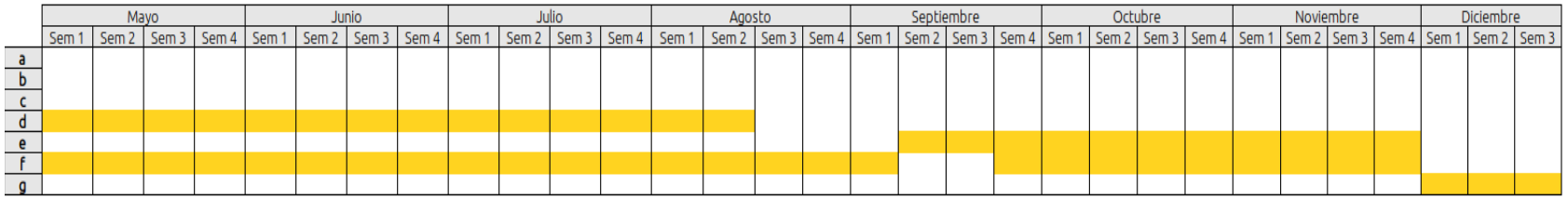
5.1. Planificación

En este apartado se detalla la planificación seguida durante el desarrollo del proyecto mediante un diagrama de Gantt.

El trabajo realizado puede subdividirse en los siguientes apartados:

a. Estudio inicial sobre la materia (aminoácido, proteínas, análisis vibracional, etc.)
b. Estudio sobre el funcionamiento del ANM y PELE
c. Desarrollo y *tests* de las funciones para la minimización
d. Implementación y *tests* de las funciones del ANM en coordenadas internas
e. Validación del modelo desarrollado
f. Documentación
g. Preparación de la presentación


Septiembre Octubre Noviembre Diciembre Enero Febrero Marzo Abril
Sem 3 Sem 4 Sem 1 Sem 2 Sem 3 Sem 4 Sem 1 Sem 2 Sem 3 Sem 4 Sem 1 Sem 2 Sem 3 Sem 4 Sem 1 Sem 2 Sem 3 Sem 4 Sem 1 Sem 2 Sem 3 Sem 4 Sem 1 Sem 2 Sem 3 Sem 4 Sem 1 Sem 2 Sem 3 Sem 4
a
b
c
d
e
f
g


Mayo Junio Julio Agosto Septiembre Octubre Noviembre Diciembre
Sem 1 Sem 2 Sem 3 Sem 4 Sem 1 Sem 2 Sem 3 Sem 4 Sem 1 Sem 2 Sem 3 Sem 4 Sem 1 Sem 2 Sem 3 Sem 4 Sem 1 Sem 2 Sem 3 Sem 4 Sem 1 Sem 2 Sem 3 Sem 4 Sem 1 Sem 2 Sem 3 Sem 4 Sem 1 Sem 2 Sem 3 Sem 4 Sem 1 Sem 2 Sem 3
a
b
c
d
e
f
g

5.2. Costes

Para calcular el valor total de un proyecto como este se han de considerar los costes de las personas involucradas, del espacio utilizado y del material necesario para llevarlo a cabo.

En el desarrollo del proyecto han participado un jefe de proyecto con un salario de 50€/hora y un programador con un salario de 30€/hora:

Recurso	Precio	Horas	Total
Jefe de proyecto	50,00 €/hora	60	3.000 €
Programador	30 €/hora	730	21.900 €
		Total:	24.900 €

Tabla 3 *Costes de personal*

El material empleado ha sido: utensilios de oficina (papel, bolígrafos, etc) y un ordenador portátil con un precio de 850€ con una vida útil de 2 años. Por tanto, el precio de 15 meses de uso ha sido de 531€.

Concepto	Precio
Ordenador portátil (15 meses)	531 €
Costes de material	40 €
Total:	571 €

Tabla 4 *Costes de material*

El espacio utilizado ha sido una oficina de unos $8m^2$ que tiene costes asociados como son el alquiler y el consumo de electricidad:

Concepto	Precio	Meses	Total
Alquiler	80 €/mes	15	1.200 €
Electricidad	35 €/mes	15	525 €
		Total:	1.725 €

Tabla 5 *Costes de espacio*

El coste total de todo el proyecto será:

Concepto	**Precio**
Costes de personal	24.900 €
Costes de material	571 €
Costes de oficina	1.725 €
Total:	27.196 €

Tabla 6 *Coste total*

6. Conclusiones

6.1. Objetivos cubiertos

El principal objetivo de este proyecto final de carrera ha sido la implementación del modelo ANM en coordenadas internas para integrarlo en el software PELE y así complementar la actual implementación en coordenadas internas.

Para ello se han tenido que cumplir una serie de objetivos:

- Introducción a la materia: se ha estudiado la base teórica sobre biología necesaria para poder entender y desarrollar el proyecto (aminoácidos, proteínas, dinámica molecular, funcionamiento de PELE, modelos ANM, etc.).
- Estudio del modelo ANM en PELE: se ha estudiado la implementación actual en coordenadas internas de PELE. También se han documentado sus diferentes fases para poder compararlas con la nueva implementación realizada.
- Implementación del ANM en coordenadas internas: se han implementado e integrado en PELE las clases y funciones necesarias para calcular el análisis vibracional en coordenadas internas así como los *tests* para asegurar su correcto funcionamiento.

  Por otra parte, se han desarrollado funciones para el cálculo de la energía potencial, el gradiente y el Hessiano dados los cuatro átomos que forman un ángulo diedro. Estas funciones se utilizan durante la minimización de la proteína que se produce después de la perturbación local de PELE.

- Validación del modelo: se han llevado a cabo una serie de pruebas para comprobar que los modos normales calculados con la nueva implementación del ANM son correctos, es decir, son equivalentes a los originales en coordenadas cartesianas.
- Memoria del proyecto: se ha documentado todo el trabajo realizado en esta memoria.

6.2. Discusión y mejoras futuras

Una posible línea de trabajo futura sería la mejora de rendimiento del código implementado. El desarrollo actual se ha hecho de forma que el código sea fácilmente entendible y refleje lo más fielmente posible las ecuaciones que definen el modelo, sin hacer hincapié en la eficiencia.

Por otra parte, durante el proceso de validación se ha podido constatar que no se ha conseguido una equivalencia total entre los modos originales en coordenadas cartesianas y los nuevos calculados en internas, además de que los modos más coincidentes han sido los de frecuencias más bajas. Sería muy interesante continuar con este estudio ampliando la muestra de proteínas.

Finalmente, se debería probar el método en simulaciones largas y encontrar metodologías para contrastar la validez de dichas simulaciones.

6.3. Valoración personal

A nivel personal la realización de este proyecto final de carrera ha supuesto la oportunidad de conocer un campo de aplicación de la ingeniería informática como es la bioinformática y de adquirir una serie de conocimientos que de otra forma no hubiera podido adquirir. También he podido ver como se estructura un proyecto de gran envergadura con una aplicación práctica como PELE y desarrollado en un lenguaje de programación que no había utilizado con anterioridad como es C++.

Ha significado un importante reto para mi ya que carecía de conocimientos previos relacionados con biología y, en concreto, con el análisis vibracional de proteínas.

Este hecho juntamente con la compatibilización del proyecto con mi trabajo prácticamente a jornada completa han sido las mayores dificultades que he encontrado.

7. Bibliografía

[1] http://es.wikipedia.org/wiki/Polipéptido (15/10/2013)

[2] http://en.wikipedia.org/wiki/Molecular_dynamics (19/10/2013)

[3] *Protein folding and disease:* http://www.nature.com/horizon/proteinfolding/background.html (20/10/2013)

[4] *FoldIt:* http://fold.it/portal/info/about (20/10/2013)

[5] *Protein folding:* http://en.wikipedia.org/wiki/Protein_folding (20/10/2013)

[6] Yaoquan Tu, *Molecular Dynamics Simulations:* http://www.theochem.kth.se/courses/BEST_Lecture_MD.pdf (12/11/2013)

[7] Elmar Krieger, *Molecular Dynamics and Force Fields:* http://swift.cmbi.ru.nl/teach/B2/HTML/MDFF.html (15/11/2013)

[8] Force fields: https://en.wikipedia.org/wiki/Force_field_%28chemistry%29 (15/11/2013)

[9] *Van der Waals interactions*: http://en.wikibooks.org/wiki/Structural_Biochemistry/Chemical_Bonding/Van_der_Waals_interaction (16/11/2013)

[10] *Potential Energy Function:* http://www.ch.embnet.org/MD_tutorial/pages/MD.Part2.html (28/11/2013)

[11] *ANM:* https://en.wikipedia.org/wiki/Anisotropic_Network_Model (4/12/2013)

[12] Jenelle Bray, *NMA Calculation of the Natural Motions of Proteins*, http://biomedicalcomputationreview.org/content/normal-mode-analysis-calculation-natural-motions-proteins (6/12/2013)

[13] http://en.wikipedia.org/wiki/Dihedral_angle (6/12/2013)

[14] http://math.stackexchange.com/questions/47059/how-do-i-calculate-a-dihedral-angle-given-cartesian-coordinates (6/12/2013)

[15] José Ramón López Blanco, *Nuevos métodos para el ajuste flexible de estructuras macromoleculares a distintas resoluciones empleando modos normales de vibración en coordenadas internas*

[16] Kenneth W. Borrelli, Andreas Vitalis, Raul Alcantara y Victor Guallar, *Protein Energy Landscape Exploration. A novel Monte Carlo Based Technique.*

[17] Furio Ercolessi, *A molecular dynamics primer*

[18] María Inés Nicolás Vázquez, Eric Marín Chiñas, Francisco Miguel Castro Martínez, René Miranda Ruvalcaba, *Algunos Aspectos Basicos de la Quimica Computacional*

[19] Xiong Jiao, Shan Chang, Lifeng Yang, Meiwen An and Weiyi Chen, *Vinculin Motion Modes Analysis with Elastic Network Model*

[20] José Ramón Lopéz-Blanco, Pablo Chacón, *iMODFIT: Efficient and robust flexible fitting based on vibrational analysis in internal coordinates*

[21] Carlos Gómez Moreno, *Estructura de proteínas*

[22] Mingyang Lu, Billy Poon and Jianpeng Ma, *A New Method for Coarse-Grained Elastic Normal-Mode Analysis*

[23] Qiang Cui and Ivet Bahar, *Normal mode analysis, theory and applications to biological and chemical systems*

[24] A. R. Atilgan, S. R. Durell, R. L. Jernigan, M. C. Demirel, O. Keskin and I. Bahar, *Anisotropy of Fluctuation Dynamics of Proteins with an Elastic Network Model*

[25] Jianpeng Ma, *New Advances in Normal Mode Analysis of Supermolecular*

Complexes and Applications to Structural Refinement

[26] Tod D. Romo and Alan Grossfield, *Validating and improving elastic network models with molecular dynamics simulations*

[27] Alexander D. MacKerell, Jr., *Protein Force Fields*

[28] Timothy R. Lezon, Indira H. Shrivastava, Zheng Yang and Ivet Bahar, *Elastic Network Models For Biomolecular Dynamics: Theory and Application to Membrane Proteins and Viruses*

[29] Moon K. Kim, Yunho Jang, and Jay I. Jeong, *Using Harmonic Analysis and Optimization to Study Macromolecular Dynamics*

[30] Werner Braun, Shuzo Yoshioki, Nobuhiro Go, *Formulation of Static and Dynamic Conformational Energy Analysis of Biopolymer Systems Consisting of Two or More Molecules*

[31] *RCSB Protein Data Bank: http://www.rcsb.org*

[32] Vicky Choi, *On Updating Torsion Angles of Molecular Conformations*

I. Anexo: archivos entregados

Junto con la memoria del proyecto se entrega un archivo comprimido que contiene todo el código fuente desarrollado.

PELE no es un software de código abierto, por lo tanto sólo se pueden distribuir las partes del código implementadas en este proyecto y las que están relacionadas y ayudan a contextualizarlo.

El contenido del archivo es el siguiente:

- **maxima**: contiene los *scripts* de *wxMaxima* que generan el código de las funciones utilizadas en la minimización.
- **validacion**: contiene los *scripts* en *Python* utilizados para el proceso de validación y sus juegos de pruebas.
- **pele_src**: contiene el código de PELE, los *tests* y sus juegos de pruebas.

  Las carpetas que tienen el código del proyecto son:

  - **ANM/ModesCalculator/Internals/MatrixCalculationFunctions**: implementación de la funciones del ANM en coordenadas internas.
  - **Energy/PotentialConstraint/ConstraintTerms/HarmonicDihedral Constraint**: implementación de las funciones usadas en la minimización.