Análisis de la relación entre cambios en la isomería de las cadenas R y la constante de disociación en la formación de complejos proteicos.

Francisco Javier Lobo Cabrera

Máster en Bioinformática y Bioestadística

Programación en Bioinformática

Brian Jiménez García

24 de Mayo de 2017

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Análisis de la relación entre cambios en la isomería de las cadenas R y la constante de disociación en la formación de complejo proteicos.*
Nombre del autor:	*Francisco Javier Lobo Cabrera*
Nombre del consultor/a:	*Brian Jiménez García*
Nombre del PRA:	
Fecha de entrega (mm/aaaa):	05/2017
Titulación::	*Máster en Bioinformática y Bioestadística*
Área del Trabajo Final:	*Programación en Bioinformática*
Idioma del trabajo:	*Castellano*
Palabras clave	*Rotámeros, termodinámica, proteínas*

Resumen del Trabajo*:*

Este trabajo ha investigado el grado de relación entre dos variables biológicas.

La primera de las variables está relacionada con el patrón de isomerÍa en proteínas que forman complejos proteicos. Más concretamente, se examinó la isomeria conformacional de las cadenas R de los aminoácidos que cambian de su posición como consecuencia de formarse el complejo proteico. Para cada rotámero se calculó su desviación conforme a un sistema de referencia de baja energía y luego se llevó a cabo un sumatorio de dichas desviaciones para todos los rotameros. Finalmente, se halló la diferencia entre los sumatorios de las proteínas del complejo en estado separado y unido, parámetro al que se denominó Δλ.

La otra de las variables estudiadas fue la constante termodinámica de disociación del complejo proteico. Este tipo de información está disponible en la base de datos PDBbind.

El trabajo consistió en hallar coincidencias entre las bases de datos estructural y termodinámica, calcular el parámetro Δλ de cada coincidencia –paso más complejo– y emparejarlo con la constante de disociación asociada. Era de esperar que si la formación del complejo implicaba un “relajamiento” termodinámico de los rotámeros de las cadenas R esto tendría un efecto en la constante de disociación termodinámica. Pruebas estadísticas de calidad de simulaciones estructurales de complejos proteicos podrían realizarse en base a esa relación. Por otro lado, una independencia entre las variables implicaría una baja acción de los cambios de isomería conformacional de las cadenas R en la termodinámica de formación del complejo.

Abstract (in English, 250 words or less):

The goal of this project was to determine the degree of relationship between two biological variables.

The first of the studied variables is associated with isomeric patterns in proteins forming part of protein complexes. Specifically, it is related to R-chain conformational isomerism in amino acids that change their position when the protein complex is formed. Every rotamer's deviation from a low energy reference system was calculated and then each of the deviations was added up. Finally, such sumatories when proteins were unbounded were substracted to those when proteins were bounded in the complex. This difference was called the Δλ parameter.

The other variable under study was the protein complex thermodynamic dissociation constant. This type of information is available at the PDBbind database.

The project dealt with finding coincidences between the structural and thermodynamic databases employed, calculating the Δλ parameter for each coincidence –most complex step– and matching that with the related dissociation constant. Thermodynamic "relaxation" of the rotamers in the formation of the protein complex would presumably influence the thermodynamic dissociation constant. Statistical tests on the quality of structural simulations of protein complexes may be carried out on the basis of such relationship. On the other hand, independency between the variables would show that R-chain isomeric changes have little effect on the thermodynamics of protein complex formation.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

La isomería conformacional de las cadenas R proteicas es un aspecto cada vez más considerado en biología estructural. De particular interés es conocer los ángulos rotacionales medios de las cadenas R de los residuos implicados en la interacción proteína-proteína. [1] Al mismo tiempo, ciertos complejos proteicos no han podido ser cristalografiados ni resueltos mediante Resonancia Magnética Nuclear u otras técnicas, por lo que se recurre a simulaciones estructurales en esos casos. [2] No obstante, sólo hasta hace poco tiempo se está empezando a contar con la potencia de hardware y el software necesario para incluir a las cadenas R en dichas simulaciones. [3]

Simultáneamente, existen recursos experimentales relacionados con los complejos proteicos alternativos a los estudios de estructura. Un ejemplo son los análisis termodinámicos de formación de complejos proteicos. Estos estudios termodinámicos son más simples que aquellos de determinación de estructura. Además, actualmente ya existen bases de datos que recogen información de esta índole, como PINT Database [4] ó PDBbind. [5]

Este TFM estudió la posibilidad de cruzar las informaciones estructural y termodinámica disponibles de complejos proteicos. El objetivo fue estudiar la dependencia/independencia entre ambos datos, en particular en lo relacionado a la isomería conformacional de las cadenas R. En caso de hallar dependencia se podría generar un modelo que relacionara ambos tipos de información, y con él testear simulaciones estructurales de complejos proteicos. Este es el interés principal de este TFM. Además, se creará conocimiento de interés para el campo de simulaciones estructurales de proteínas y termodinámica.

1.2 Objetivos del Trabajo

A continuación, se exponen los objetivos principales del trabajo junto con los subobjectivos específicos:

a) Creación de software que produzca un modelo lineal que relacione la variable estructural Δλ –explicada más adelante– con las constantes de disociación termodinámica (Kd) de los complejos proteicos.

a1) Creación de software que genere, a partir de una base de datos estructural y otra termodinámica, una lista de pares Δλ-Kd.
*a*2) Creación de software que produzca un modelo lineal que relacione Δλ con Kd a partir de la lista formada por el objetivo a1).

b) Creación de software que a partir de un par Δλ-Kd evalúe su conformidad de acuerdo al modelo generado por el objetivo a).

c) Creación de software que aglutine los objetivos a) y b) en una única interfaz de interacción con el usuario.

1.3 Enfoque y método seguido

Tal cómo se discutió en la PAC1 de la asignatura, entre las posibles estrategias a seguir se encontraron:

a) Creación del software para los distintos objetivos desde cero. Esto permitiría desarrollar un programa más eficiente, por tener un diseño más específico. Como contrapartida, sería más probable la existencia de errores informáticos en el código. Otra desventaja es el mayor tiempo de desarrollo de los software diseñados totalmente a medida.

b) Creación del software utilizando en la medida de lo posible softwares ya desarrollados. Por otro lado, las piezas de software creadas o empleadas pueden estar escritos en lenguajes de programación diferentes. Mediante esta esta estrategia, la probabilidad de errores en el código es menor. Asimismo, el tiempo de desarrollo es del software es menor. Como desventaja, la eficiencia final puede no ser la más óptima. Además, el equipo informático del usuario deberá contar con todas las dependencias que el software necesita para su funcionamiento. Esto redunda en una menor portabilidad del programa

La opción escogida fue la b). Dada la complejidad del proyecto, se ha preferido la fiabilidad sobre la eficiencia y portabilidad. Otro motivo para dicha elección ha sido acelerar el tiempo de desarrollo del software. De este modo se ha permitido alcanzar los objetivos en los plazos deseados.

1.4 Planificación del Trabajo

Como recursos para realizar el TFM se ha contado con cinco horas diarias de trabajo disponibles. En cuanto a recursos materiales, se ha utilizado un ordenador personal con sistema operativo Ubuntu 16.04.

En primer lugar, se realizó la PAC "0", donde se llevó a cabo una presentación personal y se estudió el Plan Docente de la asignatura.

A continuación, como parte de la PAC1 se creó una lista de las tareas a realizar hasta finalizar el proyecto (Tabla 1).

Objetivo	Tarea	Descripción	Duración
1.1	1	Aprendizaje de Python de interés para los objetivos 1.1. 1.3. 2.1. 2.3 y 3.1.	4d 4h
1.1	2	Pruebas con el framework Lightdock.	1d 1h
1.1	3.1	Creación de algoritmo de cálculo de λ de un aminoácido.	3d 1h
1.1	3.2	Pruebas del algoritmo creado de cálculo de λ de un aminoácido.	1d 1h
1.2	4	Creación de código que extraiga la información de las bases de datos termodinámicas.	1d 3h
1.2	5	Creación de código que seleccione coincidencias entre las bases de datos estructurales y termodinámicas.	1d 3h
1.2	6	Creación de código en R que genere el modelo entre Δλ y el parámetro termodinámico*. (*) Como parámetro termodinámico finalmente se eligió la constante de disociación (Kd) disponible en la base de datos PDBBind [2].	1d 3h
1.3	7.1	Aglutinación de todo el software del objetivo 1 en un mismo código.	1d 1h
1.3	7.2	Pruebas del código aglutinado de todo el objetivo 1.	1d
2.1	8.1	Creación de código que invocando al software del objetivo 1 calcule para un complejo su Δλ.	1d
2.1	8.2	Pruebas del código que invocando al software del objetivo 1 calcula Δλ.	2h
2.2	9	Creación de código en R para prueba de hipótesis con el modelo y datos del problema.	2d 2h
2.3	10.1	Aglutinación de todo el software del objetivo 2 en un mismo código.	2d
2.3	10.2	Pruebas del código aglutinado de todo el objetivo 2.	1d
3.1	11.1	Aglutinación de los softwares de los objetivos 1 y 2.	1d 2h
3.1	12	Creación de interfaz de interacción con el usuario para software unido de objetivos 1 y 2.	4h
3.1	13	Pruebas del programa completo.	1d
-	14	PAC2.	1d 2h
-	15.1	PAC3 (fase I).	3h
-	15.2	PAC3 (fase II).	3h
-	16.1	Memoria del TFM (fase I).	4d 2h
-	16.2	Memoria del TFM (fase II).	1d 3h
-	16.3	Memoria del TFM (fase II).	4d
-	17	Preparación de la defensa.	6d
-	18	Ensayo de la defensa pública.	3d
-	19	Defensa pública.	1d

Tabla 1. Lista de tareas contenidas en la PAC1.

La Figura 1 muestra el calendario –diagrama de Gantt– elaborado. La planificación tiene presente los días festivos 10 a 14 de Abril. En dichos días no hay avance en las horas de trabajo realizadas. Por su parte, los fines de semana tampoco figuran como días laborables.

La nomenclatura W de la Figura 1 hace referencia a los recursos de tiempo utilizados. Por ejemplo, W 20 significa que para dicha tarea se está dedicando un 20% del total del tiempo diario destinado al TFM. La ausencia de corchetes tras W indica una dedicación del 100% del tiempo diario al TFM a dicha tarea.

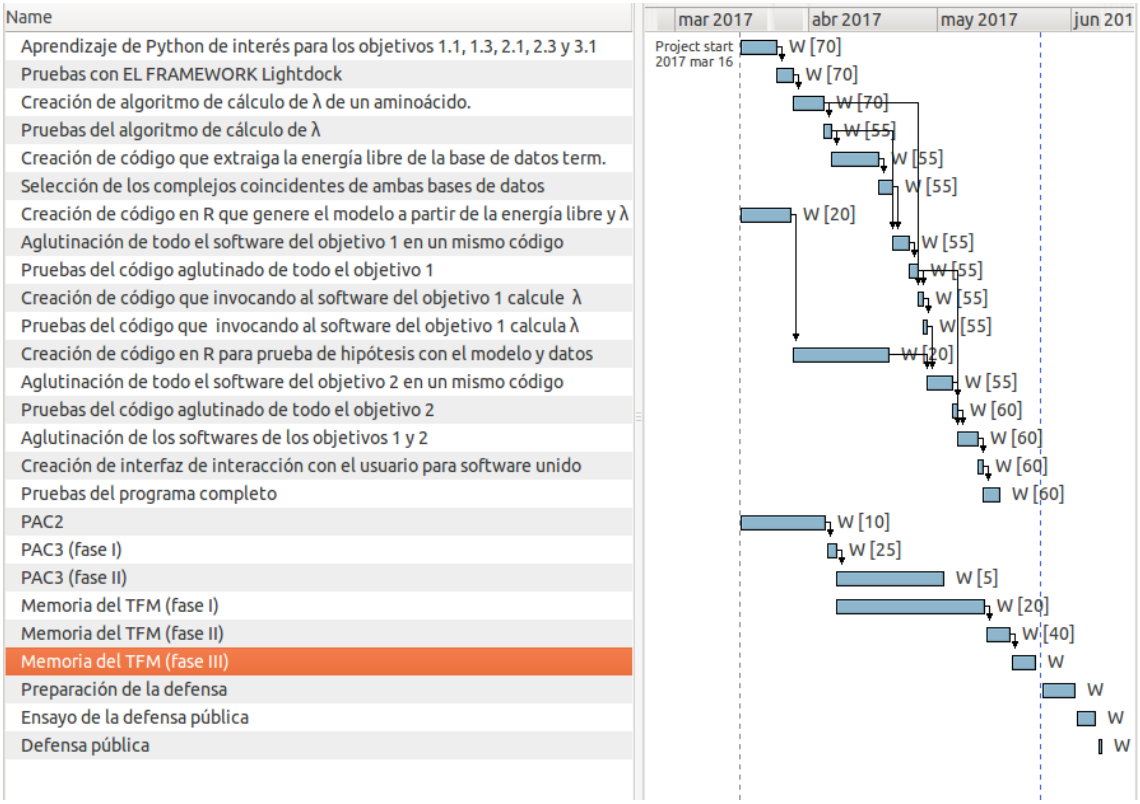

Figura 1. Diagrama de Gantt de las tareas a realizar.

1.5 Breve sumario de productos obtenidos

a) Software que a partir de una base de datos estructural y otra termodinámica genera un modelo lineal entre Δλ y Kd. La definición de Δλ y Kd se tratan en profundidad posteriormente en esta memoria.

b) Software capaz de ejecutar un test estadístico de calidad de simulaciones de complejos proteicos basado en el modelo de a).

c) Programa que integra los softwares de los apartados a) y b) en una única interfaz de interacción con el usuario.

d) Memoria explicativa (presente documento) del programa desarrollado y de su motivación.

Asimismo, existen otros documentos relacionados con la consecución del trabajo. Entre ellos se encuentran las PECs de presentación inicial, las de seguimiento del proyecto y las de presentación hacia el jurado.

1.6 Breve descripción de los capítulos de análisis informático

Los capítulos de análisis informático son los siguientes:

- Acerca de la isomería conformacional y las cadenas R aminoacídicas (capítulo 2.0):

Este capítulo consta de un introducción a los enlaces rotables. En particular, se analizan aquellos de los residuos proteicos en sus cadenas R .

- Definición de parámetros Δλ y Kd (capítulo 2.1):

En este capítulo se explica en profundidad a qué se denominó Δλ en este TFM. También se ofrece un breve resumen del significado de Kd.

- Motivación racional del modelo a estudiar (capítulo 2.2):

Una descripción de los motivos por los que se ha presupuesto una posible relación entre Δλ y Kd.

- Panorámica general del algoritmo de cálculo de los valores Δλ (capítulo 2.3):

Visualización de los pasos seguidos para el cálculo del Δλ de cada complejo proteico.

- Algoritmos para la creación del modelo y la prueba de hipótesis (capítulo 2.4):

Por un lado, en este apartado se describe la generación del modelo entre Δλ y Kd. Por otro lado, se explica el test de de calidad de simulaciones de complejos en base al modelo construido.

- Determinación de los ángulos chi estables (capítulo 2.5):

Explicación de cómo se determinaron los ángulos de rotación de estabilidad de referencia.

- Características de las bases de datos utilizadas (capítulo 2.6):

En este capítulo se describen las bases de datos estructurales y termodinámicas consideradas en el estudio. Se incluyen asimismo los argumentos por los que se escogió finalmente unas en lugar de otras.

- Diferentes lenguajes de programación empleados (capítulo 2.7):

Una perspectiva global de qué partes del software se han desarrollado en cada lenguaje de programación.

- Módulos externos en Python (capítulo 2.8):
Un resumen de qué código externo desarrollado en Python se ha integrado en el proyecto. En él se incluye fundamentalmente funciones ubicadas dentro del framework Lightdock. [6]

- Módulos creados en Python (capítulo 2.9):

Exposición de cada uno de los módulos de Python generados. Dentro de este capítulo se incluye la descripción de los módulos:

a) extraer_informacion_bases_de_datos.py (2.9.1).
b) alineamientos_estruc.py (2.9.2).
c) func_lambda.py (2.9.3).
d) primer_indice_res.py (2.9.4).
e) parte1.py (2.9.5).
f) parte2.py (2.9.6).
g) software_completo.py (2.9.7).
h) Software auxiliar: transformar_ficheros_sdf_ideal_a_pdb.py (2.9.8). *
i) Software auxiliar: calcular_chi_angles_equilibrio.py (2.9.9). *

- Módulos creados en R (capítulo 2.10):

Explicación de los módulos relacionados con la programación estadística:

a) codigo_en_R_para_analizar_alineamiento_estructural.R (2.10.1).
b) codigo_en_R_para_generar_modelo.R (2.10.2).
c) codigo_en_R_para_prueba_de_hipotesis_con_el_modelo_y datos_del_problema.R (2.10.3).

(*) Las piezas de software auxiliar fueron desarrolladas sólo con el propósito de generar el fichero de ángulos chi estables (angulos_chi_equilibrio). Una vez generado dicho fichero, no son de aplicación para el resto del programa.

- Determinación de los residuos que cambian de posición al formarse el complejo –residuos clave– (capítulo 2.11):

Descripción de la metodología de alineamientos estructurales para la obtención de los residuos *clave*.

- Resultados (capítulo 2.12):

Exposición de los resultados obtenidos. Clasificación en apartados:

a) Coincidencias de complejos entre las bases de datos estructural y termodinámica (capítulo 2.12.1).
b) Visualización del gráfico Kd VS Δλ (capítulo 2.12.2).
c) Análisis de dependencia entre las variables Kd y Δλ (capítulo 2.12.3).
d) Validez o no de la hipótesis de trabajo (capítulo 2.12.4).

2. Análisis informático

2.0 Acerca de la isomería conformacional y las cadenas R aminoacídicas

La isomería representa la relación entre moléculas con la misma fórmula molecular pero distinta fórmula lineal o estereoquímica. [7] Dentro de la isomería, la esteroisomería hace referencia a aquellos isómeros con una misma interconectividad entre los átomos pero diferente disposición espacial. [7]

En un nivel de clasificación más específico, los confórmeros son esteroisómeros que se pueden interconvertir sólo por rotaciones de enlaces simples. Supóngase un caso simple, como el etano. En esta molécula el enlace simple carbono-carbono origina que la misma molécula pueda tener diferentes conformaciones. Las dos posiciones más extremas constituyen las denominadas conformaciones alternada y eclipsada (Figura 2).

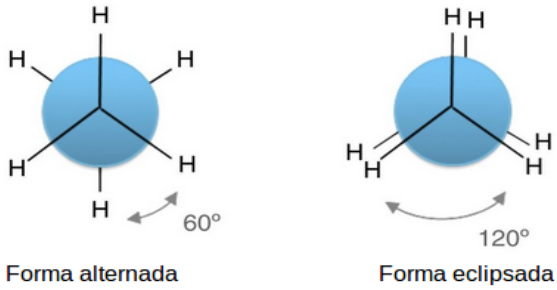

Figura 2. Conformaciones isoméricas extremas para el etano [8] (modificado).

Existe una continua interconversión entre las formas de un confórmero. Debido a la agitación térmica, los enlaces simples rotan y producen dichas interconversiones. Para el etano, las conformaciones alternada y eclipsada corresponden a las estructuras más y menos estable respectivamente. La raíz de ello es la repulsión electrostática entre las nubes electrónicas [9], en este caso de los átomos de hidrógeno. Por ello, la distribución temporal de las formas tiene un pico máximo en la conformación alternada y un mínimo en la eclipsada.

Por otra parte, si se sustituyen algunos de los átomos de hidrógeno del etano la molécula perderá simetría. Así, la molécula podrá adoptar más de una conformación alternada y más de una conformación eclipsada diferente. En cualquier caso, la conformación de menor energía libre corresponderá a una de las formas alternadas y la de mayor energía a una de las formas eclipsadas.

No obstante, examínese ahora un compuesto que contenga varios enlaces que provoquen isomería conformacional. En dicho caso, la determinación de la conformación de menor energía es menos simple. En esta ocasión, no se puede garantizar que en la forma de mínima energía todos los enlaces rotables se hallen en conformación alternada (aspecto más discutido en el capítulo 2.5). Otras excepciones a la regla de la forma alternada de mínima energía son efecto Gauche [10] y la deslocalización vecinal. [9]

Los aminoácidos proteicos, exceptuando la glicina y alanina, contienen enlaces rotables en sus cadenas R –denominados enlaces chi– que provocan isomería conformacional. Además, dependiendo del aminoácido en cuestión, pueden llegar a tener hasta un total de 4 de dichos enlaces. El ángulo que posee cada uno de los enlaces chi se denomina ángulo chi (χ). [11]

De este modo, se puede visualizar una proteína como un conjunto de residuos donde (casi) todos ellos presentan isomería conformacional en las cadenas R. De este modo, es comprensible la dificultad que implica simular por completo una proteína. Si por otra parte, en lugar de una proteína se simula un conjunto de proteínas que forman un complejo, la dificultad de la simulación es aún mayor. Especialmente, es de complejidad la simulación de las cadenas laterales de la zona de interacción entre las dos proteínas. No obstante, se han desarrollado metodologías – como las basadas en dinámica molecular [12] o las basadas en librerías experimentales [11]– para mejorar la precisión de dichas simulaciones. Con todo eso, la simulación estructural de la zona de interacción entre proteínas sigue siendo un campo de continua mejora. La importancia de predecir estructuralmente esta zona es clave en biomedicina. [13][14]

Los siguientes capítulos de la memoria contienen la propuesta de este TFM para caracterizar las cadenas R, en particular de aquellos aminoácidos que cambien de posición al formarse el complejo (ya sean de la zona de interacción o no). El aspecto estudiado fue la desviación de cada ángulo de enlace chi respecto a su ángulo en una conformación de gran estabilidad del aminoácido. El apartado 2.5 explica en concreto cómo se halló la conformación de menor energía aproximada para cada aminoácido.

2.1 Definición de parámetros Δλ y Kd

La definición del parámetro Δλ es una propuesta de este proyecto. Representa un modo de cuantificar los cambios de estabilidad relacionados con cambios en la isomería conformacional al formarse un complejo.

Para describir Δλ se precisa definir antes también otro parámetros propuestos. Se trata de las variables $\lambda_{aa.chi}$, λ_{aa} y λ .

$\lambda_{aa.chi}$:

$\lambda_{aa.chi}$ se define como la diferencia entre el ángulo de un enlace chi (véase apartado 2.0) en un determinado estado y el ángulo de ese mismo enlace chi en la conformación de mínima energía libre:

$$\lambda_{aa.chi} = \chi_{estadoX} - \chi_{minimaenergía}$$

Fórmula 1. Definición del parámetro $\lambda_{aa.chi}$ de un determinado enlace chi. El símbolo χ representa el ángulo del enlace chi en cuestión.

El apartado 2.5 ofrece cómo se calcularon las aproximaciones de las conformaciones de mínima energía. Por su parte, el apartado 2.9.3 recoge la implementación del cálculo de $\lambda_{aa.chi}$ (y asimismo de λ_{aa}) que se realizó en Python.

λ_{aa} :

λ_{aa} se define como el sumatorio de los $\lambda_{aa.chi}$ calculados en un aminoácido. Para un aminoácido con un único enlace chi –como por ejemplo la cisteína– el valor de λ_{aa} equivaldría al de su único $\lambda_{aa.chi}$. Para un aminoácido con 4 enlaces chi –como la arginina– el valor de λ_{aa} sería el sumatorio de suss 4 $\lambda_{aa.chi}$ hallados.

$$\lambda_{aa} = \sum_{j=1} \left(\lambda_{aa\,.chi}\right)_j$$

Fórmula 2. Definición del parámetro λ_{aa} de un aminoácido. El valor máximo de j es el número total de enlaces chi del aminoácido en cuestión.

λ :

Si bien $\lambda_{aa.chi}$ era un parámetro referido a un enlace chi y λ_{aa} estaba referido a un aminoácido, el ámbito de aplicación de λ es el de una proteína que forme parte de un complejo.

λ va a definirse como el sumatorio de los λ_{aa} de residuos que cambien signiticativamente de posición al formarse el complejo. A estos residuos se les denomina en esta memoria residuos *clave.* Lógicamente, el valor de λ variará según cómo se defina *variar significativamente de posición*. El criterio seguido en este trabajo ha sido aquel recogido en el módulo codigo_en_R_para_analizar_alineamiento_estructural (capítulo 2.10.1).

$$\lambda = \sum_{k=1} \left(\lambda_{aa\,.\,chi}\right)_k$$

Fórmula 3. Definición del parámetro *λ* de una proteína. El índice k indica cada residuo que cambia significativamente de posición al constituirse el complejo.

Nota: en un principio *λaa.chi* se pensaba referir a aquellos aminoácidos de la zona de interacción del complejo. Finalmente, por cuestiones operativas se decidió referirlo en general a los residuos que cambiasen significativamente de posición al formarse el complejo. Concretamente, se hubiese requerido mayor tiempo de desarrollo de código para considerar sólo los aminoácidos de la zona de interacción. Como ventaja de la opción utilizada, destaca que así se tienen en consideración otros residuos además de los de interacción. Dichos residuos son los que, aún no estando en la zona de interacción, varían de posición como consecuencia de cambios conformacionales de las proteínas.

Δλ :

Una vez descritos las variables $\lambda_{aa.chi}$, λ_{aa} y λ , se puede proseguir a definir finalmente Δλ. El ámbito de aplicación de Δλ es de un nivel aún superior, concierne a las 4 estructuras que caracterizan un complejo de 2 proteínas. En un complejo de 2 proteínas, una de ellas va a ser denominada ligando y la otra receptor. Entonces, las 4 estructuras a las que hace referencia Δλ son:

- Ligando unbound(l_u): es la estructura del ligando en estado no unido.
- Receptor unbound (r_u): estructura del receptor en estado no unido.
- Ligando bound (l_b): la estructura del ligando cuando forma parte del complejo.
- Receptor bound (r_b): aquella estructura del receptor cuando está formando parte del complejo.

Δλ se define como la diferencia entre los parámetros *λ* del ligando y receptor en estado no unido y aquellos en estado unido (Fórmula 4).

$$\Delta\lambda=\lambda_{l-u}+\lambda_{r-u}-\left(\lambda_{l-b}+\lambda_{r-b}\right)$$

Fórmula 4. Valor *Δλ* de un complejo proteico. Nótese como la averiguación de Δλ requiere a su vez la de los diferentes parámetros λ. A su vez, el conocimiento de λ requiere el cálculo de los valores λ_{aa} y éstos últimos de los valores $\lambda_{aa.chi}$.

Una vez descrito el significado de *Δλ* en este trabajo se prosigue ahora con Kd:

Kd simboliza la constante de disociación de un complejo proteico, como está definida tradicionalmente en la literatura científica. [15][16][17] La fórmula 5 recoge la reacción de disociación de un complejo de dos proteínas. La fórmula 6 se basa en la fórmula 5 para especificar el valor de Kd del complejo.

$$AB \Leftrightarrow A+B$$

Fórmula 5. Reacción de disociación de un complejo de 2 proteínas. El complejo está simbolizado por AB, mientras que las proteínas por separado corresponden a A y B.

$$Kd=\frac{[A][B]}{[AB]}$$

Fórmula 6. Constante de disociación del complejo AB. Los términos [A], [B] y [AB] representan las concentraciones en el equilibrio de A, B y AB respectivamente.

2.2 Motivación racional del modelo a estudiar

El principal objetivo del trabajo fue investigar una supuesta correlación entre Δλ y Kd. A continuación, se exponen en profundidad los motivos que hicieron suponer esa posible relación:

Dada la definición de λ (apartado 2.1) se puede concebir dicho parámetro como una desviación sobre la estabilidad en una proteína. Más en concreto, como una desviación sobre la estabilidad en lo referente a la isomería de sus cadenas R. Más exactamente, sólo de aquellas cadenas R de los residuos que varían de posición la formarse el complejo. Esto es equivalente a concebir λ como una medida de *inestabilidad.* Es decir, mediría la inestabilidad de una proteína a raíz de los patrones isoméricos anteriormente comentados.

Con esta perspectiva, revísese ahora no el concepto de λ sino el de Δλ (también capítulo anterior). Puede observarse que una interpretación de Δλ sería la de la fórmula 7:

$$\Delta\lambda = \sum inestabilidad_c inicial - \sum inestabilidad_c final$$

Fórmula 7. Lectura de *Δλ* en términos de inestabilidad. El estado inicial hace referencia al ligando y receptor en estado unbound. El estado final es ligando y receptor formando el complejo. La 'c' en ambos sumandos denota que sólo se está considerando la inestabilidad asociada al patrón isomérico de ciertos enlaces chi.

De esta manera, Δλ va a medir un cambio de inestabilidad al crearse el complejo proteico. Como la constitución del complejo ocurre efectivamente en la naturaleza, es de esperar que la inestabilidad final sea menor que la inicial. Asimismo, cuanto mayor sea el valor de *Δ*λ mayor será la estabilidad conseguida –asociada a la isomería conformacional delas cadenas R– al formarse el complejo.

De este modo, Δλ se podría asimilar en este sentido a un cambio de energía libre de Gibbs de la reacción. No obstante, ha de recordarse que Δλ sólo es una medición de inestabilidad relacionada con un determinado parámetro estructural. A ciencia cierta se desconoce el papel que juega Δλ en la termodinámica global de reacción. Una de las metas que pretendía este TFM era aportar conocimiento en este sentido.

Por otro lado, hay que reseñar que Δλ no constituye ni una medida indirecta ni una aproximación de cambios en la energía libre de Gibbs. De hecho, sus unidades no son de energía sino de diferencias angulares. Con todo eso, es fácilmente entendible que ambas magnitudes estén relacionadas; Δλ está midiendo desviaciones con respecto a un sistema de estabilidad.

Por todo esto, se ha hipotetizado una posible relación entre Δλ y Kd. Ambas magnitudes deberían afectarse de manera inversamente proporcional. Un mayor valor de Δλ debería contribuir a mayor concentración de complejo en el equilibrio. Dada la fórmula 6, esto equivaldría a una menor Kd.

2.3 Panorámica general del algoritmo de cálculo de los valores Δλ

De acuerdo a las fórmulas 1-4 recogidas en el apartado 2.1, se puede generar un algoritmo para el cálculo de Δλ. La Figura 3 recoge un diagrama de flujo del cálculo de Δλ de un complejo proteico.

Al examinar la Figura 3, se observa que es preciso conocer

qué residuos de las proteínas son los denominados residuos *clave*. El capítulo 2.11 contiene una exposición de cómo se identificaron dichos residuos.

Por otra parte, también se puede apreciar un control de calidad en el cálculo de Δλ. Sólamente se calculó Δλ si se habían podido analizar correctamente al menos el 90% de los residuos *clave*. Las causas para no poder analizar un residuo *clave* son muy diversas. Fundamentalmente, se trata de residuos que por sus características en el fichero .pdb que los contiene no han podido ser procesados.

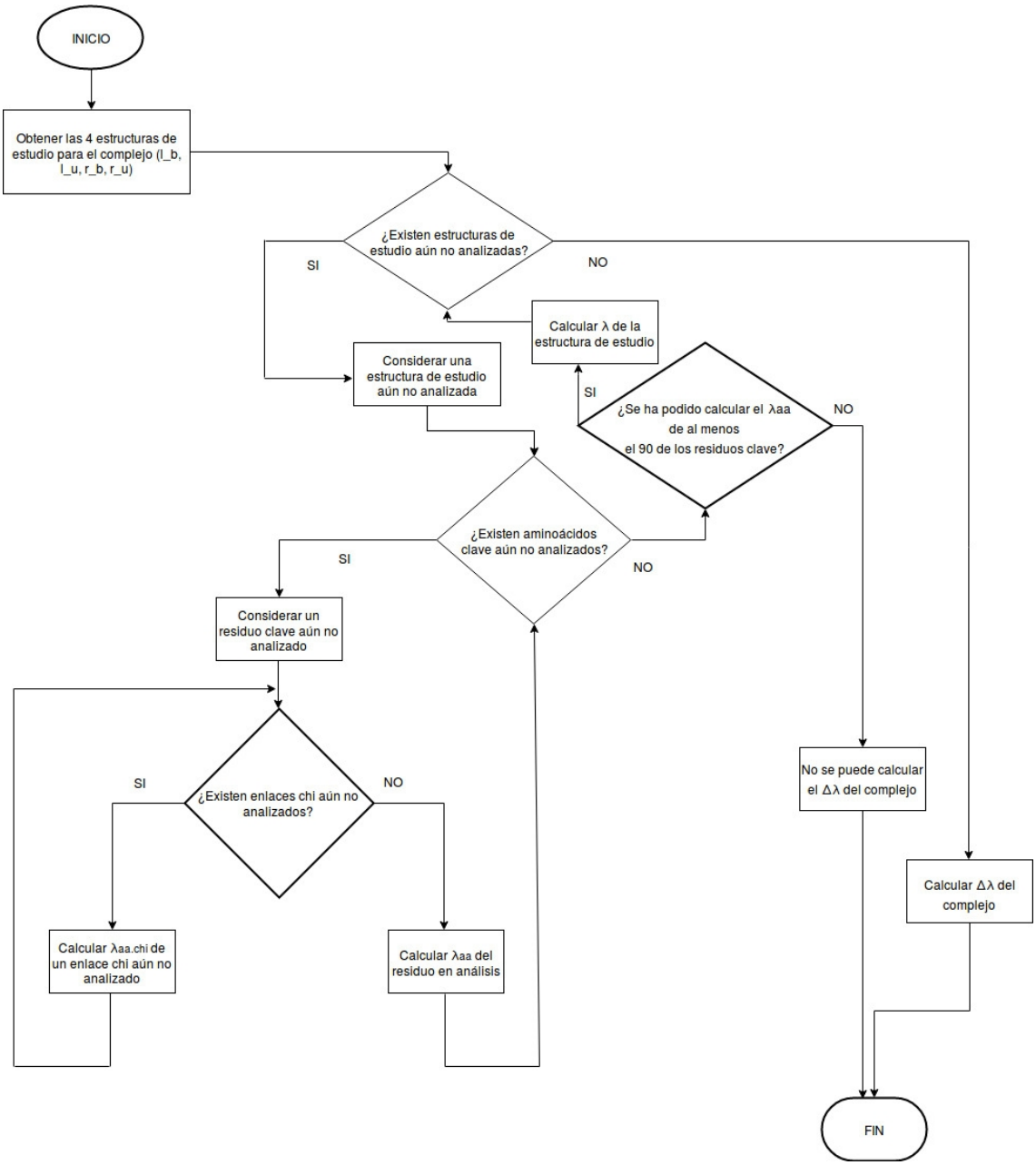

Figura 3. Pasos necesarios para poder calcular el Δλ de un complejo proteico.

2.4 Algoritmos para la creación del modelo y la prueba de hipótesis

La creación del modelo lineal de este trabajo requiere de una lista de pares Δλ-Kd. La Figura 4 recoge con qué procedimiento se obtiene dicha lista en el programa.

Uno de los pasos de la Figura 4 se muestra una decisión en función de si se han obtenido o no los residuos clave de las 4 estructuras de cada complejo. Si no se han obtenido, es que no se han podido realizar correctamente los alineamientos estructurales para el complejo. Una de las posibles causas es que la proteína (ligando o receptor) en cuestión contenga más átomos de los que soporta por defecto el software de alineamiento estructural.

Por su parte, el paso denominado «Intentar calcular el Δλ del complejo» corresponde al proceso explicado anteriormente (Figura 3).

Una vez formada la lista de pares Δλ-Kd; se pasa a generar un modelo lineal que relacione ambas variables. Este proceso se realiza de acuerdo a la forma estándar recogida en la literatura científica. [18] [19]

En lo referido a la prueba de hipótesis, ésta está basada en comparar una observación particular (Δλ-Kd) con el modelo lineal previamente generado. Dependiendo del ajuste de la observación al modelo se podría rechazar la observación – por no concordar con lo predicho por el modelo.

La metodologia para llevar a cabo la prueba de hipótesis contiene las siguientes etapas:

1º Cálculo de la Kd predicha por el modelo de acuerdo al dato de Δλ de la observación:

Se va a suponer que dicho Kd calculado es una variable aleatoria que sigue una función de probabilidad normal. Esta variable aleatoria tiene como media el Kd predicho por el modelo. Como desviación típica tiene aquella característica del modelo lineal.

2º Prueba de test de t de Student para rechazar o no que la Kd de la observación corresponda a un valor de la Kd predicho por el modelo:

Se desconoce el tamaño muestral. Es decir, no se conoce cuántas mediciones se hicieron para calcular el valor de la Kd del problema. Por defecto, se va a especificar un tamaño muestral de n = 3 .

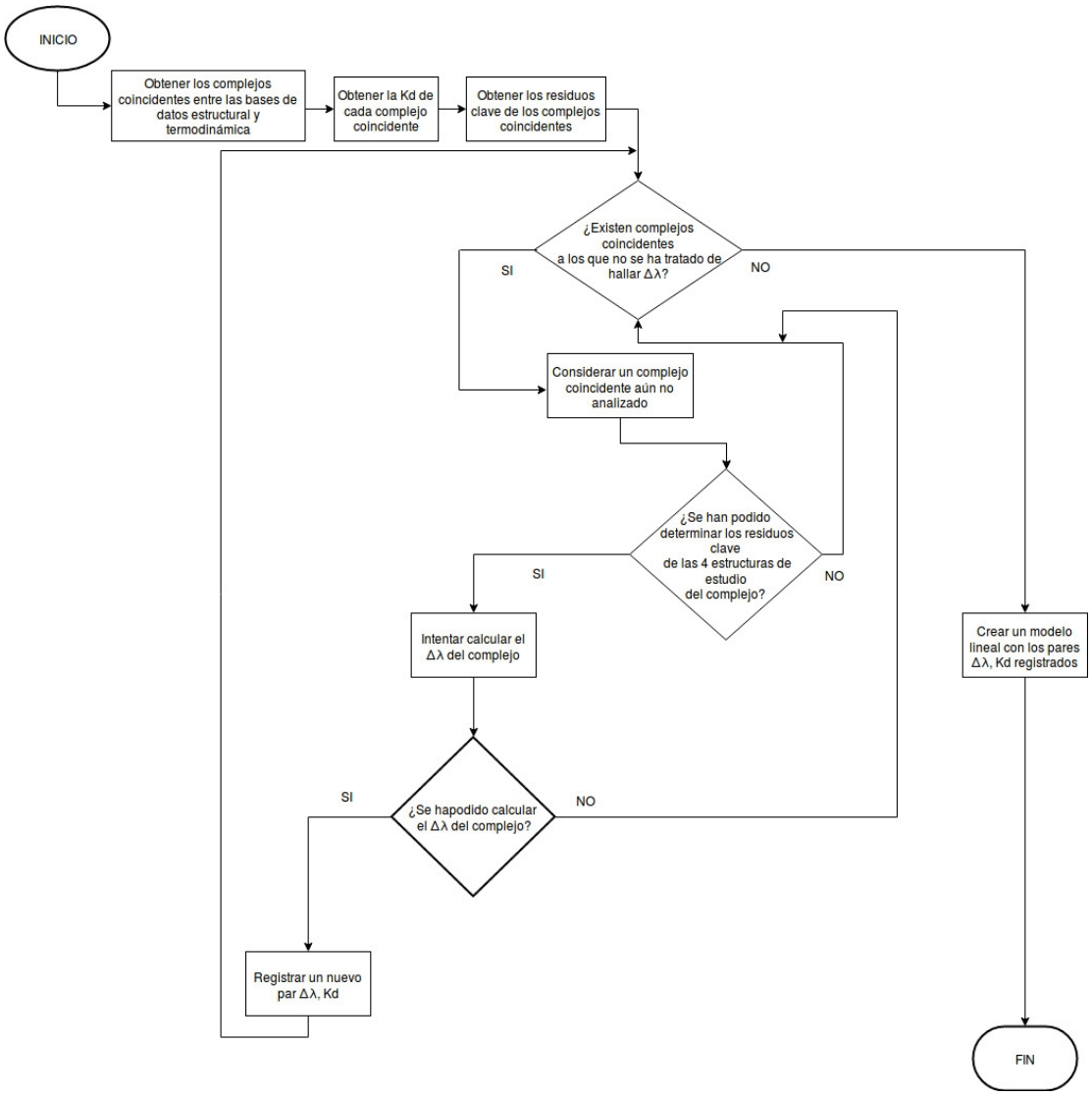

Figura 4. Obtención de datos para crear un modelo lineal que relacione Δλ y Kd. El paso «Intentar calcular el Δλ del complejo» engloba al proceso descrito en la Figura 3.

2.5 Determinación de los ángulos chi estables

El capítulo 2.1 muestra como la averiguación de $\lambda_{aa.chi}$ se apoya en un un sistema de referencia de mínima energía. De este modo, se puede comparar cada ángulo chi con su valor en la conformación de mínima energía.

No obstante, la determinación de las formas más estables no es trivial. De hecho, éstas varían según cada caso específico. Es decir, la conformación más estable de cada aminoácido –y aún de cada enlace chi de cada aminoácido– va a depender de múltiples factores. Entre ellos, los aminoácidos vecinos, la estructura de la proteína, la naturaleza del solvente, la temperatura, etc. El criterio seguido en este trabajo ha

sido directo; considerar una única conformación estable para cada uno de los 20 aminoácidos proteicos. Esto supone en cierta medida una simplificación, pero la magnitud del proyecto no permitía profundizar más en este aspecto. De esta manera, es más correcto referirse a los ángulos chi a calcular como ángulos chi estables. Es decir, no corresponden realmente a ángulos de estabilidad máxima (ángulos chi de equilibrio). A pesar de ello, a lo largo de este proyecto se nombran indistintamente también como ángulos chi de equilibrio.

Por todo esto, la obtención de los ángulos chi estables ha requerido de un procedimiento y del desarrollo a su vez de software específico. Las fases del procedimiento han consistido en:

1º Descarga de la estructura en formato SDF de cada uno de los 20 aminoácidos proteicos. Las estructuras están disponibles en RSCB PDB . [20] Dentro de las opciones «Model» e «Ideal» se escogió la última por corresponder a aquella de mayor estabilidad del aminoácido. [21]

2º Utilización de la herramienta OpenBabel [22] para la conversión de los ficheros SDF a formato PDB. La conversión del aminoácido serina no es llevada correctamente de este modo, por lo que se tuvo que realizar manualmente.

3º Cálculo de los ángulos chi de equilibrio (ó estables) de las estructuras PDB.

El software específico desarrollado para el procedimiento incluye los módulos auxiliares:
a) transformar_ficheros_sdf_ideal_a_pdb.py
b) calcular_chi_angles_equilibrio.py.
Ambos módulos se tratan en los capítulos 2.9.8 y 2.9.9 respectivamente.

2.6 Características de las bases de datos utilizadas

Se han barajado un total de 4 bases de datos distintas. Dos de ellas corresponden a bases de datos estructurales y dos a bases termodinámicas.

Como base de datos estructural, se consideró en primer lugar DOCKGROUND. [23] DOCKGROUND contiene estructuras PDB de complejos proteicos y de proteínas en estado unbound. No obstante, dada la metodología finalmente empleada de alineamientos estructurales resultaba no adecuado trabajar con esta base de datos.

Benchmark [24] por su parte es otra base de datos estructural. Contiene para cada complejo proteico las estructuras PDB de ligando y receptor en estado bound y unbound. Así pues, facilitó la estrategia de

alineamientos estructurales para el hallazgo de los residuos *clave*. Además, el cálculo de los diferentes parámetros relacionados con Δλ se realiza a partir de las estructuras PDB que contiene. La utilización de Benchmark por el programa requiere la descarga de sus estructuras PDB en el equipo del usuario.

PINT Database [4] se escogió en un principio como base de datos termodinámica. No obstante, más adelante se sustituyó por el banco de datos más extenso PDBBind. [5]

PDBBind recoge diversos conjuntos de complejos biológicos, entre ellos de ADN-Proteína, Proteína-pequeñas moléculas y Proteína-Proteína. Este último es el empleado por el software. Cada entrada de información de PDBBind contiene la constante de disociación del complejo proteico. Para el uso de PDBBind se requiere que el usuario disponga del archivo INDEX_general_PP.2016 o análogo. Este archivo está disponible en el portal PDBBind bajo licencia gratuita.

2.7 Diferentes lenguajes de programación empleados

La mayoría del programa ha sido desarrollado en Python 2.7 [25], que además ha servido como lenguaje aglutinador de todo el programa. Asimismo, se han empleado funciones ya desarrolladas en Python ubicadas dentro del framework Lightdock. [6]

Por otro lado, las partes del software más relacionadas con la rama estadística han sido creadas en R [26].

Finalmente, la aplicación externa de alineamientos estructurales Lovoalign [27] –Fortran 90– se integró asimismo dentro del programa. Además, la herramienta OpenBabel [22] (implementado en C++) fue de utilidad para la conversión a formato PDB de estructuras aminoacídicas.

2.8 Módulos externos en Python

Se han utilizado en el programa una serie de funciones pertenecientes a otros proyectos. En específico, se ha empleado software del framework Lightdock. [6] Lightdock es un proyecto amplio con software de gran interés en el campo de las interacciones interproteicas. En este proyecto ha sido de utilidad:

a) La clase «Complex» del módulo lightdock.structure.complex:

La creación de objetos de esta clase permite a su vez seleccionar residuos específicos. Por ejemplo, supóngase que se transforma una proteína denominada X a un objeto de clase «Complex». Entonces se

podrán seleccionar en Python sus residuos mediante X.residues[i]; siendo i el índice del residuo en cuestión.

b) La función «parse_complex_from_file» de lightdock.pdbutil.PDBIO:

Esta función lee una estructura proteica PDB y devuelve una serie de variables que son las que posibilitan crear objetos de la clase «Complex».

c) La función «calculate_chi_angles» de lightdock.rotamer.predictor:

Mediante «calculate_chi_angles» se puede hallar los ángulos chi de los residuos de los objetos «Complex».

Las 3 piezas de software de Lightdock son importadas en los módulos calcular_chi_angles_equilibrio.py, parte1.py y parte2.py. En el primer de los módulos contribuyen al cálculo de ángulos chi (en este caso del sistema de referencia de estabilidad). En los otros dos módulos miden los ángulos chi de las proteínas de estudio y así poder ccontribuir al cálculo de Δλ.

2.9 Módulos creados en Python

En los capítulos siguientes se tratan los diversos módulos de Python que han sido creados para poder llevar a cabo el proyecto.

Nota: El código disponible en los anexos contiene muchas referencias a la palabra lambda. Dependiendo del contexto, ésta puede estar refiriéndose a:
λaa.chi: si se está haciendo referencia a un enlace.
λaa: referido a un residuo.
λ: si se está trabajando con una proteína.
Δλ: si se está trabajando con un complejo proteico.

2.9.1 Módulo extraer_informacion_bases_de_datos.py

Este módulo contiene funciones para extraer las coincidencias entre las bases de datos estructural y termodinámica. Es decir, qué complejos proteína-proteína registrados en una de las bases de datos se hallan también registrados en la otra base.

El módulo contiene asimismo una función para extraer las constantes de disociacion (Kd) de una lista de complejos proteicos. Dicha función por tanto es la que interpreta la información de la base de datos PDBBind.

Además, en este módulo se halla una función de transformación de unidades. Dicha función es invocada a su vez por la de extracción de Kd. Así, se logra referir todos los datos de Kd en nanomolar.

El código del módulo está disponible en el anexo IA.

2.9.2 Módulo alineamientos_estruc.py

En este módulo se halla por un lado una función para el cálculo de los alineamientos estructurales. Recibiendo los parámetros adecuados, la función «calcular» invoca al software de Lovoalign. Por otro lado otra función denominada «analizar» es la encargada de identificar los residuos clave, a los que denomina en este caso residuos *relevantes*. Para ello invoca a su vez al módulo en R codigo_en _R_para_analizar_ alineamiento_estructural.R .

El anexo IB contiene el código correspondiente a este módulo.

2.9.3 Módulo func_lambda.py

En primer lugar, este módulo contiene una función para la lectura del fichero con los ángulos chi de equilibrio. Ésta es pues una forma de evitar codificar de forma "hardcore" los ángulos chi de equilibrio en el programa. Si se desea utilizar otros ángulos chi de equilibrio solo habrá que modificar el fichero de ángulos chi de equilibrio.

En el módulo func_lambda.py se halla asimismo otra función, denominada «contribucion_residuo». Esta función es la responsable del cálculo de $\lambda_{aa.chi}$ y λ_{aa}, siendo el segundo de estos parámetros el que retorna cuando es invocada.

El código se encuentra en el anexo IC.

2.9.4 Módulo primer_indice_res.py

Este módulo (código en el anexo ID) identifica el índice del primer residuo de un fichero PDB. Ha de recibir la ruta de dicho fichero PDB como input.

De esta manera, se puede compatibilizar los residuos que devuelve Lovoalign con los que interpreta la clase «Complex» de lightdock.structure.complex. Al identificar el primer residuo del fichero PDB, se puede hallar la correspondencia entre ambas formas de indexar a los residuos.

2.9.5 Módulo parte1.py

Esta parte del software es la implementación del proceso descrito en la Figura 4. Por tanto, parte1.py elabora en primer lugar una lista de pares Δλ-Kd; y a continuación un modelo a partir de dicha lista. Para lograr todo ello, importa los 4 módulos anteriores recién descritos además de los módulos externos de Lightdock.

El código de parte1.py está disponible en el anexo IE.

2.9.6 Módulo parte2.py

El software parte2.py (anexo IF) calcula la observación (Δλ-Kd) del complejo problema. A continuación, invoca al software en R de prueba de hipótesis (capítulo 2.10.3) para testear los resultados con el modelo lineal.

2.9.7 Módulo software_completo.py

Módulo a ejecutar por el usuario. A éste se le ofrecen tres opciones:

1) Como primera opción se ofrece crear un modelo lineal a partir de las bases de datos. Esto se consigue mediante la ejecución del módulo parte1.py .

2) Como segunda opción posibilita el testeo de un complejo problema con un modelo creado a partir de las bases de datos. Para ello, se ejecuta secuencialmente parte1.py y parte2.py

3) Finalmente, como última opción ofrece testear el complejo problema sin crear un modelo a partir de bases de datos. De este modo, el usuario puede confeccionar manualmente el fichero de input modelo_lineal.txt . Esta opción sólo requiere la ejecución de parte2.py .

El código de software_completo.py (anexo IG) es muy simple, al estar la complejidad contenida en los módulos anteriores.

2.9.8 Módulo auxiliar transformar_ficheros_sdf_ideal_a_pdb.py

Esta pieza de código –anexo IH (1)– tiene como finalidad transformar cada una de las estructuras aminoacídicas estables de formato SDF a formato PDB. A partir de una lista de abreviaturas de los 20 aminoácidos proteicos, ejecuta el software externo OpenBabel [22] para obtener los ficheros PDB.

Por alguna razón desconocida, la conversión de formato del aminoácido serina no es llevada a cabo correctamente. Dicha conversión se hubo de realizar manualmente.

2.9.9 Módulo auxiliar calcular_chi_angles_equilibrio.py

Este script –anexo IH (2)– procesa los ficheros PDB de los aminoácidos en su configuración estable. Como resultado, proporciona los ángulos chi que poseen cada una de dichas estructuras.
El código hace uso de la función «calculate_chi_angles» de lightdock.rotamer.predictor.

2.10 Módulos creados en R

Para un desarrollo más rápido del software del proyecto, se crearon diversos módulos en el lenguaje de programación R. La mayor experiencia en programación estadística utilizando este lenguaje motivó esta acción.

2.10.1 Módulo codigo_en_R_para_analizar_alineamiento_estructural.R

Este código devuelve los índices de los residuos que más han cambiado su posición al pasar la proteína a formar parte de un complejo. Éste es pues el código responsable de establecer qué residuos son los residuos *clave* de una proteína.

Para lograr su objetivo, el módulo analiza los resultados de cada alineamiento estructural. Primero calcula el percentil 90 de la distribución de RMSFs de los átomos. A continuación, selecciona los residuos que contengan al menos un átomo con un RMSFs superior a dicho percentil.

El código está disponible en el anexo IIA.

2.10.2 Módulo codigo_en_R_para_generar_modelo.R

El propósito de este módulo es generar un modelo lineal a partir de los pares Δλ-Kd obtenidos como parte de la ejecución de parte1.py. Corresponde así al último paso del proceso descrito en la Figura 4.
Código disponible en anexo IIB.

2.10.3 Módulo codigo_en_R_para_prueba_de_hipotesis_con_el_modelo_y datos_del_problema.R

Este script –anexo IIC– es la implementación en R de la prueba de hipótesis de conformidad con el modelo (explicada en apartado 2.4). Como salida, imprimirá en sobre_el_rechazo_de_la_hipotesis.txt la palabra "SI" si se rechaza la hipótesis o "No" si no se rechaza.

2.11 Determinación de los residuos que cambian de posición al formarse el complejo –residuos clave–

La búsqueda de los residuos *clave* se ha basado en la ejecución de alineamientos estructurales. El principio es que si se alinea a una proteína consigo misma se pueden identificar los residuos que cambian de posición.Estos alineamientos han sido facilitados por el software Lovoalign. [27] La principal ventaja de Lovoalign fue funcionaba a través de comandos. De esta manera, se pudo controlar en el software de este proyecto qué alineamientos iba realizando Lovoalign.

En concreto, para complejo proteico se realizan un total de 4 alineamientos estructurales. Los pares de dichos alineamientos son:

I) Estructura de ligando bound – Estructura de ligando unbound.
II) Estructura de ligando unbound – Estructura de ligando bound.
III) Estructura de receptor bound – Estructura de receptor unbound.
IV) Estructura de receptor unbound – Estructura de receptor bound.

Como resultado de cada alineamiento se obtiene un fichero de resultados. Este fichero contiene información de cada átomo de la primera de las 2 proteínas en el par del alineamiento. Más exactamente, para cada átomo se recoge una medida (RMSF) [27] de la diferencia de posición espacial con respecto a la otra proteína alineada.

Por otra parte, se desarrolló el módulo de R tratado en el capítulo 2.10.1 para analizar precisamente el fichero de resultados que ofrece Lovoalign para cada alineamiento. De esta manera se logra determinar mediante codigo_en_R_para_analizar_alineamiento_estructural.R los residuos *clave* de cada una de las 4 estructuras del complejo. El capítulo 2.10.1 ofrece el criterio seguido para ello.

2.12 Resultados

En los siguientes capítulos se exponen los resultados obtenidos.

2.12.1 Coincidencias de complejos entre las bases de datos estructural y termodinámica

La ejecución del módulo parte1.py muestra un total de 132 coincidencias entre las bases de datos Benchmark y PDBBind en el momento actual.

Dado el criterio de calidad en el cálculo de Δλ expuesto en el capítulo 2.3, el software sólo ha obtenido 94 de los 132 en principio posibles pares Δλ-Kd.

2.12.2 Visualización del gráfico Kd VS Δλ

En la Figura 5 se muestra el gráfico Kd VS Δλ obtenido tras la ejecución de parte1.py .

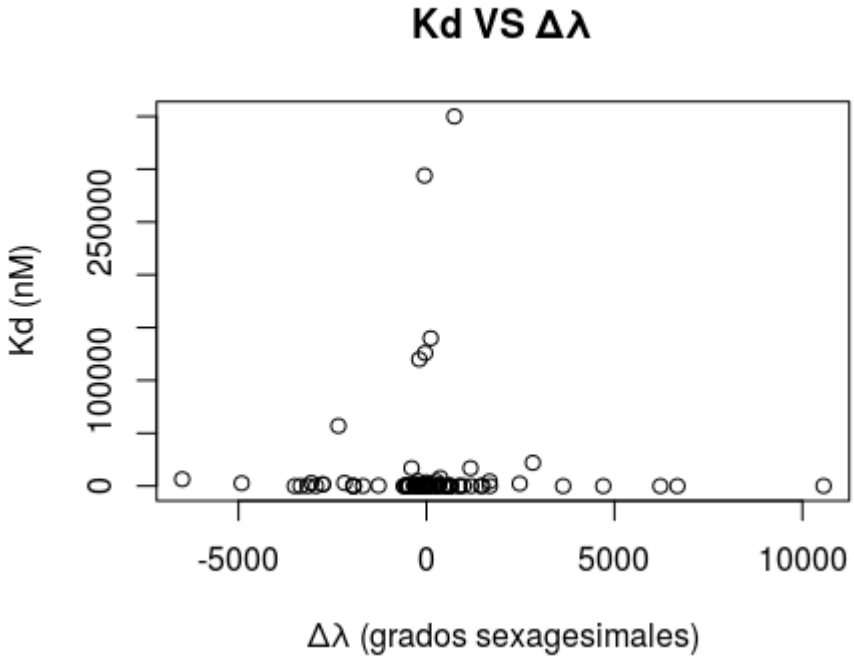

Figura 5. Gráfico de dispersión entre las variables Kd y Δλ.

La observación de la Figura 5 muestra de forma directa un muy bajo grado de dependencia entre las variables. No obstante, en los dos capítulos siguientes se investiga dicha relación de forma analítica.

2.12.3 Análisis de independencia entre las variables Kd y Δλ

Para conocer la relación –o su ausencia– entre Kd y Δλ se llevó a cabo un estudio donde se dividió el espacio cartesiano en 4 bloques. Los ejes divisorios fueron la media de Kd y Δλ, respectivamente. Así, cada par Kd -Δλ se convirtió en una observación para 2 variables cualitativas. Una de dichas variables cualitativas sería la que determina si la Kd del punto es menor o mayor que la media de Kd; y la otra variable determina si la Δλ del punto es menor o mayor que la media de Δλ.

Un test de independencia de Fisher [28] entre las 2 variables cualitativas mostraría sí se podría rechazar o no una correlación entre Kd y Δλ. Esto es debido a que en caso de correlación sí habría dependencia entre las dos variables cualitativas definidas. En el caso de este trabajo, se

buscaba hallar una correlación inversamente proporcional entre Kd y Δλ (capítulo 2.2).

Los resultados del test de Fisher se muestran en la Figura 6. En lugar de realizarse un único test de Fisher, se realizó una serie de ellos donde variara el cálculo de las medias de Kd. En particular, en cada test de Fisher el factor de acortamiento (Trim en inglés) para el cálculo de la media de Kd era diferente. De este modo, se eliminaba una proporción menor o mayor de los valores extremos de Kd para el cálculo de su media.

En ningún caso el p-valor de algún test de Fisher fue inferior a 0.1. Al mismo tiempo, se aprecia una tendencia a menores p-valores a medida que aumenta el factor Trim (Figura 6).

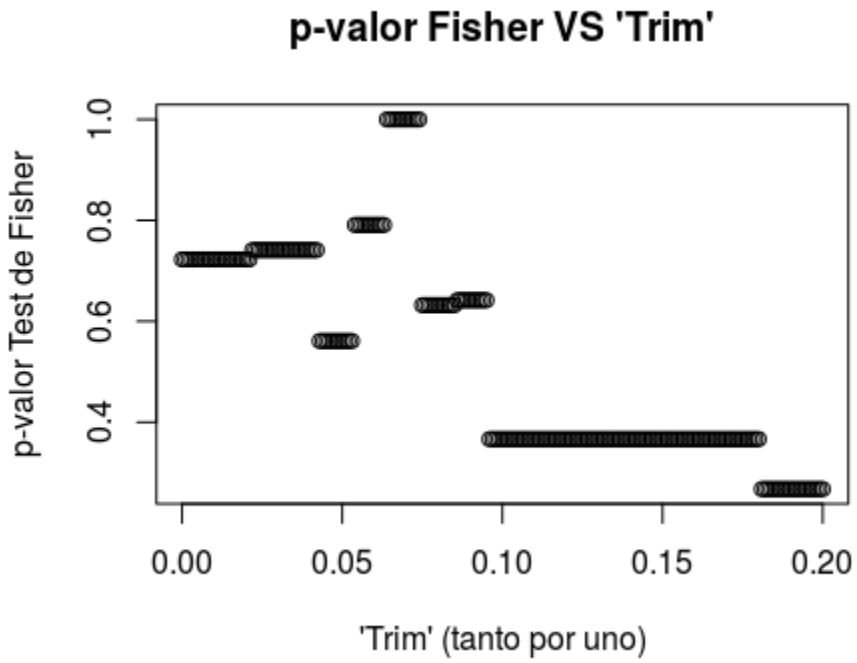

Figura 6. Tests de Fisher para determinar si se puede rechazar o no la correlación entre Δλ y Kd. El factor Trim es la proporción de valores extremos de Kd no incluídos para el cálculo de la media de Kd.

2.12.4 Validez o no de la hipótesis de trabajo

Frente a la situación descrita en el capítulo anterior, se buscó indicios de que pudiera existir una correlación (aunque débil) entre Δλ y Kd.

En busca de la correlación lineal inversa, se realizaron los siguientes pasos:

1º División del eje OX (variable Δλ) en cuatro intervalos delimitados por los cuartiles de Δλ.

2º Averiguación de los valores medios de Kd en cada uno de los 4 intervalos.

3º Comprobación de si las medias calculadas descienden desde el primer intervalo (primer cuartil) al segundo, del segundo al tercero y del tercero al cuarto. Este orden descendiente sería esperable para una relación lineal inversa entre Δλ y Kd.

De forma análoga a la táctica del capítulo anterior, la comprobación del paso 3º se realizó para diversos valores del factor Trim. En este caso, el factor Trim va a afectar a la media de cada intervalo. Los resultados (Figura 7) muestran que la situación esperada sólo se da con factores 'Trim' mayores de 0.17.

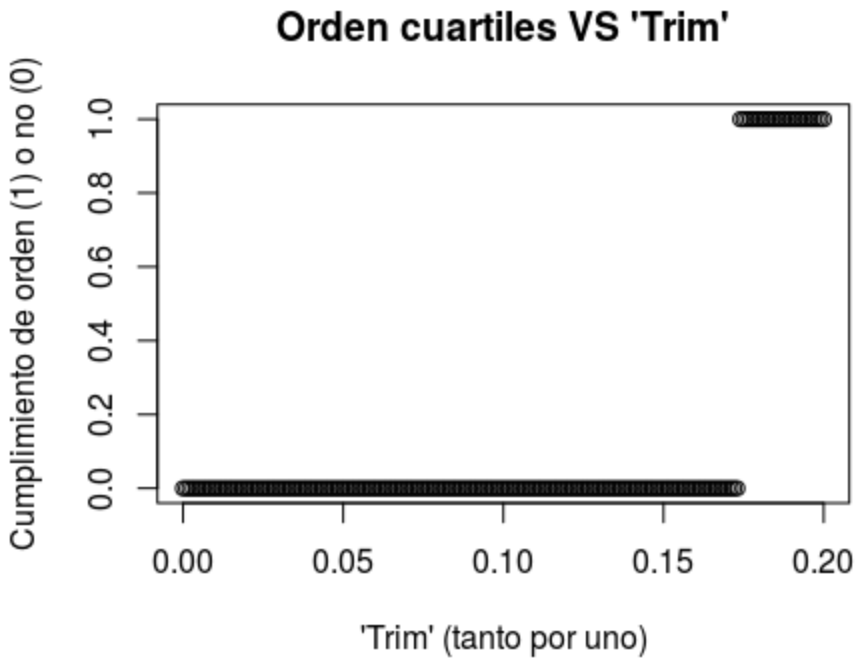

Figura 7. Cumplimiento del orden decreciente de las medias de Kd, organizando los datos según los cuartiles de Δλ.

La combinación de la información de las Figuras 6 y 7 hace sospechar de unos indicios de relación lineal inversa (tal como se estimaba en el apartado 2.2). No obstante, dicha posbile relación sólo ocurriría si se elimina una alta proporción de valores extremos de Kd y aún así el p-valor del test de Fisher no sería inferior de 0.1 . Por tanto, en ese caso tampoco se podría rechazar la independencia entre las variables.

3. Conclusiones

3.1 Conclusiones generales

Como principal conclusión del TFM se tiene que no se puede demostrar una relación entre ambas variables de estudio. Es decir, no se puede probar que los cambios de isomeria globales de las cadenas R afecten significativamente a la termodinámica de formación de los complejos proteicos.

Por lo recién expuesto, se podría afirmar que probablemente sean otro tipo de cambios estructurales en las proteínas los que afecten significativamente a la constante de disociacion de los complejos proteicos. Ese tipo de cambios pueden estar relacionados con otro tipo de alteraciones de las cadenas R distintas a las de la isomería. También podrían deberse a alteraciones de la estructura del esqueleto polipéptidico.

Al mismo tiempo, hay indicios de que sí podría existir la esperada relación entre ambas variables. Esto es notable cuando se eliminan los valores más extremos de Kd. Sería necesario un refinamiento de la determinación de los ángulos chi del sistema de referencia de estabilidad. Es decir, que la determinación de los ángulos chi de estabilidad tuviera en cuenta más factores (como por ejemplo la identidad de los residuos vecinos). De este modo, puede que el indicio de la esperada relación se confirmase.

Ocurre que es principalmente la isomería de las cadenas R el aspecto menos sencillo de simular. Por ello existe una necesidad de evaluar las simulaciones realizadas particularmente en dicho aspecto. Más aún si se trata en complejos proteicos. Este es el camino del que este TFM ha avanzado un primer paso .

3.2 Consecución de los objetivos

Se han cumplido los objetivos propuestos inicialmente. Por un lado, se ha desarrollado un software capaz de generar un modelo de relación de las variables de estudio. Por otro, se han creado herramientas estadísticas que en base al modelo testean la calidad de simulaciones de complejos proteicos. Por último, el acceso a ambas funcionalidades está unificado en único programa que puede ser utilizado con facilidad por el usuario.

Es de destacar además que el proceso se ha automatizado totalmente. El usuario sólo tiene que proporcionar los datos de input al programa.

Si bien los objetivos se han cumplido, el test de calidad de simulaciones de complejos generado no es de aplicación práctica debido a la no correlación entre las variables de estudio. No obstante, el mismo algoritmo propuesto (e incluso el mismo código) es de utilidad para cualquier otro test de cálidad análogo basado en otros parámetros.

De igual forma, las piezas de software relacionadas con el tratamiento de la isomería pueden servir para otro tipo de proyectos. Esto se debe en parte a que el software ha sido diseñado de forma modular.

Finalmente, hay que destacar que el proyecto ha aportado conocimiento hasta ahora desconocido a la comunidad científica.

3.3 Seguimiento de la planificación

En términos generales se ha seguido la planificación propuesta en un principio. Ha habido situaciones en las que alguna de las tareas se retrasó, pero finalmente la situación fue resuelta. Por ejemplo, como se detalla en la PAC2, el módulo con las funciones relacionadas con el cálculo de $\Delta\lambda$ no se terminó en plazo. No obstante, al mismo tiempo el modúlo de R relacionado con el test de calidad se había completado antes de plazo. El tiempo ahorrado en una tarea se invirtió en la otra.

Por otra parte, la propia redacción de esta memoria acumuló cierto retraso inicial de aproximadamente tres horas. Para subsanarlo, fue necesario incrementar la carga diaria de trabajo dedicada al TFM.

Por tanto, en un principio la planificación propuesta ha sido realista. También es cierto que el seguimiento de la planificación ha influído en la metodología de trabajo. En concreto, el ajuste a los plazos ha influído en la forma de desarrollar el software.
De este modo, se ha priorizado el ajuste al calendario sobre la eficiencia final del programa. La razón de esto estriba en que, conforme se probaba el código en creación, se hacía evidente que el programa sería de ejecución rápida. Dicha ejecución tardaría como sumo horas.
A efectos prácticos, para este tipo de programas no hay diferencias significativas en reducir incluso hasta en un 50% el tiempo de ejecución. La razón estriba en que no se requiere de una interacción continua con el programa por parte del usuario. El usuario simplemente aporta los datos de input y ordena comenzar el programa. Además, en estos intervalos de tiempo el efecto de una reducción no es prioritario.
Como ejemplo de lo expuesto, véase el caso siguiente que tuvo lugar desarrollando el software:

Como se trató en uno de los apartados anteriores, una de las etapas clave del programa fue la realización de alineamientos estructurales. Estos alineamientos superponían cada una de las proteínas del complejo con ella misma. La meta era conocer qué residuos cambiaban más de posición espacial al constituirse el complejo.

Si se observa la función calcular del módulo alineamientos_estruc.py, se aprecia que se están realizando dos alineamientos equivalentes para cada ligando y receptor. Se trata del alineamiento de la proteína (ligando o receptor) en estado bound con ella misma en el estado unbound y viceversa (proteína en estado unbound con ella en el estado bound). De este modo, se consiguió determinar los residuos cambiantes en ambos estados de cada proteína.
Un enfoque alternativo menos costoso computacionalmente hubiese sido ejecutar un solo alineamiento en lugar de dos. A continuación, habría que haber diseñado código para con los resultados de ese alineamiento determinar los residuos cambiantes en el otro estado de la proteína. Esto se podría haber realizado gracias a que Lovoalign imprime el alineamiento como parte de sus resultados. El hipótetico código hubiese sido de más rápida ejecución que el segundo alineamiento estructural al que sustituye. Esto se debe a que hubiese realizado operaciones sobre un fichero de texto en lugar de un alineamiento de dos proteínas. No obstante, se optó por la estrategia de los dos alineamientos. La razón fue que la otra alternativa hubiese supuesto el desarrollo de un módulo de software adicional. Así, se priorizó cumplir con las metas de la planificación.

También con el objeto de cumplir con los plazos, se escogió la base de datos estructural Benchmark [24] en lugar de DOCKGROUND [20]. El ordenamiento de los ficheros en Benchmark fue ideal para la ejecución del proyecto. De hecho, el diseño de esta base de datos influyó en gran medida en el desarrollo del propio programa. En particular, la disposición de los ficheros PDB de cada proteína en estado bound y en estado unbound hizo posible la estrategia de los alineamientos estructurales. Esta estrategia aceleró el desarrollo del proyecto frente a una basada en determinación de residuos de interacción mediante accesibilidad de un solvente (SASA). No es que una estrategia fuera *per se* más rápida que la otra, sino que los ficheros PDB de los complejos proteicos no contienen exactamente los mismos aminoácidos que las dos proteínas por separado. Esto se debe a la naturaleza experimental de los ensayos cristalográficos. Por tanto, la estrategia alternativa de los alineamientos estructurales basada en Benchmark ahorró la creación de código de tratamiento de los ficheros PDB.

3.4 Adecuación de la metodología prevista

La metodología ha sido adecuada en el sentido de que ha permitido cumplir los objetivos propuestos. La descripción de las tareas a cumplir no fue ni demasiado general ni demasiado específica. Unas tareas demasiado generales hubiesen desorientado en la realización del proyecto. Por el contrario, tareas demasiado específicas no hubiesen dejado flexibilidad para adoptar los cambios que fueron necesarios. Entre estos cambios destacan el empleo de bases de datos diferentes a

las previstas y el reconocimiento de los aminoácidos clave basado en alineamientos estructurales.

No obstante, podría arguirse que una mejor metodología no habría planteado el objetivo 2 del trabajo sin conocer el resultado del objetivo 1. En efecto, la utilidad del objetivo 2 dependía en gran medida de sí había relación entre las variables estudiadas. Así, de no existir tal relación (como finalmente sucedió) el tiempo dedicado al objetivo 2 podría haberse dedicado a otro tipo de actividades, como estudios adicionales sobre el objetivo 1.
Frente a esto, esto puede argumentarse que:

i) El tiempo dedicado al objetivo 2 –un total de 20 horas de trabajo– es muy inferior las más de 115 horas de trabajo dedicados al objetivo 1.

ii) El código creado, al ser de tipo modular, puede utilizarse para proyectos equivalentes.

iii) El test de calidad fue diseñado para ser más permisivo a medida que el modelo lineal generado tuviese mayor grado de dispersión. Es decir, el test de calidad hubiese servido incluso con modelos lineales que mostraran una relación muy moderada entre las variables.

iv) La proposición del enfoque del objetivo 2 para un test de calidad es por sí un elemento de valor del TFM. Se trata de un enfoque que evalúa la calidad de las simulaciones sin tener a disposición la estructura real. Esta táctica, empleando otras variables, podría ser utilizada en otras investigaciones.

3.5 Líneas de trabajo futuras

En primer lugar, sería interesante cómo se ha comentado en el capítulo 3.1 refinar el cálculo de los ángulos chi de equilibrio. A continuación, habría que comprobar si se puede hallar una relación entre las variables de estudio.

Igualmente, podrían desarrollarse estrategias análogas a las de este TFM con el objeto de evaluar las simulaciones de complejos proteicos. Se trata de valorar simulaciones de complejos sin contar con la estructura experimental del complejo en cuestión. En este caso, se ha demostrado que la isomeria conformacional no es un buen candidato para dicho propósito. No obstante, podrían probarse otros parámetros. Por ejemplo, podría tratar de hallarse la relación entre cambios de posición en el esqueleto polipéptidico al formarse el complejo y datos termodinámicos. Es decir, la filosofía planteada puede servir de ejemplo para poder finalmente desarrollar un test de de simulaciones de complejos proteicos.

4. Glosario

Ángulo chi: ángulo de rotación de enlace chi.

Cadenas R: cadenas laterales de los aminoácidos.

Enlaces chi: Enlace decadena R de aminoácido que presenta isomería conformacional.

Ligando bound (l_b): Estructura de una de las proteínas del complejo cuando está formando parte del complejo.

Receptor bound (r_b): Estructura de la segunda de las proteínas del complejo cuando está formando parte del complejo.

Ligando unbound (l_u): Estructura de una de las proteínas del complejo cuando no está formando parte del complejo.

Receptor unbound (r_u): Estructura de la segunda de las proteínas del complejo cuando está formando parte del complejo.

Residuos *clave*: residuo de una proteína que cambia de posición de forma apreciable al formarse el complejo proteico. Un residuo *clave* debe contener al menos un átomo entre el 10% de los átomos de la proteína que más cambian de posición al formarse el complejo.

$\lambda_{aa.chi}$: Diferencia entre un ángulo chi y ese mismo ángulo chi en el sistema de referencia de estabilidad.

λ_{aa} : Sumatorio de los valores $\lambda_{aa.chi}$ de un residuo.

λ : Sumatorio de los valores λ_{aa} de los residuos clave de una proteína.

$\Delta\lambda$: Diferencia entre la suma de λ de Ligando y Receptor en estado unbound y la suma de λ de Ligando y Receptor en estado bound.

5. Bibliografía

[1] Peterson, L., Kang, X. and Kihara, D. (2014). Assessment of protein side-chain conformation prediction methods in different residue environments.
Proteins: Structure, Function, and Bioinformatics, 82(9), pp.1971-1984.

[2] Ozlem, K., Tuncbag, N. and Gursoy, A. (2016). Predicting Protein−Protein Interactions from the Molecular to the Proteome Level. *Chemical Reviews*, 116(8), pp.4884–4909.

[3] Maximova, T., Moffatt, R., Ma, B., Nussinov, R. and Shehu, A. (2016). Principles and Overview of Sampling Methods for Modeling Macromolecular Structure and Dynamics. *PLOS Computational Biology*, 12(4), p.e1004619.

[4] Kumar, M. (2006). PINT: Protein-protein Interactions Thermodynamic Database.
Nucleic Acids Research, *34*(90001), D195-D198.
http://dx.doi.org/10.1093/nar/gkj017

[5] Wang, R., Fang, X., Lu, Y., Yang, C., & Wang, S. (2005). The PDBbind Database: Methodologies and Updates.
Journal Of Medicinal Chemistry, *48*(12), 4111-4119.
http://dx.doi.org/10.1021/jm048957q

[6] Jiménez-García, B., & Fernández-Recio, J. (2015). *LightDock: a novel protein-protein docking framework for the new challenges in the interactomics era*. Presentation, XXXVIII Congreso SEBBM. València.

[7] Nic, M., Jirat, J., & Kosata, B. (2017). *IUPAC Gold Book* - Retrieved 17 May 2017, from http://goldbook.iupac.org/

[8] Khalilian, M., Khosravi, H., & Mirzaei, S. (2016). Modified Newman projections: A new representation of the Newman notations to convey conformational properties.
Educación Química, *27*(4), 269-277.
http://dx.doi.org/10.1016/j.eq.2016.02.003

[9] Rastelli, A., Cocchi, M., & Schiatti, E. (1991). Model calculations of chemical interactions. Part 1.—Intramolecular interactions and rotational barriers.
J. Chem. Soc., Faraday Trans., 86(5), 777-781.
http://dx.doi.org/10.1039/ft9908600777

[10] Thiehoff, C., Rey, Y., & Gilmour, R. (2016). The Fluorine Gauche Effect: A Brief History.
Israel Journal Of Chemistry, *57*(1-2), 92-100.
http://dx.doi.org/10.1002/ijch.201600038

[11] Kirys, T., Ruvinsky, A., Tuzikov, A., & Vakser, I. (2012). Rotamer libraries and probabilities of transition between rotamers for the side chains in protein-protein binding. Proteins: Structure, Function, And Bioinformatics, 80(8),2089-2098.
http://dx.doi.org/10.1002/prot.24103

[12] Zhang, L., & Buck, M. (2013). Molecular Simulations of aDynamic Protein Complex: Role of Salt-Bridges and Polar Interactions in Configurational Transitions.
Biophysical Journal, *105*(10), 2412-2417.
http://dx.doi.org/10.1016/j.bpj.2013.09.052

[13] Cazals, F., & Kornprobst, P. (2013). *Modeling in Computational Biology and Biomedicine: A Multidisciplinary Endeavor* (1st ed., pp. 20-26). Springer.

[14] Nero, T., Morton, C., Holien, J., Wielens, J., & Parker, M. (2014). Oncogenic protein interfaces: small molecules, big challenges.
Nature Reviews Cancer, *14*(4), 248-262.
http://dx.doi.org/10.1038/nrc3690

[15] Bisswanger, H. (2008). *Enzyme kinetics* (2nd ed.). Weinheim: Wiley.

[16] Kastritis, P., & Bonvin, A. (2012). On the binding affinity of macromolecular interactions: daring to ask why proteins interact.
Journal Of The Royal Society Interface, *10*(79), 20120835-20120835.
http://dx.doi.org/10.1098/rsif.2012.0835

[17] Jha, R., Gaiotto, T., Bradbury, A., & Strauss, C. (2014). An improved rotein G with higher affinity for human/rabbit IgG Fc domains exploiting computationally designed polar network.
Protein Engineering Design And Selection, *27*(4), 127-134.
http://dx.doi.org/10.1093/protein/gzu005

[18] Faraway, J. (2015). *Linear models with R* (1st ed.). Boca Raton [etc.]: CRC PRESS.

[19] Carmona, F. (2005). *Modelos lineales*. Edicions Universitat Barcelona.

[20] Nakamura, H. (2013). Protein Data Bank (PDB) for Big Data Era. *Seibutsu Butsuri*, *53*(1), 044-046.
http://dx.doi.org/10.2142/biophys.53.044

[21] Westbrook, J., Shao, C., Feng, Z., Zhuravleva, M., Velankar, S., & Young, J. (2014). The chemical component dictionary: complete descriptions of constituent molecules in experimentally determined 3D macromolecules in the Protein Data Bank.
Bioinformatics, *31*(8), 1274-1278.
http://dx.doi.org/10.1093/bioinformatics/btu789

[22] O'Boyle, N., Banck, M., James, C., Morley, C., Vandermeersch, T., & Hutchison, G. (2011). Open Babel: An open chemical toolbox.
Journal Of Cheminformatics, *3*(1), 33.
http://dx.doi.org/10.1186/1758-2946-3-33

[23] Douguet, D., Chen, H., Tovchigrechko, A., & Vakser, I. (2006). DOCKGROUND resource for studying protein-protein interfaces.
Bioinformatics, *22*(21), 2612-2618.
http://dx.doi.org/10.1093/bioinformatics/btl447

[24] Vreven, T., Moal, I., Vangone, A., Pierce, B., Kastritis, P., & Torchala, M. et al. (2015). Updates to the Integrated Protein–Protein Interaction Benchmarks: Docking Benchmark Version 5 and Affinity Benchmark Version 2.
Journal Of Molecular Biology, *427*(19), 3031-3041.
http://dx.doi.org/10.1016/j.jmb.2015.07.016

[25] Python Core Team (2015). Python: A dynamic, open source programming language.
Python Software Foundation. URL https://www.python.org/.

[26] R Core Team (2014). R: A language and environment for statisticalcomputing. R Foundation for Statistical Computing, Vienna, Austria.
http://www.R-project.org/

[27] Martínez, L., Andreani, R., & Martínez, J. (2007). Convergent algorithms for protein structural alignment.
BMC Bioinformatics, *8*(1), 306.
http://dx.doi.org/10.1186/1471-2105-8-306

[28] Daya, S. (2002). Fisher exact test.
Evidence-Based Obstetrics & Gynecology, *4*(1), 3-4.
http://dx.doi.org/10.1054/ebog.2002.0026

6. Anexos

ANEXO IA –

MÓDULO EXTRAER_INFORMACION_BASES_DE_DATOS.PY:

```
import os

def transformacion_de_unidades_a_nm(val):
    "Devuelve el valor en nanomolar a partir de datos en fM, uM .. (pe
    '2uM')"

    #Parte numerica de val:
    numer = ""

    #Indice para controlar las posiciones de val:
    ind = 0

    #Variable auxiliar:
    auxil = 0

    #Bucle para ir construyendo el valor:
    while auxil == 0:
        #Si se esta en la parte numerica de val:
        if val[ind] in ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", r"."]:
            numer = numer + val[ind]
            ind = ind + 1
        else:
            auxil = 1
            numer = float(numer)
            #Si se trata de milimolar:
            if val[ind] == "m":
                numer = numer * 1000000
            #Si se trata de micromolar:
            if val[ind] == "u":
                numer = numer * 1000
            #Si se trata de picomolar:
            if val[ind] == "p":
                numer = numer / 1000
            #Si se trata de fentomolar:
            if val[ind] == "f":
                numer = numer / 1000000

    return(numer)
```

```
def coincidencias_fichero_termodinamico_y_
estructuras_pdb_benchmark(ruta_fichero_termodinamico,
ruta_absoluta_directorio_benchmark_structures):
  "Devuelve coincidencias entre benchmark y la base de datos te
rmodinamica"

#Fichero base de datos termodinamicos tipo INDEX_general_PP.2016
PDBBind:
    f=open(ruta_fichero_termodinamico,"r")
    lis = f.readlines()
    f.close()

    #Lista de ficheros pdb en directorio de benchmark structures:
    ficheros_pdb_benchmark_structures = []
    for f in os.listdir(ruta_absoluta_directorio_benchmark_structures):
        ficheros_pdb_benchmark_structures.append(f)

    #Lista de coincidencias:
    coinc = []

    for i in lis:
        #Si hay coincidencia del elemento (codigo PDB) y todavia no ha
        #sido registrado entonces se registra en coinc:
        for j in ficheros_pdb_benchmark_structures:
            if ((i[0:4].upper() in j) and (i[0:4] not in coinc)):
                coinc.append(i[0:4])

    return(coinc)



def    coincidencias_fichero_termodinamico_y
_fichero_dockground(ruta_fichero_termodinamico,
ruta_fichero_dockground):
  "Devuelve coincidencias la base de datos termodinamica y
DOCKGROUND"

 #Fichero base de datos termodinamicos tipo INDEX_general_PP.2016
PDBBind:
    f=open(ruta_fichero_termodinamico,"r")
    lis2 = f.readlines()
    f.close()

    #Fichero Dockground:
    f=open(ruta_fichero_dockground,"r")
    lis = f.readlines()
    f.close()

    #Lista de coincidencias:
    coinc = []
```

```
    for i in lis:
        for j in lis2:
            if i[0:4] in j:
                coinc.append(i[0:4])

    return(coinc)



def extraer_cte_ disociacion(lista_complejos_pdb,
ruta_fichero_termodinamico):
"Devolver lista de tuplas PDB-Kd a partir de una lista de complejos PDB"

  #Fichero base de datos termodinamicos tipo INDEX_general_PP.2016
PDBBind:
    f=open(ruta_fichero_termodinamico,"r")
    lista1 = f.readlines()
    f.close()

    #Lista de tuplas:
    tuplas = []

    #Constante de disociacion
    kd = ""

    #Indice para controlar las posiciones para leer el dato de Kd:
    indice = 21

    #Bucle para obtener las tuplas:
    for i in lista_complejos_pdb:
        for j in lista1:
            if i in j:
                #Asegurar que existe una kd:
                if "Kd=" in j:
                    #Variable auxiliar:
                    aux = 0
                    #Bucle para ir construyendo kd:
                    while aux == 0:
                        #Si no se ha llegado al final del dato de kd:
                        if j[indice] != "M":
                            kd = kd + j[indice]
                            indice = indice + 1
                        else:
                            #Transformacion de unidades a nm (y ademas se
                            #transforma kd de string a float):
                            kd = transformacion_de_unidades_a_nm(kd)
                            #Nuevo elemento en la lista de tuplas:
                            tuplas.append((i, kd))
                            #Reseteo de kd:
                            kd = ""
```

```
                    #Reseteo de indice:
                    indice = 21
                    #Cambio de aux:
                    aux = 1

    return(tuplas)
```

ANEXO IB –

MÓDULO ALINEAMIENTOS_ESTRUC.PY:

```
import os
from collections import deque

def calcular(tuplas_trabajo, ruta_directorio_de_output, ruta_ejecutable_lovoalign, ruta_directorio_ficheros_benchmark):
    '''Calcula los alineamientos estructurales para cada par bound/unbound'''

    #ATENCION: Las rutas de los directorios recibidos deben acabar en "\".

    #Se van a calcular los alineamientos para las 4 posibles combinaciones:

    #a) Calculo de alineamientos estructurales del ligando en estado bound con el
    #ligando en estado unbound:
    print "Calculando alineamientos estructurales (Paso 1/4) ..."
    for i in tuplas_trabajo:
        j = i[0]
        j = j.upper()

        primera_parte_comando = "./" + ruta_ejecutable_lovoalign + " -p1 " + ruta_directorio_ficheros_benchmark + j + "_l_b.pdb" + " -p2 " + ruta_directorio_ficheros_benchmark + j
                    segunda_parte_comando = "_l_u.pdb" + " -o " + ruta_directorio_de_output + j + "l_b.pdb -rmsf " + ruta_directorio_de_output + "rmsf_" + j + "_l_b.dat" + " -all"
        comando = primera_parte_comando + segunda_parte_comando

        os.system(comando)
        print ".."


    #b) Calculo de alineamientos estructurales del ligando en estado unbound con el
    #ligando en estado bound:
    print "Calculando alineamientos estructurales (Paso 2/4) ..."
    for i in tuplas_trabajo:
        j = i[0]
        j = j.upper()

        primera_parte_comando = "./" + ruta_ejecutable_lovoalign + " -p1 " + ruta_directorio_ficheros_benchmark + j + "_l_u.pdb" + " -p2 " + ruta_directorio_ficheros_benchmark + j + "_l_b.pdb"
```

```
        segunda_parte_comando = " -o " + ruta_directorio_de_output + j +
"l_u.pdb -rmsf " + ruta_directorio_de_output + "rmsf_" + j + "_l_u.dat" + "
-all"
        comando = primera_parte_comando + segunda_parte_comando

        os.system(comando)
        print ".."


     #c) Calculo de alineamientos estructurales del receptor en estado
bound con el
    #receptor en estado unbound:
    print "Calculando alineamientos estructurales (Paso 3/4) ..."
    for i in tuplas_trabajo:
        j = i[0]
        j = j.upper()

        primera_parte_comando = "./" + ruta_ejecutable_lovoalign + " -p1 "
+ ruta_directorio_ficheros_benchmark + j + "_r_b.pdb" + " -p2 " +
ruta_directorio_ficheros_benchmark + j
                    segunda_parte_comando = "_r_u.pdb" + " -o " +
ruta_directorio_de_output + j + "r_b.pdb -rmsf " +
ruta_directorio_de_output + "rmsf_" + j + "_r_b.dat" +" -all"
                    comando = comando = primera_parte_comando +
segunda_parte_comando

        os.system(comando)
        print ".."

     #d) Calculo de alineamientos estructurales del receptor en estado
unbound con el
    #receptor en estado bound:
    print "Calculando alineamientos estructurales (Paso4/4) ..."
    for i in tuplas_trabajo:
        j = i[0]
        j = j.upper()

        primera_parte_comando = "./" + ruta_ejecutable_lovoalign + " -p1 "
+ ruta_directorio_ficheros_benchmark + j + "_r_u.pdb" + " -p2 " +
ruta_directorio_ficheros_benchmark + j + "_r_b.pdb"
        segunda_parte_comando =  " -o " + ruta_directorio_de_output + j +
"r_u.pdb -rmsf " + ruta_directorio_de_output + "rmsf_" + j + "_r_u.dat" +"
-all"
        comando = primera_parte_comando + segunda_parte_comando

        os.system(comando)
        print ".."
```

```
def analizar(ruta_script_auxiliar_R, ruta_fich_resultado_alin,
ruta_fich_output):
    '''Obtener residuos que cambien su posicion al formarse el complejo'''

    #Esta funcion va a utilizar el script auxiliar
    #"codigo_en_R_para_analizar_alineamiento_estructural.R", al que se
    #le envian como parametros las rutas de los ficheros de input y output:
    com_a_ejecutar = "Rscript " + ruta_script_auxiliar_R + " " +
     ruta_fich_resultado_alin + " " + ruta_fich_output

    #Ejecucion del script:
    os.system(com_a_ejecutar)

    fich_output_del_analisis = open(ruta_fich_output, "r")
    residuos_relevantes = fich_output_del_analisis.read().splitlines()
    return(residuos_relevantes)
```

ANEXO IC –

MÓDULO FUNC_LAMBDA.PY:

```
import sys
import math


def leer_fichero_angulos_chi_equilibrio():
    '''Crear una lista de listas con la informacion de
    angulos_chi_equilibrio'''

    #A partir del fichero angulos_chi_equilibrios se va a crear una
    #lista de listas denominada "informacion". Cada elemento de
    #"informacion" es una lista que contendra en primer lugar
    #la abreviatura en mayusculas de aminoacido, luego su
    #angulo chi x2, luego su angulo chi x3, luego su angulo chi x1
    #y por ultimo su angulo chi x4.

    g = open("angulos_chi_equilibrio", "r")

    informacion = []

    lista_lineas_del_fichero = g.read().splitlines()


    for i in lista_lineas_del_fichero:
        elemento = []
        elemento.append(i[0:3])

        #Para empezar a leer cada linea desde los angulos chi:
        indice = 11

        #Para contar los angulos chi registrados:
        contador = 0

        while (contador < 4):
            #Si dentro de dos posiciones hay una "N" entonces el
            #angulo chi es "None":
            if (i[indice] == "N"):
                elemento.append("None")
                #Posicionamiento del indice en el siguiente angulo chi:
                if ((indice + 12) < len(i)):
                    indice = indice + 12
                else:
                    break
            else:
                #En la variable "almacen" se va a ir construyendo el
                #angulo chi en cuestion:
                almacen = ""
```

```
            while ((i[indice] != ",") and (i[indice] != "}")):
                almacen = almacen + i[indice]
                if ((indice + 1) < len(i)):
                    indice = indice + 1
                else:
                    break

            #Pasar de string a float:
            almacen = float(almacen)
            #Anadir a elemento:
            elemento.append(almacen)
            #Desplazar indice:
            if ((indice + 8) < len(i)):
                indice = indice + 8
            else:
                break

        contador = contador + 1
    informacion.append(elemento)

g.close()

return informacion



def contribucion_residuo(nombre_res, angulos_chi, informac):
    '''Calcular la contribucion a lambda de un residuo'''

    #El primer parametro de entrada ha de ser la abreviatura del
    #aminoacido en cuestion en 3 letras (y mayusculas)

    #El segundo parametro de entrada ha de ser un diccionario gene-
    #rado por la funcion predictor.calculate_chi_angles . Contiene
    #los angulos chi obtenidos del residuo a estudiar.

    #El tercer parametro de entrada ha de ser una lista de listas
    #retornada por la funcion leer_fichero_angulos_chi_equilibrio .
    #Contiene los angulos chi de equilibrio de cada uno de los
    #20 aminoacidos.

    claves_angulos_chi = ["x2", "x3", "x1", "x4"]
    contribucion = 0.0
    #La variable i sera una lista con el nombre del residuo primero
    #y los angulos chi de equilibrio caracteristicos a continuacion:
    for i in informac:
        #Indice para controlar las posiciones de la variable i:
        indi = 1
        #Si se halla el residuo a comparar:
```

```
if (nombre_res == i[0]):
    #Calculo para cada chi:
    for j in claves_angulos_chi:
        #if (angulos_chi[j] no es ningun numero por no existir ese
        #ese angulo chi:
        if (angulos_chi[j] != "None") & (angulos_chi[j] is not None):
            #Primer caso:
            if ((angulos_chi[j] >= 0) and (i[indi] >= 0)):
                distancia = abs(angulos_chi[j] - i[indi])
                contribucion  =  contribucion + distancia
            #Segundo caso:
            elif ((angulos_chi[j] <= 0) and (i[indi] <= 0)):
                distancia = abs(angulos_chi[j] - i[indi])
                contribucion  =  contribucion + distancia
            #Tercer caso:
            else:
                #Si no se han dado los casos anteriores
                #se ha de tener un angulo positivo y
                #otro negativo. El calculo entonces de la
                #distancia es similar a calcular la distancia
                #minima entre dos agujas de un reloj, estando una
                #en el hemisferio derecho del reloj y la otra
                #en el izquierdo. Las "12" serian -180 grados
                #(o 180), las "6" equivaldria a un angulo de 0
                #grados, "menos cuarto" e "y cuarto" serian -90
                #y 90 grados respectivamente:
                if (angulos_chi[j] > 0):
                    #Si se ha dado esta condicion es que i[indi] < 0:
                    arco_inferior_reloj = (180 - angulos_chi[j]) + (180 +
                     i[indi])
                    arco_superior_reloj = (0 + angulos_chi[j]) + (0 - i[indi])
                    if (arco_inferior_reloj <= arco_superior_reloj):
                        #Es decir, por ejemplo las "9 y 5 min":
                        distancia = (180 - angulos_chi[j]) + (180 + i[indi])
                        contribucion  =  contribucion + distancia
                    else:
                        #Es decir, por ejemplo las "8 y 20 min":
                        distancia = (0 + angulos_chi[j]) + (0 - i[indi])
                        contribucion  =  contribucion + distancia
                else:
                    #Si se ha dado esta condicion es que i[indi] > 0.
                    #El codigo es como el anterior intercambiando
                #el orden de angulos_chi[j] y i[indi]:
                arco_inferior_reloj = (180 - i[indi]) + (180 + angulos_chi[j])
                arco_superior_reloj = (0 + i[indi]) + (0 - angulos_chi[j])
                if (arco_inferior_reloj <= arco_superior_reloj):
                    #Es decir, por ejemplo las "8 y 5 min":
                    distancia = (180 - i[indi]) + (180 + angulos_chi[j])
                    contribucion  =  contribucion + distancia
                else:
```

```
                    #Es decir, por ejemplo las "8 y 20 min":
                    distancia = (0 + i[indi]) + (0 - angulos_chi[j])
                    contribucion  =  contribucion + distancia

                     #Para pasar al siguiente angulo chi:
                indi = indi + 1

            #Como ya se ha hallado el residuo a comparar:
            break

        return contribucion
```

ANEXO ID –

MÓDULO PRIMER_INDICE_RES.PY:

```
def buscar_primer_indice(ruta_fich):
    '''Busca el indice del primer residuo del fichero .pdb'''

    indice_res = ""
    f = open(ruta_fich, "r")
    primera_linea = f.readline()
    #Contador de secciones en blanco:
    contador = 0
    #Controlador para desplazarse por la linea:
    con = 0
    #Senal para indicar si se ha identificado ya el indice:
    senal_fin = 0
    #Senal para saber si el indice estaba en la quinta columna del
    #fichero .pdb y no en la sexta:
    senal_indice_no_en_columna_6 = 0

    #Recorrer la linea:
    while senal_fin == 0:
        #Si no es el primer caracter:
        if con != 0:
            #Detectar seccion en blanco:
            if (primera_linea[con] == " ") & (primera_linea[con - 1] != " "):
                contador = contador + 1

        #Si contador == 4 y el fichero PDB no tenia identificador de cadena
        #entonces se puede estar ante el indice del residuo:
        if contador == 4:
            #Dado que el identificador de cadena es un caracter (no es "int"):
            if primera_linea[con] in ["0","1","2","3","4","5","6","7","8","9"]:
                indice_res = indice_res + primera_linea[con]
                senal_indice_no_en_columna_6 = 1


        #Si contador == 5 y primera_linea[con] != " " se esta
        #en el indice del residuo, siempre que no estuviera
        #en la columna anterior:
        if (contador == 5) & (primera_linea[con] != " "):
            if senal_indice_no_en_columna_6 == 0:
                indice_res = indice_res + primera_linea[con]

        #Si ya se ha identificado el indice:
        if (contador > 5):
            senal_fin = 1

        con = con + 1
```

```
indice_res = int(indice_res)
return (indice_res)
```

ANEXO IE –

MÓDULO PARTE1.PY:

```
from lightdock.pdbutil.PDBIO import parse_complex_from_file
from lightdock.structure.complex import Complex
from lightdock.rotamer.predictor import calculate_chi_angles
import sys
import os
import extraer_informacion_bases_de_datos
import alineamientos_estruc
import func_lambda
import primer_indice_res

try:
    f = open("datos_input", "r")
except:
        print "Este programa necesita de un fichero de input llamado datos_input localizado en el mismo directorio que este script de Python. En dicho fichero, han de incluirse los siguientes datos, uno en cada linea: \nRuta del fichero termodinamico de tipo INDEX_general_PP.2016 de la base de datos PDBbind. \nRuta del directorio que contiene las estructuras .pdb de tipo de la base de datos benchmark. \nRuta del directorio que contiene los modulos de Python y scripts de R en general utilizados, asi como el fichero de angulos chi de equilibrio. \nRuta del script de R en particular para analizar los alineamientos estructurales. \nRuta del ejecutable de lovoalign.\nAtencion: las rutas de los directorios deben acabar en barra invertida (backslash)."
    sys.exit()

datos_fich_input = f.read().splitlines()
f.close()


#Ruta del fichero termodinamico:
f_ter = datos_fich_input[0]
#Las rutas de los directorios deben acabar en "/":
#Ruta del directorio de estructuras pdb benchmark:
d_ben = datos_fich_input[1]
#El directorio de trabajo (d_trab) contiene los modulos de Python y R:
d_trab = datos_fich_input[2]
#Ruta del codigo en R auxiliar para analizar alineamiento estructural:
r_lin = datos_fich_input[3]
#Ruta del ejecutable de lovoalign. Nota: Si al ejecutar el programa se
#observa error de Fortran "Sementation fault" entonces acudir al directorio
#de lovoalign => src/sizes.f90 y disminuir maxatom por ejemplo a 3500
#y maxfile a 5000, volver a compilar y probar con el nuevo ejecutable. Sin
```

```
#embargo, es posible que asi queden algunos complejos sin analizar por
superar
#o bien el ligando o bien el receptor ese numero de atomos.
r_lov = datos_fich_input[4]



#Variable para indicar el numero de complejos de los que no ha podido
#haber todos los alineamientos estructurales:
senal = 0
#Contador de complejos analizados:
contad = 0
#Lista de claves de "ligando_bound", "ligando_unbound",
"receptor_bound"
#y "receptor_unbound":
l_clav = ["l_b", "l_u", "r_b", "r_u"]
#Variable para contener los angulos chi mas estables de los
#residuos individualmente:
chi_equ = func_lambda.leer_fichero_angulos_chi_equilibrio()
#c va a contener las coincidencias de las bases de datos:
c = []
#tuplas va a contener las coincidencias de las bases de datos con la
#constante de disociacion especifica de cada una:
tuplas = []
#resul_fin va a ser otra lista de tuplas. En este caso va a contener los
#pares valor de lambda del complejo-constante de disociacion del
complejo:
resul_fin = []



#Extraccion de coincidencias de las bases de datos:
print "Extrayendo coincidencias de las bases de datos.."
c = extraer_infor_bases_de_datos.coincidencias_
fichero_termodinamico_y_estructuras_pdb_benchmark(f_ter, d_ben)



#Extraccion de informacion termodinamica (ctes de disociacion) de
#las coincidencias.."
print "\nExtrayendo informacion termodinamica de las coincidencias.."
tuplas = extraer_infor_bases_de_datos.extraer_cte_disociacion(c, f_ter)



#Si no existe el fichero de angulos chi de equilibrio en el directorio
#de trabajo se avisa al usuario y se termina el programa:
listar_elementos_directorio_trabajo = "ls " + d_trab
elementos_directorio_trabajo =
os.popen(listar_elementos_directorio_trabajo).read()
if "angulos_chi_equilibrio" not in elementos_directorio_trabajo:
```

```
    print "No existe el fichero de angulos chi de equilibrio. Para crearlo
use el script
calculador_de_chi_angles_de_equilibrio_para_cada_aminoacido.py"
    sys.exit()

#Si no existe un directorio llamado archivos_temporales (corresponde a
#los alineamientos estructurales) en el directorio de trabajo se crea:
if "archivos_temporales" not in elementos_directorio_trabajo:
    crear_directorio = "mkdir " + d_trab + "archivos_temporales"
    os.system(crear_directorio)
    print "\nAlineamientos estructurales (15 min para el caso estandar)"
    os.system("sleep 15")
      #La siguiente linea tarda unos 15 min en ejecutarse para el caso
estandar:
      alineamientos_estruc.calcular(tuplas, "archivos_temporales/", r_lov,
d_ben)

#Si no existe un directorio llamado "otros_fich_temps" (contiene ficheros
#generados por el script de R de analisis de los alineamientos
estructurales)
#en el directorio de trabajo se crea:
if "otros_fich_temps" not in elementos_directorio_trabajo:
    crear_directorio = "mkdir " + d_trab + "otros_fich_temps"
    os.system(crear_directorio)



print "\n\n\nAnalizando resultados de los alineamientos (3 min)"

#Contador de complejos satisfactoriamente analizados:
comp_analizados = 0

#Para cada una de las coincidencias entre las bases de datos:
for i in tuplas:
    #Si no se han podido realizar previamente los cuatro alineamientos
    #del complejo
    #(es decir, si no existen todos los ficheros .dat correspondientes) se
    #salta
    #esa iteracion (ademas mediante la variable senal se cuentan dichos
    #saltos):
            aux1 = d_trab + "archivos_temporales/" + "rmsf_" + i[0].upper()
        + "_l_b.dat"
            aux2 = d_trab + "archivos_temporales/" + "rmsf_" + i[0].upper()
        + "_l_u.dat"
            aux3 = d_trab + "archivos_temporales/" + "rmsf_" + i[0].upper()
        + "_r_b.dat"
            aux4 = d_trab + "archivos_temporales/" + "rmsf_" + i[0].upper()
        + "_r_u.dat"
            if (os.path.isfile(aux1) & os.path.isfile(aux2) &
        os.path.isfile(aux3) & os.path.isfile(aux4) == False):
```

```
        senal = senal + 1
        continue


    #Si en cambio se han podido realizar los cuatro alineamientos del
     #complejo:

    #Se actualiza primero el contador del complejo que se esta analizado:
    contad = contad + 1
    print "Analizando complejo " + str(contad) + ".."

    #Reseteo de los valores de lambda para cada una de las cuatro
    #proteinas:
    lam_ligando_bound = 0.0
    lam_ligando_unbound = 0.0
    lam_receptor_bound = 0.0
    lam_receptor_unbound = 0.0

    #Senal para saber si el complejo ha podido ser analizado (un valor
    #igual a
    #cero indica que todo va funcionando):
    senal_analisis_comp = 0
    #Para cada una de las cuatro proteinas en cuestion("ligando_bound",
    #"ligando_unbound", "receptor_bound" y "receptor_unbound"):
    for j in l_clav:
        #Ruta del fichero pdb de la proteina en cuestion:
        ruta_pdb = d_ben + i[0].upper() + "_" + j + ".pdb"
        #Generacion del objeto Complex:
        atoms, residues, chains = parse_complex_from_file(ruta_pdb)
        try:
            proteina_complejo = Complex(chains, atoms)
        except:
            senal_analisis_comp = 1

        #Ruta del fichero con los resultados del alineamiento de la proteina
        #en cuestion con su par bound/unbound:
        alin = d_trab + "archivos_temporales/" + "rmsf_" + i[0].upper() + "_"
          + j + ".dat"
        #res_cam va a contener los indices de lor residuos que cambian de
        #posicion en la proteina en cuestion:
        res_cam = alineamientos_estruc.analizar(r_lin, alin, d_trab +
          "otros_fich_temps/" + i[0] + j + ".out")

        #Adaptacion de res_cam a la nomenclatura de indices de
        #lightdock.structure.complex. Como la indexacion en
        #lightdock.structure.complex es 0, 1, 2, ... es necesario
        #restar la diferencia existente. Ademas, en este paso
        #se va a convertir los elementos de res_cam de strings a integers:
        try:
            diferencia = primer_indice_res.buscar_primer_indice(ruta_pdb)
```

```
        w = 0
        #print res_cam
        while w < len(res_cam):
            res_cam[w] = int(res_cam[w]) - diferencia
            w = w + 1
        #print res_cam
              except:
                  senal_analisis_comp = 1


    #Calculo de la contribucion de lambda de la proteina en cuestion.
    #Para ello, se van analizando los angulos chi de cada uno de
    #los residuos indexados por res_cam:

    #Contador de residuos que no han podido ser analizados. Si supera
    #el umbral de 1/10 de los residuos que cambian de esa proteina
    #entonces el conjunto de
    #4 alineamientos al que pertenece la proteina no es contabilizado
    #(senal_analisis_comp valdria 1):
    residuos_con_errores = 0
    #Calculo de la contribucion de lambda de la proteina en cuestion:
    for h in res_cam:
        #Senal de que ha habido un error analizando ese residuo:
        senal_error = 0
        #Creacion del objeto residuo:
        try:
            residuo = proteina_complejo.residues[h]
        except:
            residuos_con_errores = residuos_con_errores + 1
            senal_error = 1
        #Calculo de los angulos chi del residuo:
        try:
                    ang_chi = calculate_chi_angles(residuo)
        except:
          if senal_error == 0:
              residuos_con_errores = residuos_con_errores + 1
              senal_error = 1
       #Calculo de la contribucion a lambda del residuo:
        try:
res_la = func_lambda.contribucion_residuo(residuo.name, ang_chi,
     chi_equ)
        except:
if senal_error == 0:
    residuos_con_errores = residuos_con_errores + 1
    senal_error = 1

        #Actualizacion de los valores de lambda de la proteina
        #en cuestion si el residuo ha podido ser analizado:
        if senal_error == 0:
            if j == "l_b":
```

```
                lam_ligando_bound = lam_ligando_bound + res_la
            if j == "l_u":
                lam_ligando_unbound = lam_ligando_unbound + res_la
            if j == "r_b":
                lam_receptor_bound = lam_receptor_bound + res_la
            if j == "r_u":
                lam_receptor_unbound = lam_receptor_unbound + res_la


        #Si para la proteina en cuestion 1/10 de los residuos que debian ser
        #analizados no han podido ser analizados, entonces se descarta el
        #complejo del que forma parte la proteina para el ouput del
         #programa:
        if residuos_con_errores >= (len(res_cam)/10):
            senal_analisis_comp = 1

    #Calculo del valor de lambda del complejo (si todo ha funcionado
    #correctamente):
    if senal_analisis_comp == 0:
            lam_comp = lam_ligando_unbound + lam_receptor_unbound -
(lam_ligando_bound + lam_receptor_bound)
        #Actualizacion de la lista de tuplas de resultados
        #(lam_comp, cte_disociacion del complejo):
        par_lam_comp_cte_disoc = (lam_comp, i[1])
        resul_fin.append(par_lam_comp_cte_disoc)
  #Actualizacion del contador de complejos analizados:
        comp_analizados = comp_analizados + 1

g = open("resultados_output", "w")

g.write("Los pares lambda - cte disoc son:\n")
for i in resul_fin:
    par_resultado = str(i[0]) + " " + str(i[1])
    g.write(par_resultado)
    g.write("\n")

g.close()

#Creación del modelo lineal:

g = open("lambda.txt", "w")

for i in resul_fin:

    g.write(str(i[0]))

    g.write("\n")

g.close()
```

```
g = open("Kd.txt", "w")

for i in resul_fin:

    g.write(str(i[1]))

    g.write("\n")

g.close()



os.system("Rscript codigo_en_R_para_generar_modelo.R")

print "Fichero 'resultados_output' creado."


if senal != 0:
    print "\nAVISO: Solo se han podido procesar finalmente " +
str(comp_analizados) + " de los " + str(len(tuplas)) + " complejos con
coincidencias. Pruebe a aumentar la variable maxatom en el codigo
fuente de Lovoalign (fichero src/sizes.f90). Como contrapartida, puede
que el software no funcione ya que requerira de mas memoria RAM."
```

ANEXO IF –

MÓDULO PARTE2.PY:

```
from lightdock.pdbutil.PDBIO import parse_complex_from_file
from lightdock.structure.complex import Complex
from lightdock.rotamer.predictor import calculate_chi_angles
import sys
import os
import extraer_informacion_bases_de_datos
import alineamientos_estruc
import func_lambda
import primer_indice_res


try:
    f = open("datos_input", "r")
except:
        print "Este programa necesita de un fichero de input llamado datos_input localizado en el mismo directorio que este script de Python. En dicho fichero, han de incluirse los siguientes datos, uno en cada linea: \nRuta del fichero con la informacion de la constante de disociacion que este en el formato de INDEX_general_PP.2016 de la base de datos PDBbind. \nRuta del directorio que contiene las estructuras del ligando y receptor en estados unbound y bound siguiendo la nomenclatura de la base de datos benchmark. \nRuta del directorio que contiene los modulos de Python y scripts de R en general utilizados, asi como el fichero de angulos chi de equilibrio. \nRuta del script de R en particular para analizar los alineamientos estructurales. \nRuta del ejecutable de lovoalign.\nAtencion: las rutas de los directorios deben acabar en barra invertida (backslash)."
    sys.exit()

datos_fich_input = f.read().splitlines()
f.close()


#Ruta del fichero con la constante de disociacion:
f_ter = datos_fich_input[0]
#Las rutas de los directorios deben acabar en "/":
#Ruta del directorio de estructuras pdb:
d_ben = datos_fich_input[1]
#El directorio de trabajo (d_trab) contiene los modulos de Python y R:
d_trab = datos_fich_input[2]
#Ruta del codigo en R auxiliar para analizar alineamiento estructural:
r_lin = datos_fich_input[3]
#Ruta del ejecutable de lovoalign. Nota: Si al ejecutar el programa se
#observa error de Fortran "Sementation fault" entonces acudir al directorio
#de lovoalign => src/sizes.f90 y disminuir maxatom por ejemplo a 3500
```

```
#y maxfile a 5000, volver a compilar y probar con el nuevo ejecutable.
#Sin embargo, es posible que asi queden algunos complejos sin analizar
#por superar
#o bien el ligando o bien el receptor ese numero de atomos.
r_lov = datos_fich_input[4]


#Lista de claves de "ligando_bound", "ligando_unbound",
"receptor_bound"
#y "receptor_unbound":
l_clav = ["l_b", "l_u", "r_b", "r_u"]
#Variable para contener los angulos chi mas estables de los
#residuos individualmente:
chi_equ = func_lambda.leer_fichero_angulos_chi_equilibrio()
#c va a contener las coincidencias de las bases de datos:
c = []
#tuplas va a contener las coincidencias de las bases de datos con la
#constante de disociacion especifica de cada una:
tuplas = []
#resul_fin va a ser otra lista de tuplas. En este caso va a contener los
#pares valor de lambda del complejo-constante de disociacion del
complejo:
resul_fin = []


#Verficiacion de que el complejo indicado en el archivo de la constante
de
#disociacion y el indicado por los ficheros de estructura son los mismos:
print "Comprobando concordancia de los datos.."
c =     extraer_infor_bases_de_datos.coincidencias
_fichero_termodinamico_y_estructuras_pdb_benchmark(f_ter, d_ben)
if len(c) == 0:
    print "El nombre del complejo no coincide."
    sys.exit()


#Extraccion de informacion termodinamica (ctes de disociacion):"
print "\nExtrayendo informacion termodinamica.."
tuplas = extraer_infor_bases_de_datos.extraer_cte_disociacion(c, f_ter)


#Si no existe el fichero de angulos chi de equilibrio en el directorio
#de trabajo se avisa al usuario y se termina el programa:
listar_elementos_directorio_trabajo = "ls " + d_trab
elementos_directorio_trabajo =
os.popen(listar_elementos_directorio_trabajo).read()
if "angulos_chi_equilibrio" not in elementos_directorio_trabajo:
    print "No existe el fichero de angulos chi de equilibrio. Para crearlo
use                el                script
calculador_de_chi_angles_de_equilibrio_para_cada_aminoacido.py"
```

```
    sys.exit()

#Si no existe un directorio llamado fich_temporales (corresponde a
#los alineamientos estructurales) en el directorio de trabajo se crea:
if "fich_temporales" not in elementos_directorio_trabajo:
    crear_directorio = "mkdir " + d_trab + "fich_temporales"
    os.system(crear_directorio)
    print "\nAlineamientos estructurales.."
    os.system("sleep 2")
    alineamientos_estruc.calcular(tuplas, "fich_temporales/", r_lov, d_ben)

#Si no existe un directorio llamado "otros_arch_temps" (contiene ficheros
#generados por el script de R de analisis de los alineamientos
estructurales)
#en el directorio de trabajo se crea:
if "otros_arch_temps" not in elementos_directorio_trabajo:
    crear_directorio = "mkdir " + d_trab + "otros_arch_temps"
    os.system(crear_directorio)



print "\n\n\nAnalizando resultados de los alineamientos.."


for i in tuplas:
    #Si no se han podido realizar previamente los cuatro alineamientos
    #del complejo
    #(es decir, si no existen todos los ficheros .dat correspondientes) se
   #salta sa iteracion (ademas mediante la variable senal se cuentan
   #dichos      saltos):
    aux1 = d_trab + "fich_temporales/" + "rmsf_" + i[0].upper() + "_l_b.dat"
    aux2 = d_trab + "fich_temporales/" + "rmsf_" + i[0].upper() + "_l_u.dat"
    aux3 = d_trab + "fich_temporales/" + "rmsf_" + i[0].upper() + "_r_b.dat"
    aux4 = d_trab + "fich_temporales/" + "rmsf_" + i[0].upper() + "_r_u.dat"
     if (os.path.isfile(aux1) & os.path.isfile(aux2) & os.path.isfile(aux3) &
os.path.isfile(aux4) == False):
        senal = senal + 1
          print "No se han podido realizar los alineamientos estructurales
necesarios."
        sys.exit()


     #Si en cambio se han podido realizar los cuatro alineamientos del
#complejo:

    #Se actualiza primero el contador del complejo que se esta analizado:
    contad = contad + 1
    print "Analizando complejo " + str(contad) + ".."
```

```
    #Reseteo de los valores de lambda para cada una de las cuatro
    #proteinas:
            lam_ligando_bound = 0.0
            lam_ligando_unbound = 0.0
    lam_receptor_bound = 0.0
    lam_receptor_unbound = 0.0

    #Senal para saber si el complejo ha podido ser analizado (un valor
    #igual a
            #cero indica que todo va funcionando):
    senal_analisis_comp = 0
    #Para cada una de las cuatro proteinas en cuestion("ligando_bound",
    #"ligando_unbound", "receptor_bound" y "receptor_unbound"):
    for j in l_clav:
        #Ruta del fichero pdb de la proteina en cuestion:
        ruta_pdb = d_ben + i[0].upper() + "_" + j + ".pdb"
        #Generacion del objeto Complex:
        atoms, residues, chains = parse_complex_from_file(ruta_pdb)
        try:
            proteina_complejo = Complex(chains, atoms)
        except:
            senal_analisis_comp = 1

        #Ruta del fichero con los resultados del alineamiento de la proteina
        #en cuestion con su par bound/unbound:
        alin = d_trab + "fich_temporales/" + "rmsf_" + i[0].upper() + "_" + j +
".dat"
        #res_cam va a contener los indices de lor residuos que cambian de
        #posicion en la proteina en cuestion:
            res_cam = alineamientos_estruc.analizar(r_lin, alin, d_trab +
"otros_arch_temps/" + i[0] + j + ".out")

        #Adaptacion de res_cam a la nomenclatura de indices de
        #lightdock.structure.complex. Como la indexacion en
        #lightdock.structure.complex es 0, 1, 2, ... es necesario
        #restar la diferencia existente. Ademas, en este paso
        #se va a convertir los elementos de res_cam de strings a
       #integers:
        try:
            diferencia = primer_indice_res.buscar_primer_indice(ruta_pdb)
            w = 0
            #print res_cam
            while w < len(res_cam):
                res_cam[w] = int(res_cam[w]) - diferencia
                w = w + 1
            #print res_cam
        except:
            senal_analisis_comp = 1

#Calculo de la contribucion de lambda de la proteina en cuestion.
```

```
#Para ello, se van analizando los angulos chi de cada uno de
#los residuos indexados por res_cam:

 #Contador de residuos que no han podido ser analizados. Si supera el
 #umbral de 1/10 de los residuos que cambian de esa proteina entonces
#el conjunto de
        #4 alineamientos al que pertenece la proteina no es contabilizado
        #(senal_analisis_comp valdria 1):
        residuos_con_errores = 0
        #Calculo de la contribucion de lambda de la proteina en cuestion:
        for h in res_cam:
            #Senal de que ha habido un error analizando ese residuo:
            senal_error = 0
            #Creacion del objeto residuo:
            try:
                residuo = proteina_complejo.residues[h]
            except:
                residuos_con_errores = residuos_con_errores + 1
                senal_error = 1
            #Calculo de los angulos chi del residuo:
            try:
                ang_chi = calculate_chi_angles(residuo)
            except:
              if senal_error == 0:
                  residuos_con_errores = residuos_con_errores + 1
                  senal_error = 1
            #Calculo de la contribucion a lambda del residuo:
            try:
                   res_la = func_lambda.contribucion_residuo(residuo.name,
ang_chi, chi_equ)
            except:
                if senal_error == 0:
                    residuos_con_errores = residuos_con_errores + 1
                    senal_error = 1

            #Actualizacion de los valores de lambda de la proteina
            #en cuestion si el residuo ha podido ser analizado:
            if senal_error == 0:
                if j == "l_b":
                    lam_ligando_bound = lam_ligando_bound + res_la
                if j == "l_u":
                    lam_ligando_unbound = lam_ligando_unbound + res_la
                if j == "r_b":
                    lam_receptor_bound = lam_receptor_bound + res_la
                if j == "r_u":
                    lam_receptor_unbound = lam_receptor_unbound + res_la


        #Si para la proteina en cuestion 1/10 de los residuos que debian ser
        #analizados no han podido ser analizados, entonces se descarta el
```

```
        #complejo del que forma parte la proteina para el ouput del
        #programa:
        if residuos_con_errores >= (len(res_cam)/10):
            senal_analisis_comp = 1

    #Calculo del valor de lambda del complejo (si todo ha funcionado
    #correctamente):
    if senal_analisis_comp == 0:
            lam_comp = lam_ligando_unbound + lam_receptor_unbound -
(lam_ligando_bound + lam_receptor_bound)
        #Actualizacion de la lista de tuplas de resultados
        #(lam_comp, cte_disociacion del complejo):
        par_lam_comp_cte_disoc = (lam_comp, i[1])
        resul_fin.append(par_lam_comp_cte_disoc)
        #Actualizacion del contador de complejos analizados:
        comp_analizados = comp_analizados + 1

#Creacion de uno de los ficheros que va a utilizar el script
#codigo_en_R_para_prueba_de_hipotesis_con_el_modelo
_y_datos_del_proble        ma.R:
g = open("problema.txt", "w")
for i in resul_fin:
    g.write(str(i[1]))
g.write("\n")
g.write(str(i[0]))
g.close()


#Ejecucion del script
#codigo_en_R_para_prueba_de_hipotesis_
con_el_modelo_y_datos_del_probl ema.R:
os.system("Rscript      codigo_en_R_para_prueba_de_hipotesis_con
_el_modelo_y_datos_del_proble ma.R")

#Lectura del output del script ejecutado para concluir sobre el rechazo o
#no rechazo de los
#datos del problema al modelo:
h = open("sobre_el_rechazo_de_la_hipotesis.txt", "r")
output_final = h.read().splitlines()
h.close()
if output_final[0] == "NO":
     print "No se rechaza la hipotesis de que los datos del problema se
ajusten al modelo."
else:
    print "Se rechaza la hipotesis de que los datos del problema se
ajusten al modelo."
```

ANEXO IG –

MÓDULO SOFTWARE_COMPLETO.PY:

```
import os

print "Bienvenido, introduzca (1) si desea solo generar un modelo lineal, (2) si desea testear unos datos frente al modelo generado, o (3) si desea testear unos datos frente a un modelo proporcionado por usted:"

decision = raw_input()

if decision == "1":
    os.system("python parte1.py")

if decision == "2":
    os.system("python parte1.py")
    os.system("python parte2.py")

if decision == "3":
    os.system("python parte2.py")
```

ANEXO IH (1) –

MÓDULO AUXILIAR
TRANSFORMAR_FICHEROS_SDF_IDEAL_A_PDB.PY:

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-

import sys
import os

#abreviaturas es un fichero con las abreviaturas de los aminoácidos:
f = open("abreviaturas", "r")

#Cada elemento de lista contendrá la abreviatura de un aminoácido:
lista = f.read().splitlines()

for i in lista:
        #babel es un comando del software openbabel para transformar
#ficheros:
      comando = r'babel ' + i + r'.sdf ' + i + r'.pdb'
      os.system(comando)


f.close()
```

ANEXO IH (2) –

MÓDULO AUXILIAR CALCULAR_CHI_ANGLES_EQUILIBRIO.PY

```
#Calculo de angulos chi de equilibrio e impresion en fichero de output
#llamado angulos_chi_equilibrio.

import sys
import math

from lightdock.scoring.dfire.driver import DFIREAdapter
from lightdock.pdbutil.PDBIO import parse_complex_from_file
from lightdock.structure.complex import Complex
from lightdock.rotamer.predictor import get_interface_residues,
calculate_chi_angles, steric_energy
from lightdock.rotamer.library import InterfaceSurfaceLibrary


#Para hacer funcionar script es necesario pasarle dos argumentos por
#la linea de comandos. Dichos argumentos son una ruta de fichero y
#una ruta de directorio (explicado mas adelante).


#Ruta de fichero con abreviaturas de los aas (ALA, MET, ...)
#en una columna:
ruta_abreviaturas = sys.argv[1]

f = open(ruta_abreviaturas, "r")
lista_abreviaturas_aas = f.read().splitlines()
f.close()

#Eliminacion del ultimo elemento de la lista (espacio en blanco):
lista_abreviaturas_aas.pop()


#Ruta de directorio (acabado en "\") que contiene el fichero PDB
#de cada aa:
ruta_pdbs = sys.argv[2]


#Preparacion de fichero de output:
g = open("angulos_chi_equilibrio", "w")

#Bucle de calculo:
for i in lista_abreviaturas_aas:
        g.write(i + " ")
        i = ruta_pdbs + i + ".pdb"
        atoms, residues, chains = parse_complex_from_file(i)
        aminoacido = Complex(chains, atoms)
        residue = aminoacido.residues[0]
```

```
        chi_angles = calculate_chi_angles(residue)

        for chi, angle in chi_angles.iteritems():
            try:
                chi_angles[chi] = math.degrees(angle)
            except TypeError:
                pass

        s = str(chi_angles)
        g.write(s + "\n")


g.close()
```

ANEXO IIA –

MÓDULO CODIGO_EN_R_PARA_ANALIZAR_ALINEAMIENTO _ESTRUCTURAL.R:

```
#!/usr/bin/env Rscript

#Este script devuelve los residuos que más han cambiado #significativamente su posición.
#Para ello se va a calcular el percentil 90 de la distribución de RMSFs de los átomos
#y se van a seleccionar los residuos que contengan al menos un átomo con un RMSFs superior
#a dicho percentil.

#Para recibir como comandos a través del terminal:
#i)la ruta del fichero a analizar
#ii)la ruta del fichero de output a crear
rutas_por_comando = commandArgs(trailingOnly=TRUE)
ruta_fichero_a_analizar = rutas_por_comando[1]
ruta_fichero_output = rutas_por_comando[2]

datos_alin <- read.table(ruta_fichero_a_analizar)

percentil90 <- quantile(datos_alin$V2, probs = c(0.90))

#Variable que contendrá el conjunto de residuos con átomos que han cambiado en gran
#propoción su posición:
conjunto_res <- vector()

i <- 1

while (i <= length(datos_alin$V1))
{
  #Si el RMSF del átomo es superior al percentil 90:
  if (datos_alin$V2[i] > percentil90)
  {
    #Si el residuo que contiene dicho átomo no había sido
    #incluído en conjunto_res antes:
    if (is.element(datos_alin$V1[i], conjunto_res) == FALSE)
    {
      conjunto_res <- c(conjunto_res, datos_alin$V1[i])
    }
  }

  i <- i + 1
}
```

```
#Finalmente, se escriben los resultados en un fichero:
#Primero se asegura que el fichero de output ya no exista. Para ello,
#se ejecuta las siguientes dos líneas de código:
comando_para_eliminar_fichero_output_si_existia <- paste("rm ",
ruta_fichero_output)
try(system(comando_para_eliminar_fichero_output_si_existia), silent =
TRUE)

#Luego se crea el nuevo fichero de output:
comando_para_crear_fichero_output <- paste("touch ",
ruta_fichero_output)
system(comando_para_crear_fichero_output)

#A continuación se escribe en el fichero:
for (j in conjunto_res)
{
  write(j, file = ruta_fichero_output,
        append = TRUE, sep = "\n")
}
```

ANEXO IIB –

MÓDULO CODIGO_EN_R_PARA_GENERAR_MODELO.R:

```
#!/usr/bin/env Rscript

#Este script de R leerá los valores de λ y constante de disociación (Kd) a
#partir de los ficheros lambda.txt y kd.txt
#generados por otras partes del software. El fichero lambda.txt debe
#constar de una única
#columna con los valores de lambda para cada proteína. Por su parte, el
#fichero Kd.txt
#ha de recoger en una única columna la constante de disociación
asociada con cada complejo.


#La variable respuesta sería Kd y la variables explicativas serían
lambda, lambda^2 y lambda^3 .

#Lectura de los parámetros:

lectura_lambda <- read.table("lambda.txt")
lambda <- lectura_lambda$V1

lectura_Kd <- read.table("Kd.txt")
Kd <- lectura_Kd$V1

#Creación del modelo lineal:
lambda_cuadrado <- lambda ^2
modelo_lineal <- lm(Kd ~ lambda_cuadrado + lambda)

#A continuación se calculan la desviación típica y coeficientes del
modelo
#y se imprimen en el archivo modelo_lineal.txt:
desviacion_tipica_modelo <- sigma(modelo_lineal)
coeficientes_del_modelo <- as.numeric(modelo_lineal$coefficients)

write(desviacion_tipica_modelo, file = "modelo_lineal.txt")
write(coeficientes_del_modelo[1], file = "modelo_lineal.txt", append =
TRUE)
write(coeficientes_del_modelo[2], file = "modelo_lineal.txt", append =
TRUE)
write(coeficientes_del_modelo[3], file = "modelo_lineal.txt", append =
TRUE)
```

ANEXO IIC –

MÓDULO
CODIGO_EN_R_PARA_PRUEBA_DE_HIPOTESIS_CON_EL_
MODELO_Y_DATOS_DEL_PROBLEMA.R:

```
#!/usr/bin/env Rscript

#Este script de R generará un fichero denominado
#"sobre_el_rechazo_de_la_hipotesis.txt", en el
#que se imprimirá la palabra "SI" o bien la
#palabra "NO". Se dará
#el primer caso si se rechaza que los datos de la
#simulación problema concuerden con los del modelo
#lineal. Si no se rechaza la hipótesis, se dará
#el segundo caso.

#Paso 1:
#Lectura de parámetros del modelo lineal a partir de
#un fichero. Dicho fichero ha de contener en primer
#lugar la desviación típica del modelo lineal y a
#continuación los parámetros β en el orden β0, β1,
#y β2:

parametros_mod_lin <- scan(file = "modelo_lineal.txt",
                  what = double(),  n = -3, sep = "")

#Se guarda la desviación típica del modelo lineal en:
desv_tip_mod_lin <- parametros_mod_lin[1]

#Se guardan los parámetros β del modelo lineal en:
parametros_beta_mod_lin <-
parametros_mod_lin[2:length(parametros_mod_lin)]

#Paso 2:
#Se procede al conteo de parámetros β del modelo lineal:
num_parametros_beta_mod_lin <- length(parametros_beta_mod_lin)


#Paso 3:
#Lectura de datos del problema. Se espera que los datos del
#problema estén contenidos en un fichero. Dicho fichero ha
#de contener la Kd y lambda del problema:

datos_problema <- scan(file = "problema.txt",
                  what = double(),  n = -2, sep = "")


#Se guarda la Kd del problema en:
```

```
Kd_problema <- datos_problema[1]

#Se guarda el parámetro lambda del problema en:

lambda_problema <- datos_problema[2]



#Paso 4:
#Para testear si los datos del problema se ajustan al modelo
#lineal se va a seguir el siguiente procedimiento:

#a)Primero se va a calcular el Kd predicho por el modelo de acuerdo
#al dato de lambda del problema. Se va a suponer que dicho Kd
calculado
#es una variable aleatoria que sigue una función de
#probabilidad normal. Esta variable aleatoria tiene como
#media el Kd predicho por el modelo. Como desviación típica tiene
#aquella característica del modelo lineal, la cual se suministró en
#el fichero modelo_lineal.txt .

#b)A continuación, se va a realizar un test de t de Student para
#rechazar o no que el Kd del problema sea un valor probable del
#Kd predicho por el modelo.


#a)Cálculo de Kd predicho por el modelo de acuerdo
#al dato de lambda del problema:

Kd_predicho_mod_lin <- parametros_beta_mod_lin[1] +
                        parametros_beta_mod_lin[2] *
                       lambda_problema +
                        parametros_beta_mod_lin[3] *
                        lambda_problema


#b)Test de t de Student:
#En la fórmula del estadístico t de Student se está sustituyendo
#la cuasidesvación típica muestral por la desviación típica
#poblacional (ya que esta última se conoce):

t <- (Kd_problema - Kd_predicho_mod_lin) /
    (desv_tip_mod_lin / sqrt(length(Kd_problema)))

#A continuación se va a calcular la probabilidad acumulada del
#estadistico t calculado. Se desconoce el tamaño muestral. Es
#decir, no se conoce cuántas mediciones del problema se hicieron
#para calcular su valor. Por defecto, se va a especificar un
#tamaño muestral de n = 3 . por ello, el número de grados de
#libertad será df = n - 1 = 2.En futuras versiones del programa
```

```
#el tamaño muestral puede podría ser también un input a introducir
#por el usuario.

t_probabilidad_acumulada <- pt(t, df = 2)


#Si la probabilidad acumulada inferior del 5% o bien superior al
#95% entonces se rechazará la hipótesis con alfa = 0.05. Es decir,
#se rechazará que los datos del problema se ajusten al modelo lineal:


if ((t_probabilidad_acumulada < 0.05) | (t_probabilidad_acumulada >
0.95))
{
  write("SI", file = "sobre_el_rechazo_de_la_hipotesis.txt")
}else
{
  write("NO", file = "sobre_el_rechazo_de_la_hipotesis.txt")
}
```