UNIVERSIDAD DE GRANADA

ANÁLISIS BIOINFORMÁTICO DE LA PROTEÍNA "proteasa m-AAA humana"

CUADERNO PERSONAL DE ACTIVIDADES: INGENIERÍA DE PROTEÍNAS

4º CURSO DEL GRADO EN BIOTECNOLOGÍA

CURSO 2021-2022

Raúl Miñán Campos

Índice

1. Revisión Bibliográfica de la proteína

1.1. Objetivos y metodología

Los objetivos de esta primera actividad son sencillos: Se hará una revisión y documentación bibliográfica sobre la proteasa m-AAA humana para conocer lo mejor posible dicha proteína y su función. Para ello, se revisarán las bases de datos convenientes, como, por ejemplo, Scopus o Pubmed, así como la base de datos PDB para obtener la estructura de la proteína.

1.2. Introducción

Antes de hablar de la proteasa m-AAA es necesario hacer una breve introducción a las proteasas AAA (*ATPases Associated with diverse celular Activities*) mitocondriales. Esta familia de proteínas tiene un papel fundamental en el control de calidad de proteínas de la mitocondria. En concreto, capturan energía de la hidrólisis de ATP para desestabilizar y degradar substratos proteicos en ambos lados de la membrana interna. El mal funcionamiento de estas proteínas conduce a la desestabilización de la proteostasis mitocondrial y está ligado al desarrollo de varias enfermedades en humanos. Dentro de esta familia, destacan dos proteínas: la *intermerbrane space*-AAA y la *matrix*-AAA (m-AAA), siendo la segunda la que nos compete. [4]

1.3. Aspectos funcionales y estructurales

Las proteasas m-AAA son responsables de la regulación de la proteostasis y del control de calidad de las proteínas en la mitocondria, que son esenciales para múltiples funciones, incluyendo la fosforilación oxidativa, la síntesis de proteínas mitocondriales, el ensamblaje de ribosomas mitocondriales, la integridad de los complejos respiratorios I y III y la homeostasis de calcio en la mitocondria. [2].

En cuanto a su estructura, es una proteína hexamérica transmembrana con dos isoformas: Una es un complejo homo-oligomérico formado únicamente por subunidades de la proteína AFG3L2 (*ATPase family gene 3-like 2 polypeptide*), y otra formada por AFG3L2 y paraplegina, un homólogo [2]. La distribución de ambas isoformas es específica de tejido, por ejemplo, una proporción mayor de homo-hexámeros de AFG3L2 en tejidos neuronales. [4]. Para la realización de este cuaderno, me centraré en esta subunidad, la AFG3L2, que se puede ver en la Fig. 1.1a [1]. Esta subunidad, además se divide en 6 cadenas: A, B, C, D, E y F

El aspecto estructural más relevante de AFG3L2 es su división en dos dominios: ATPasa y proteasa. (Fig. 1.1b). [3]

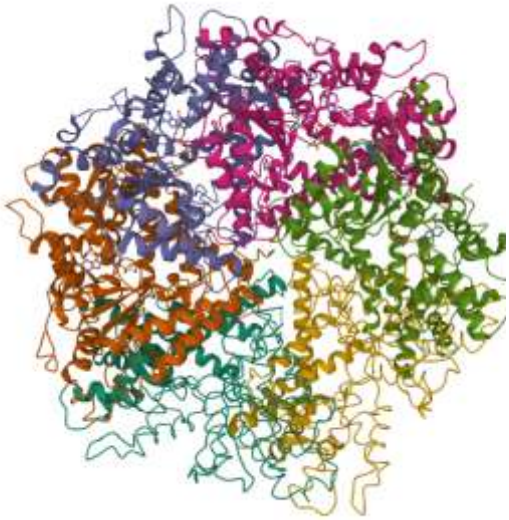


Fig. 1.1a Estructura de AFG3L2 [1]

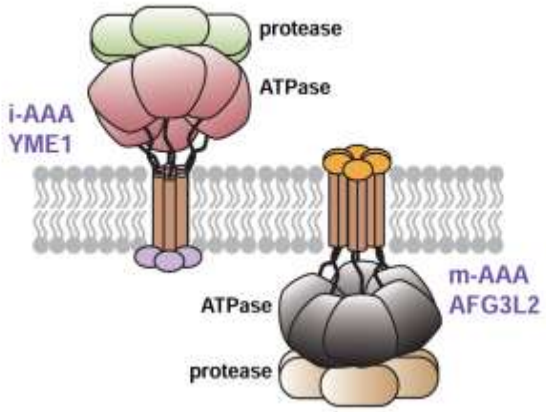


Fig. 1.1b División en dominios ATPasa y proteasa de i-AAA y m-AAA [3]

Por último, es interesante destacar que el espectro clínico de las mutaciones de AFG3L2 es muy amplio. Mutaciones dominantes pueden causar ataxia espinocerebelosa autosómica dominante tipo 28 y atrofia óptica autosómica dominante, mientras que las mutaciones bialélicas pueden conducir a ataxia espástica asociada con atrofia y diversos trastornos del movimiento hipercinético, epilepsia mioclónica progresiva y microcefalia progresiva con convulsiones. Aún hay aspectos de las mutaciones de esta proteína cuyas implicaciones no están esclarecidas por completo, como el papel de las mismas en la ataxia espinocerebral autosómica recesiva. [2]. En este contexto, las herramientas bioinformáticas se postulan como grandes candidatas para avanzar el conocimiento respecto a estas enfermedades.

Referencias

[1] Bank, R. P. D. (n.d.). *RCSB PDB - 6NYY: human m-AAA protease AFG3L2, substrate-bound*. Www.rcsb.org. Retrieved February 20, 2022, from https://www.rcsb.org/structure/6NYY

[2] Chiang, H.-L., Fuh, J.-L., Tsai, Y.-S., Soong, B.-W., Liao, Y.-C., & Lee, Y.-C. (2021). Expanding the phenotype of AFG3L2 mutations: Late-onset autosomal recessive spinocerebellar ataxia. *Journal of the Neurological Sciences*, *428*, 117600. https://doi.org/10.1016/j.jns.2021.117600

[3] Puchades, C., & E. Glynn, S. (2019). Unique Structural Features of the Mitochondrial AAA+ Protease AFG3L2 Reveal the Molecular Basis for Activity in Health and Disease. *Molecular Cell*, *75*(5), 1073-1085.e6. https://doi.org/10.1016/j.molcel.2019.06.016

[4] Steele, T. E., & Glynn, S. E. (2019). Mitochondrial AAA proteases: A stairway to degradation. *Mitochondrion*, *49*, 121–127. https://doi.org/10.1016/j.mito.2019.07.012

2. Programa informático para la discriminación de formatos de ficheros de bases de datos frecuentes

2.1. Objetivos y metodología

Uno de los principales problemas a la hora de hacer análisis bioinformáticos es la obtención de datos. Además de otros obstáculos que no son relevantes para nuestro caso (como puede ser la propia calidad de los datos), un problema común es el diferente formato que tienen los datos alojados en distintas bases de datos. Por ello, el objetivo de esta actividad es desarrollar una aplicación que pueda distinguir los diferentes formatos de las bases de datos más usadas para secuencias de proteínas: FASTA (este no es de ninguna base de datos específica, sino que es un formato muy usado disponible en todas las bases de datos), EMBL, UniProt, GenBank y PDB. Dicha aplicación se desarrollará en Pascal, usando el entorno de Lazarus.

2. 2. Diagrama de flujo

En primer lugar, propondré un diagrama de flujo para distinguir los distintos tipos de archivo, y después haré la implementación en pascal. Para proponer un diagrama de flujo, es necesario una pequeña exploración de los distintos tipos de archivo, para poder establecer unas normas en torno a las cuales se pueden discriminar. Para ello, se compararán las primeras líneas de los ficheros correspondientes a AFG3L2 de los distintos formatos.

Formato PDB:

```
HEADER    TRANSLOCASE                             12-FEB-19
6NYY
TITLE     HUMAN M-AAA PROTEASE AFG3L2, SUBSTRATE-BOUND
COMPND    MOL_ID: 1;
COMPND   2 MOLECULE: AFG3-LIKE PROTEIN 2;
```

Formato UniProt:

```
ID   AFG32_HUMAN             Reviewed;         797 AA.
AC   Q9Y4W6; Q6P1L0;
DT   27-APR-2001, integrated into UniProtKB/Swiss-Prot.
DT   20-FEB-2007, sequence version 2.
DT   29-SEP-2021, entry version 202.
```

Formato EMBL:

```
ID   18    standard; DNA; HTG; 48284 BP.
XX
AC   chromosome:GRCh38:18:12328944:12377227:1
XX
SV   chromosome:GRCh38:18:12328944:12377227:1
```

Formato GenBank:

```
LOCUS       18 48284 bp DNA HTG 16-FEB-2022
DEFINITION  Homo sapiens chromosome 18 GRCh38 partial sequence
12328944..12377227 reannotated
            via EnsEMBL
ACCESSION   chromosome:GRCh38:18:12328944:12377227:1
```

Formato FASTA:

```
>6NYY_1|Chains A, B, C, D, E, F|AFG3-like protein 2|Homo sapiens
(9606)
SNARRGPAGIGRTGRGMGGLFSVGETTAKVLKDEIDVKFKDVAGCEEAKLEIMEFVNFLKNPKQ
YQDLGAKIPKGAILTGPPGTGKTLLAKATAGEANVPFITVSGSEFLEMFVGVGPARVRDLFALA
RKNAPCILFIDQIDAVGRKRGRGNFGGQSEQENTLNQLLVEMDGFNTTTNVVILAGTN
```

Viendo todos los formatos, podemos establecer fácilmente las reglas para discriminarlos. Como podemos ver, si el archivo empieza por "HEADER", es de tipo PDB; si empieza por "ID" y la segunda línea por "AC" es de tipo UniProt, mientras que, si la segunda línea es "XX", es de tipo EMBL; si empieza por "LOCUS" es de tipo GenBank y si empieza por ">", es de tipo FASTA. Por tanto, podemos seguir el siguiente diagrama de flujo:

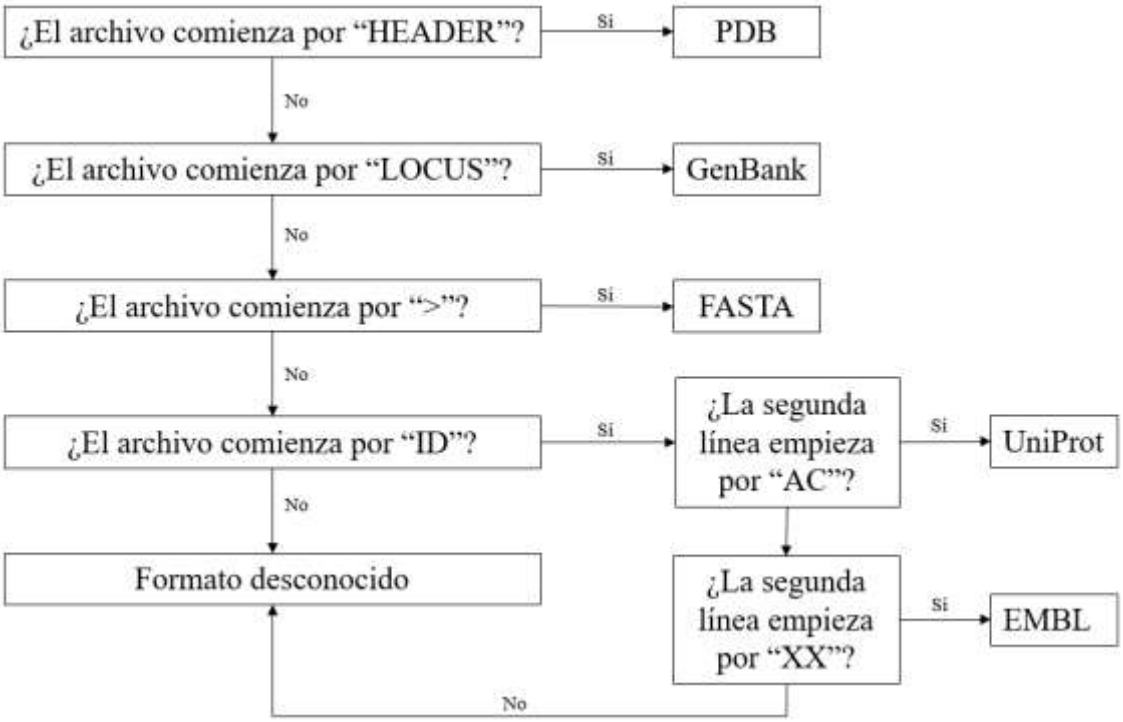


Fig 2.1. Diagrama de flujo para la discriminación de formatos. Elaboración propia.

2.3. Aplicación Informática

El código para la aplicación se puede encontrar en el repositorio. No obstante, a continuación, incluiré *snippets* de la parte más relevante del mismo. Las partes más relevantes de la aplicación son identificar el formato del archivo y obtener la secuencia de este. Para obtener el formato, basta con incluir una serie de reglas "*if"* en la carga del mismo:

```
//cambiar formato según las reglas establecidas
  if (copy(texto[0],0,1)='>') then formato:= 'FASTA';
  if (copy(texto[0],0,6)= 'HEADER') then formato:= 'PDB';
  if (copy(texto[0],0,2)='ID') and (copy(texto[1],0,2)='XX')
   then formato:= 'EMBL';
  if (copy(texto[0],0,2)='ID') and (copy(texto[1],0,2)='AC')
   then formato:= 'UniProt';
  if (copy(texto[0],0,5)='LOCUS')
   then formato:= 'GenBank';
```

Tras obtener el formato, podemos usarlo como parámetro en nuestra función `extraerSecuencia()` para obtener la secuencia del archivo:

```
function extraerSecuencia(texto: TStrings; formato: string): string;
(*
Función para extraer una secuencia de proteína en diferentes tipos
de archivos.
   :Parámetros:
      :param texto: Es el texto generado a partir del archivo
seleccionado
      :param formato: Tipo de formato (PDB, EMBL, FASTA, UniProt o
GenBank)
   :returns:
      :string: Contiene la secuencia o el aviso de que el formato elegido
no está soportado
*)
  var
    j: integer; //índice para los bucles
    sec, linea: String;
    p: TPDB;
  begin
     if formato = 'PDB' then
       begin
         p:= CargarPDB(texto);
         result:= p.secuencia;
       end
     else if formato = 'UniProt' then
        begin
          for j:= 0 to texto.count-1 do
            begin
              linea:=texto[j];
              if copy(linea, 0, 2) = '//' then Break;
              sec:= sec + trim(linea);
              if copy(linea, 0, 2) = 'SQ' then sec:= '';
            end;
         result:= trim(sec);
        end
```

```
    else if formato = 'UniProt' then
        begin
          for j:= 0 to texto.count-1 do
            begin
              linea:=texto[j];
              if copy(linea, 0, 2) = '//' then Break;
              sec:= sec + trim(linea);
              if copy(linea, 0, 2) = 'SQ' then sec:= '';
            end;
         result:= trim(sec);
        end

     else if formato = 'GenBank' then
        begin
          sec:='';
          for j:= 0 to texto.count-1 do
            begin
               linea:= texto[j];
               if copy(linea,0,6) = 'ORIGIN' then Break;
               sec:= sec + trim(linea);
               if copy(linea,22,13) = '/translation=' then sec:=
copy(linea, 35, 70);
            end;
         result:= trim(copy(sec,2,sec.Length-2));
        end

     else if formato = 'EMBL' then
         begin
           sec:= '';
           for j:= 0 to texto.count-1 do
             begin
               linea:= texto[j];
               if (copy(linea,0,2) = 'XX') and (copy(texto[j+1],0,2) =
'SQ') then Break; //para que pare en el último XX que hay

               sec:= sec + trim(copy(linea,22,58));
               if copy(linea, 22, 13) = '/translation=' then sec:=
copy(linea, 35, 39);
             end;
          result:= trim(copy(sec,2,sec.Length-2));
         end

     else if formato = 'FASTA' then
          begin
            sec:='';
            for j:=1 to texto.count-1 do //empezamos en 1 porque queremos
saltar la primera línea
              linea:= texto[j];
              begin
                 sec:= sec + trim(linea)
              end;
         result:= trim(sec);
         end
     else
         result:='Formato no soportado';
end;
```

Aunque para este caso de uso no es muy relevante, creo que es importante destacar que lo ideal hubiese sido hacer la comprobación del formato dentro de la librería *biotoolkit*. De este modo, en lugar de hacer comparaciones como `if formato = 'PDB'`, haríamos `if nueva_función = 'PDB'.` La razón de esto es la mantenibilidad del código. Si *biotoolkit* fuese a ser usada por más usuarios, sería ideal que fuese lo más mantenible posible. Un ejemplo de esto sería la posibilidad de añadir una nueva funcionalidad: si quisiésemos añadir la posibilidad de reconocer otro tipo de archivo adicional, podríamos implementar dicho cambio en *biotoolkit* y todos los usuarios de la librería recibirían la nueva función al actualizarla. Sin embargo, si implementamos la elección de formato dentro de la aplicación (como en nuestro caso), los usuarios tendrían que descargar la aplicación de nuevo. Dicho esto, y ya que este supuesto no se aplica en nuestro caso, es suficiente con la implementación anterior.

2.4. Demostración de la aplicación

Para la demostración del funcionamiento se han descargado cinco archivos de distintos formatos (los mencionados anteriormente) y se ha grabado un GIF. Cabe destacar que no todos corresponden a la misma proteína.

El GIF se puede encontrar en el repositorio, aunque también alojado en Imgur para mayor comodidad.

3. Función para el *parsing* de ficheros PDB: CargarPDB

3.1. Objetivos y metodología

Esta función se ha ido desarrollando a lo largo del cuatrimestre como parte de la librería *biotoolkit*, desarrollada en Pascal en el entorno Lazarus. El objetivo de la misma es permitir al usuario representar adecuadamente los datos estructurales relevantes de una proteína en el entorno de programación.

3.2. Definición de tipos

Pascal es un lenguaje de programación tipado. Esto implica que el tipo de dato de todas las variables debe ser declarado previamente para que su uso quede habilitado [1]. Pascal cuenta con algunos tipos predeterminados, como *integer*, *real*, o *string*, pero también nos permite definir nuestros propios tipos. Por ello, el primero paso para desarrollar nuestra función será definir los tipos que vamos a usar. Los tipos que vamos a usar los definiremos como *records*. Esto son estructuras de datos altamente estructuradas que nos permiten agrupar ítems de datos de forma lógica [3].

```
type
  TPunto = record
        (*
        Representa un punto el espacio tridimensional
        *)
    X,Y,Z: real; //coordenadas
  end;

  TPuntos = array of TPunto;

  TAtomPDB = record
      (*
      Representa un átomo de un archivo PDB, por lo
      que guardará la información de los átomos que se da en
      dichos archivos
      *)
    NumAtom: integer; //Número del átomo
    ID: string; //Identificador del átomo
    residuo: string; //Residuo al que pertenece el átomo
    subunidad: char; //Subunidad a la que pertenece el átomo
    NumRes: integer; //Número del residuo al que pertenece el átomo
    coor: Tpunto; //TPunto que guarda la información de las coordenadas
    R: real; //factor temperatura
  end;
```

```
TResiduoPDB = record
        {Representa un residuo de un archivo PDB}

    NumRes: integer; //
    Subunidad: char; //Subunidad a la que pertenece
    ID1: char; //Código de 1 letra de un aminoácido
    ID3: string; //Código de 3 letras de un aminoácido
    atm1, atmn: integer; //atmN: 1º y último átomos
    N, CA, C, O: integer; //Número de átomos
    phi, psi: real; //ángulos diedros
  end;

  TsubunidadPDB = record
        {Representa una subunidad de un archivo PDB}

    ID: char; //Identificador de la subunidad
    atm1, atmN, res1, resN: integer; //1º y último átomo y residuo
    atomCount, resCount: integer; //
    ResIndex: array of integer; //Índice del residuo
  end;

  TPDB = record
        {Representa un archivo PDB completo}

    header: string; //1ª Línea, encabezado

    //Arrays con la información de átomos, residuos y subunidades
    atm: array of TAtomPDB;
    res: array of TResiduoPDB;
    sub: array of TsubunidadPDB;

    //Número de átomos, residuos y subunidades
    NumFichas, NumResiduos, NumSubunidades: integer;

    //
    subs, secuencia: string;
  end;
```

3.3. Implementación de la función

Una vez definidos los tipos, podemos pasar a implementar la función. El funcionamiento básico se basa en recorrer línea a línea el archivo PDB. Mientras vamos recorriendo el archivo, iremos añadiendo la información correspondiente a los diferentes átomos, residuos y subunidades. Además, calcularemos los ángulos φ y ψ mediante la función `torsion()`, que desarrollaré en la actividad 6.

```
function CargarPDB(texto: TStrings): TPDB; overload;
(*
Función para cargar un fichero PDB en un tipo TPDB,
definido anterioremente
:parámetros:
   :param texto: TStrings que contiene la información del PDB
:returns:
   :TPDB: record de tipo TPDB con la información del archivo PDB
*)
  var
    j, k, F, R, S: integer;
    p: TPDB;
    linea: string;
    resno: integer;
  begin
    //ponemos los contadores a 0
    F:=0;
    R:= 0;
    S:= 0;

    //iniciamos como cadena vacía
    p.secuencia:= '';
    p.subs := '';

    //asignamos longitud sobreestimándola
    setLength(p.atm, texto.count);
    setLength(p.res, texto.count);
    setLength(p.sub, texto.count);

    for j:= 0 to texto.count-1 do
    begin
     linea:= texto[j];
     if copy(linea, 1, 6) = 'ATOM  ' then
     begin
       F:= F +1;

       p.atm[F].NumAtom:= strtoint(trim(copy(linea, 7, 5)));
       p.atm[F].ID:= trim(copy(linea, 13, 4));
       p.atm[F].residuo:= trim(copy(linea, 18, 3));
       p.atm[F].subunidad:=linea[22];
       p.atm[F].NumRes:= strtoint(trim(copy(linea, 23, 4)));
       p.atm[F].coor.X:=strtofloat(trim(copy(linea, 31, 8)));
       p.atm[F].coor.Y:=strtofloat(trim(copy(linea, 39, 8)));
       p.atm[F].coor.Z:=strtofloat(trim(copy(linea, 47, 8)));
       p.atm[F].R :=strtofloat(trim(copy(linea, 61, 6)));

       if p.atm[F].ID = 'N' then
          begin
            R:= R+1;
            p.res[R].NumRes := p.atm[F].NumRes;
            p.res[R].subunidad := p.atm[F].subunidad;
            p.res[R].ID3 := p.atm[F].residuo;
            p.res[R].ID1 := aa3to1(p.res[R].ID3);
            p.secuencia := p.secuencia + p.res[R].ID1;
            p.res[R].atm1 := F;
            p.res[R].N := F;
```

```
        if pos(p.atm[F].subunidad, p.subs)=0 then
           begin
             S:= S+1;
             p.subs:= p.subs + p.atm[F].subunidad;
             p.sub[S].ID:= p.atm[F].subunidad;
             p.sub[S].atm1:= F;
             p.sub[S].res1:= R;

           end;
      end;

    if p.atm[F].ID = 'CA' then p.res[R].CA := F;
    if p.atm[F].ID = 'C' then p.res[R].C := F;
    if p.atm[F].ID = 'O' then p.res[R].O := F;
    p.res[R].atmN:= F;
    p.sub[S].atmN:= F;
    p.sub[S].resN:= R;

  end;
  end;
  p.NumFichas:= F;
  p.NumResiduos:= R;
  p.NumSubunidades:= S;
  setLength(p.atm, F+1);
  setLength(p.res, R+1);
  setLength(p.sub, S+1);

  for j:=1 to p.NumSubunidades do with p.sub[j] do
  begin
   AtomCount:= atmN - atm1 +1;
   ResCount:= resN-res1 +1;
   for k:= p.sub[j].res1 +1 to p.sub[j].resN -1 do
   begin
       p.res[k].phi:=torsion(p.atm[p.res[k-1].C].coor,
                             p.atm[p.res[k].N].coor,
                             p.atm[p.res[k].CA].coor,
                             p.atm[p.res[k].C].coor);

       p.res[k].psi:=torsion(p.atm[p.res[k].N].coor,
                             p.atm[p.res[k].CA].coor,
                             p.atm[p.res[k].C].coor,
                             p.atm[p.res[k+1].N].coor);

     end;

     setlength(p.sub[j].resindex, p.NumResiduos + 1);
     for k:=1 to p.sub[j].ResCount do
     begin
       resno:= p.sub[j].res1 + k - 1;
       p.sub[j].resindex[p.res[resno].numres]:= resno;
     end;
   end;
   result:=p;
end;
```

La función anterior se desarrolló como función sobrecargada. La sobrecarga de funciones permite definir la misma función varias veces, pero con diferentes listas de parámetros [2]. Además, se ha desarrollado otra función `cargarPDB()` que toma directamente como parámetro de entrada un `TPDB` en lugar de un `TStrings`. Esta función permite agilizar el proceso de carga de proteínas.

```
function cargarPDB(var p: TPDB): string;
(*
Función para cargar un fichero PDB en un tipo TPDB, definido
anterioremente
:parámetros:
   :param p: TPDB: Crea un TPDB y lo asigna a la variable p
:returns:
   :string: Contenido del TPDB
*)
  var
    dialogo: TOpenDialog;
    textoPDB: TStrings;
  begin
     dialogo:= TOpenDialog.create(application);
     textoPDB:= TStringlist.create;

     if dialogo.execute then
       begin
        textoPDB.loadfromfile(dialogo.filename);
        p:=cargarPDB(textoPDB);
        result:= dialogo.filename;

       end else result:= '';

     dialogo.free;
     textoPDB.free;
  end;
```

Referencias

[1] de, C. (2003, June 30). *lenguaje de programación*. Wikipedia.org; Wikimedia Foundation, Inc. https://es.wikipedia.org/wiki/Pascal_(lenguaje_de_programaci%C3%B3n)

[2] *Function overloading*. (n.d.). Www.freepascal.org. Retrieved February 20, 2022, from https://www.freepascal.org/docs-html/ref/refse96.html

[3] *Record - Free Pascal wiki*. (n.d.). Wiki.freepascal.org. Retrieved February 20, 2022, from https://wiki.freepascal.org/Record

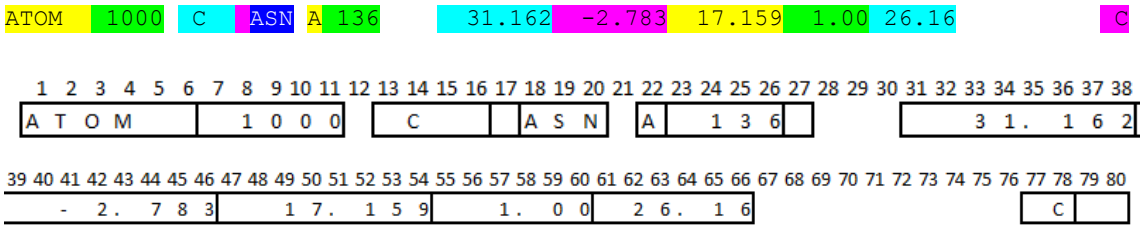
4. Aplicación informática para extraer Cα de un archivo PDB y generar un nuevo archivo PDB a partir de estos: writePDB

4.1. Objetivos y metodología

El objetivo de esta actividad es desarrollar una aplicación informática `writePDB()` que permita reconstruir una línea de texto con el formato correcto de una línea ATOM de un PDB y emplearla para generar un fichero PDB (únicamente las líneas ATOM) a partir de una estructura TPDB previamente importada. De nuevo, esta aplicación se desarrolló en Pascal en el entorno Lazarus.

4.2. Aplicación informática

El primer paso para el desarrollo de la aplicación es estudiar muy a fondo el formato en el que se describen los átomos en los ficheros PDB. Para ello, se han consultado los apuntes proporcionados por el profesor y a partir de ellos he realizado las siguientes infografías en WordPad y Excel, respectivamente:



El motivo de la realización de dichas infografías ha sido que, si bien la implementación de la función es relativamente sencilla, se pueden cometer pequeños errores con bastante facilidad.

Tras haber estudiado el formato, se desarrolló la función `writePDB()` haciendo especial uso de la función `format()` incorporada en Pascal, de la que se buscó la documentación en la wiki oficial de Pascal. [1]

De este modo, la función resultante es la siguiente:

```
function writePDB(atm: TAtomPDB):string;
(*
Función que escribe líneas que contienen la información
sobre un átomo en formato PDB
:parámetros:
   :param atm: átomo cuya info se escribe, de tipo TAtomPDB
:salida:
   :string: contiene la información del átomo
*)
var
   natm, nres, X, Y, Z, R, linea: string;
```

```
begin
  natm:= inttostr(atm.NumAtom);
  nres:= inttostr(atm.NumRes);
  X:= formatfloat('0.000', atm.coor.X);
  Y:= formatfloat('0.000', atm.coor.Y);
  Z:= formatfloat('0.000', atm.coor.Z);
  R:= formatfloat('0.00', atm.R);
  linea:= 'ATOM  ' +
          format('%5s', [natm]) + //5 caracteres alineados a la dcha
          '  ' + //espacio que hay entre las columnas
          format('%-3s', [atm.ID]) + //4 caracteres  alineados a la izq
          ' ' + //omitimos altLoc
          atm.residuo +
          '  ' + //saltamos columna vacía y ChainID
          format('%4s', [nres]) +
          '    ' + //omitimos iCode y 3 columnas vacías
          format('%8s', [X]) +
          format('%8s', [Y]) +
          format('%8s', [Z]) +
          format('%6s', ['1.00']) + //misma ocupancia, 1.00 para todos los
                                    //átomos y dos espacios a la izq
          format('%6s', [R]) + //factor temperatura con 1 espacio a la izq
          '              '; //incluimos 14 espacios para mantener el mismo
formato
  result:= linea;
end;
```

4.3. Demostración del funcionamiento de la aplicación

En el repositorio se puede ver un GIF demostrando el funcionamiento de la aplicación. En dicha animación se puede ver como se extraen los Cα correctamente y cómo se puede guardar el archivo. Además, se incluye una comparación para verificar que la línea correspondiente al primer Cα del fichero es la misma en ambos documentos (salvando las partes que hemos omitido deliberadamente).

Referencias

[1] *Format*. (n.d.). Www.freepascal.org. Retrieved February 20, 2022, from https://www.freepascal.org/docs-html/rtl/sysutils/format.html

5. Aplicación informática para la elaboración de un Diagrama de Ramachandran

5.1. Objetivos y metodología

Siguiendo la misma metodología empleada para el resto de aplicaciones, el objetivo de esta aplicación es elaborar un diagrama de Ramachandran de una proteína introducida en formato PDB.

5.2. Aplicación informática

Introducción

Antes de hablar del desarrollo de la aplicación, es conveniente hacer una breve introducción de qué son los diagramas de Ramachandran. En estos diagramas se pueden visualizar todas las combinaciones posibles de los ángulos de torsión φ y ψ, dando lugar a varias regiones energéticamente permitidas para dichos ángulos. La utilidad de estos diagramas es que permite determinar qué residuos pertenecen a qué tipo de estructura, ya que estas tienen regiones definidas en los diagramas (Fig. 5.1.) [1, 2]

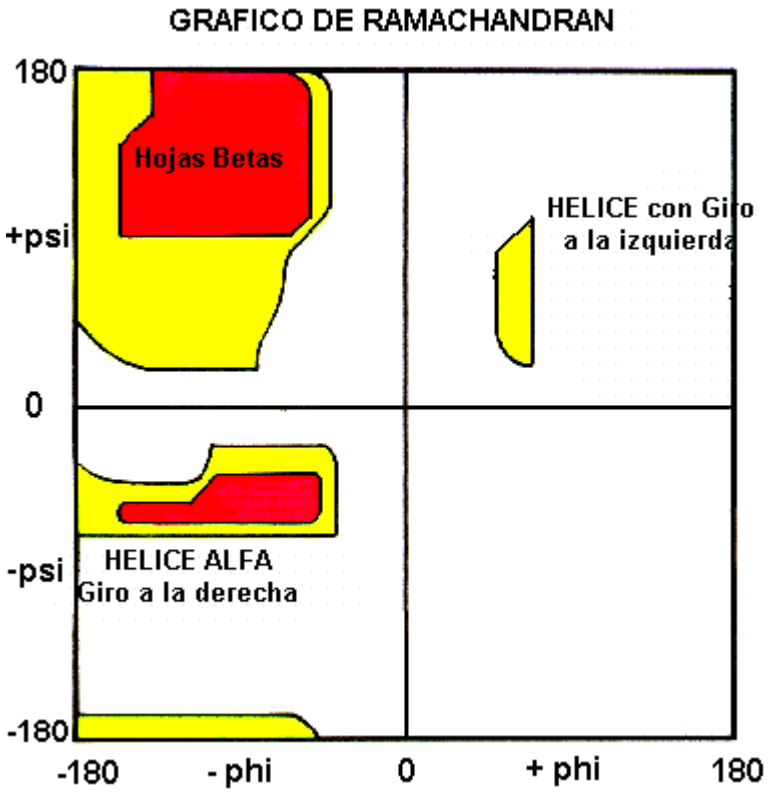


Fig. 5.1. Estructura básica de un diagrama de Ramachandran [2]

Tras esta introducción, se puede inferir que para elaborar el diagrama de Ramachandran, será necesario calcular los ángulos diedros φ y ψ.

Cálculo de ángulos diedros

Para el cálculo de ángulos diedros se han implementado la función `torsion()`, junto con una serie de funciones para operar con vectores: `sumaV()`, `diferenciaV()`, `EscalarV()`, `prodEscalar()`, `prodVectorial()` y `angulo()`. A continuación, incluiré *snippets* de las más importantes, mientras que las demás se pueden ver en el repositorio.

```
function angulo(A, B: TPunto): real;
(*
Función para calcular el ángulo formado por dos vectores

:Parámetros:
  :param A: Primer punto
  :param B: Segundo punto
:Salida:
  :real: Valor del ángulo
*)
  var
    numerador, denominador: real;
  begin
    numerador:= prodEscalar(A, B);
    denominador:= modulo(A)*modulo(B);
    if denominador > 0
       then result := arccos(numerador/denominador)
    else result := maxfloat;
  end;

function angulo(A, B, C: TPunto): real; overload;
(*
Función para calcular el ángulo formado por 3 puntos (dos vectores)

:Parámetros:
  :param A: Primer punto
  :param B: Segundo punto
  :param C: Punto que actúa como vértice
:Salida:
  :real: Valor del ángulo
*)
  var
    BA, BC: TPunto;
  begin
    BA:= diferenciaV(A, B);
    BC:= diferenciaV(C, B);
    result:= angulo(BA, BC);
  end;
```

```
function torsion (A, B, C, D: TPunto): real;
(*
Función para calcular el ángulo de torsión
formado por 4 puntos contiguos

:Parámetros:
  :param A, B, C, D: Puntos
:Salida:
  :real: Valor del ángulo de torsión
*)
  var
      BA, BC, CB, CD, V1, V2, P: TPunto;
      denominador, diedro, diedro_iupac, cosGamma: real;
    begin
      diedro_iupac:= 0;
      BA:= diferenciaV(A,B);
      BC:= diferenciaV(C,B);
      CB:= diferenciaV(B,C);
      CD:= diferenciaV(D,C);
      V1 := prodVectorial(BC, BA);
      V2 := prodVectorial(CD, CB);
      diedro := angulo(V1, V2);
      P:= prodVectorial(V2, V1);
      denominador:= modulo(P)*modulo(CB);
      if denominador > 0 then
        begin
          cosGamma:= prodEscalar(P, CB)/denominador;
          if cosGamma > 0 then cosGamma:= 1 else cosGamma:= -1;
        end else diedro_iupac:= maxfloat;
      if diedro < maxfloat then diedro_iupac:= diedro*cosGamma;
        result:= diedro_iupac;
    end;
```

Una vez calculados los ángulos diedros, basta con dibujarlos en una gráfica, para lo que se ha usado la función `PlotXY()`, de la que no incluiré el código por ser de carácter general y no específica al diagrama de Ramachandran, pero se puede ver en el repositorio.

5.3. Demostración del funcionamiento de la aplicación

Aquí se puede ver un GIF mostrando el funcionamiento de la aplicación usando como ejemplo nuestra proteína en cuestión, la AFG3L2. Además, se ha usado el software MolProbity de la Universidad de Duke [3] para comparar ambos diagramas y el valor de los primeros diez residuos.

RESIDUO	Φ (Pascal)	Ψ (Pascal)	Φ (MolProbity)	Ψ (MolProbity
GLU	-75.15	-7.05	-75.15	-7.05
ALA	-78.38	-21.68	-78.38	-21.68
ALA	-78.79	-32.94	-78.79	-32.94
LEU	-74.84	-19.45	-74.84	-19.45
ILE	-84.75	-35.44	-84.75	-35.44
ALA	-65.09	-32.41	-65.09	-32.41
ALA	-96.74	-9.23	-96.74	-9.23

ARG	-71.45	-38.65	-71.45	-38.65
HIS	-83.94	2.69	-83.94	2.69
LEU	57.16	65.28	57.16	65.28

Tabla. 5.1. Valores obtenidos para los ángulos de torsión, que, como se puede ver, son idénticos. Una herramienta más sofisticada como RASMOL podría dar resultados algo distintos, pero se puede esperar que sean igualmente muy precisos

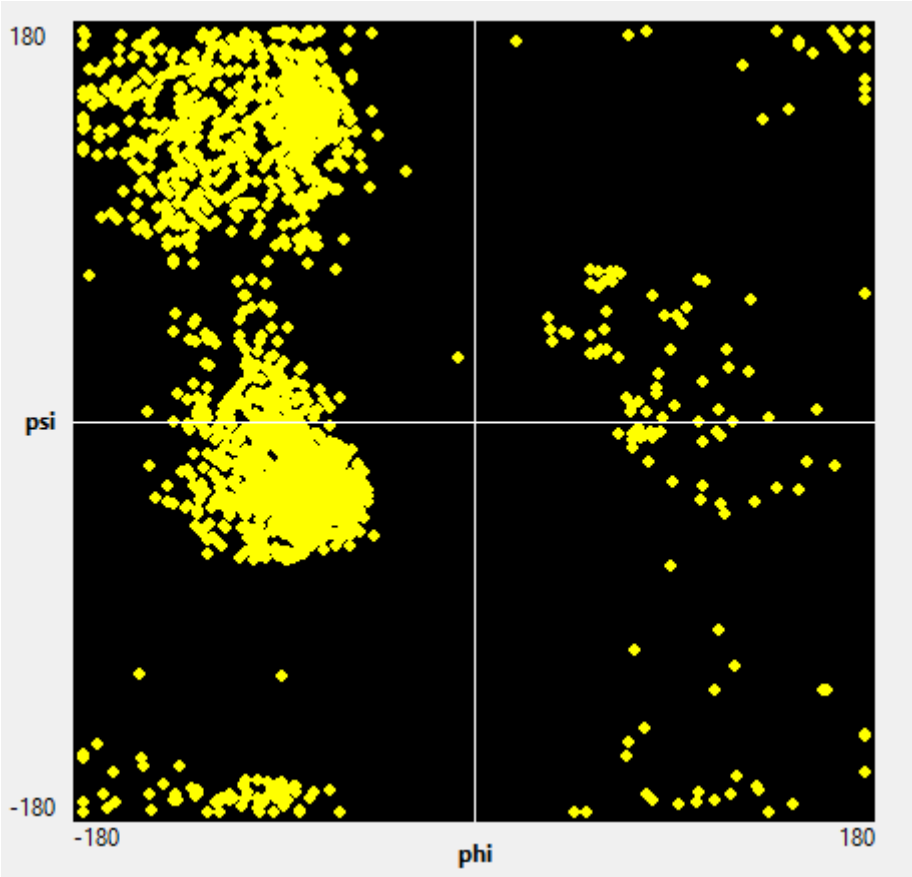


Fig. 5.2. Diagrama de Ramachandran obtenido por nuestra aplicación informática

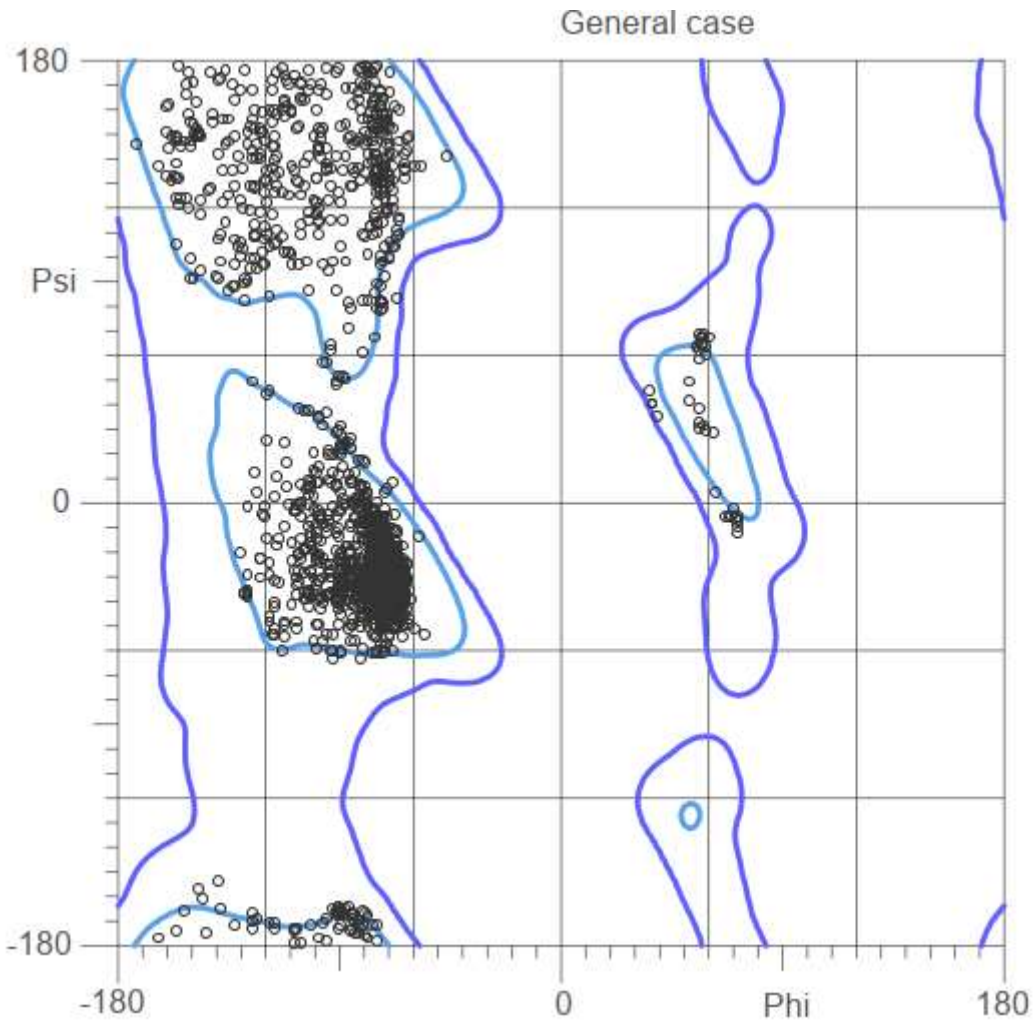


Fig. 5.3. Diagrama de Ramachandran obtenido por MolProbity [3]

Aunque los resultados obtenidos por los primeros residuos son idénticos, podemos ver que hay combinaciones de φ y ψ que nuestro diagrama presenta, pero que no aparecen en el elaborado por MolProbity, especialmente en las zonas de φ = 180, ψ = 180 y φ = 180, ψ = -180. A pesar de esto, la calidad del diagrama es bastante alta.

Referencias

[1] de, C. (2009, May 18). *Gráfico de Ramachandran*. Wikipedia.org; Wikimedia Foundation, Inc. https://es.wikipedia.org/wiki/Gr%C3%A1fico_de_Ramachandran

[2] *Principios de Estructura de Proteínas*. (n.d.). Www2.Udec.cl. Retrieved February 20, 2022, from http://www2.udec.cl/~jmartine/Capitulo3.htm

[3] Williams et al. (2018) MolProbity: More and better reference data for improved all-atom structure validation. Protein Science 27: 293-315.

6. Desarrollo de una aplicación para la elaboración de un estereodiagrama a partir de una proteína.

6.1. Objetivos y metodología

En esta actividad se va a elaborar un estereodiagrama mediante una aplicación desarrollada en Pascal, en el entorno Lazarus. Para ello será necesario desarrollar funciones para la rotación de proteínas en distintos ejes y para la elaboración de gráficas.

6.2. Aplicación informática

Introducción

Los estereodiagramas son representaciones gráficas de objetos tridimensionales en un medio plano. Para elaborar los estereodiagramas, se usan dos fotos del objeto tridimensional contiguas, rotando una de ellas una cantidad pequeña (~5º). Para ver los estereodiagramas se requiere algo de práctica, pero una buena táctica es enfocar la vista *detrás* de la pantalla (esto es, como imaginar que la pantalla del ordenador es una ventana y el objeto a observar está detrás).

Aplicación informática

Como he mencionado, es necesario rotar la proteína para poder elaborar un estereodiagrama, por lo que necesitaremos desarrollar funciones para poder hacerlo, en concreto, las funciones desarrolladas son `traslacion(), GiroOX(), Giro(OZ) y Giro(OY),` que se muestran a continuación tomando como entrada un `TPunto`, aunque en *biotoolkit* se pueden encontrar las versiones sobrecargadas para tomar como entrada `TPuntos` y `TPDB.`

```
function traslacion(dx, dy, dz: real; V: TPunto): TPunto;
(*
Función para la traslación de un punto en el espacio
  :Parámetros:
      :param dx, dy, dz: real que indica cuánto se mueve en cada
dirección
      :param V: TPunto que vamos a trasladar
  :salida:
      :TPunto: TPunto con las nuevas coordenadas
*)
  begin
  traslacion.X:= V.X + dx;
  traslacion.Y:= V.Y + dy;
  traslacion.Z:= V.Z + dz;
  end;
```

```
function GiroOX(rad: real; V: TPunto): TPunto;
(*
Función para la rotación de un punto sobre el eje OX
    :Parámetros:
        :param rad: cantidad que gira en radianes
        :param V: TPunto que se gira
    :salida:
        :TPunto: TPunto con las nuevas coordenadas
*)
  var
     seno, coseno: real;
  begin
     seno:= sin(rad);
     coseno:= cos(rad);

     GiroOX.X:= V.X;
     GiroOX.Y:= V.Y*coseno - V.Z*seno;
     GiroOX.Z:= V.Y*seno + V.Z*coseno;
  end;


function GiroOY(rad: real; V: TPunto): TPunto;
(*
Análogo a GiroOX(). Rotación en el eje Y
*)
  var
     seno, coseno: real;
  begin
     seno:= sin(rad);
     coseno:= cos(rad);

     GiroOY.X:= V.X*coseno + V.Z*seno;
     GiroOY.Y:= V.Y;
     GiroOY.Z:= V.Z*coseno - V.X*seno;
  end;


function GiroOZ(rad: real; V: TPunto): TPunto;
(*
Análogo a GiroOX(). Rotación en el eje Z
*)
 var
    seno, coseno: real;
 begin
    seno:= sin(rad);
    coseno:= cos(rad);

    GiroOZ.X:= V.X*coseno + V.Y*seno;
    GiroOZ.Y:= V.X*seno + V.Y*coseno;
    GiroOZ.Z:= V.Z;
 end;
```

Ahora, podemos usar estas funciones y sus versiones sobrecargadas mencionadas para desarrollar nuestra aplicación.

6.3. Demostración

En este GIF se puede observar el funcionamiento de la aplicación. Además, se incluye un diestereograma elaborado en YASARA del mismo fragmento de la proteína, aunque en posición espacial algo distinta.

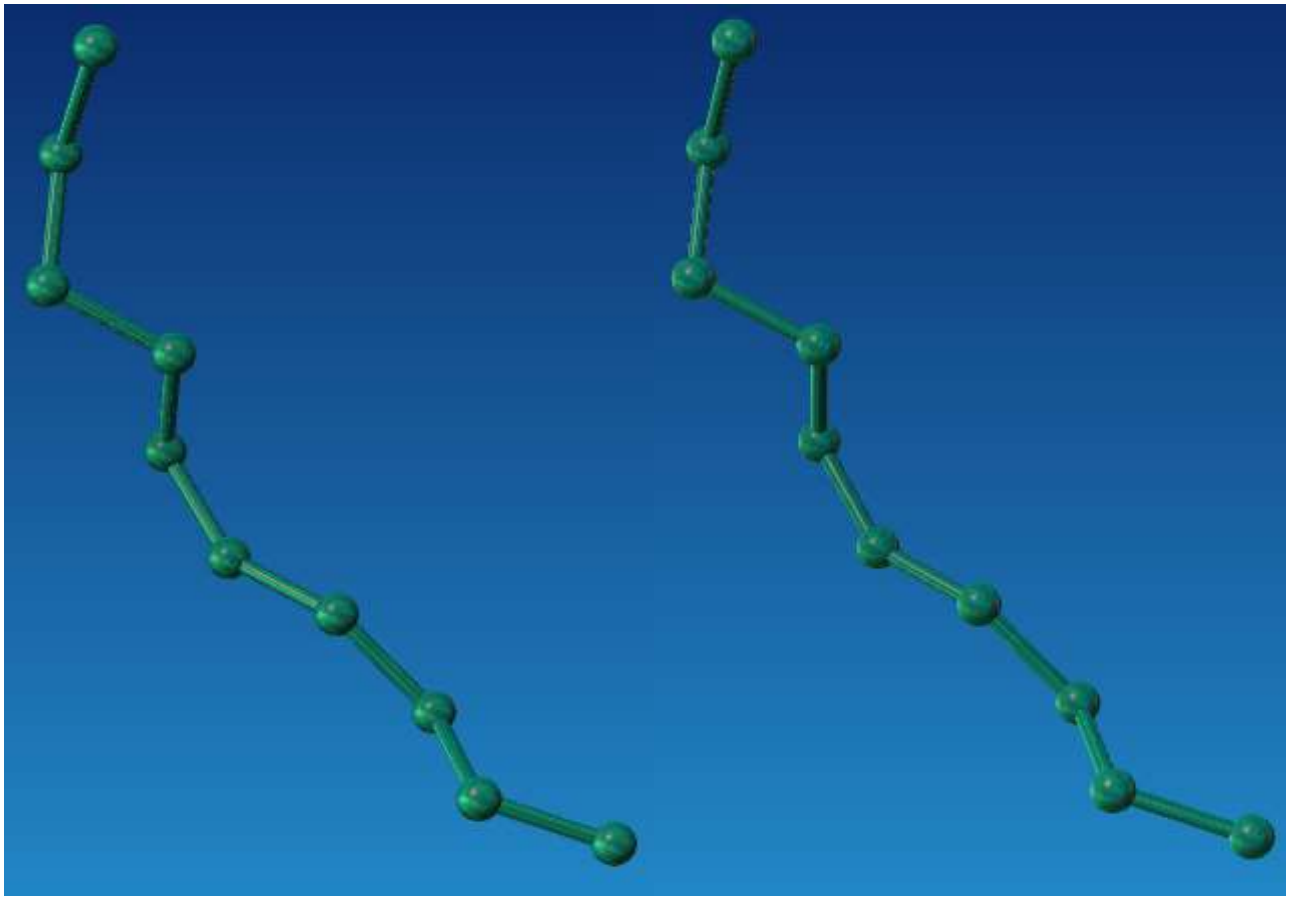
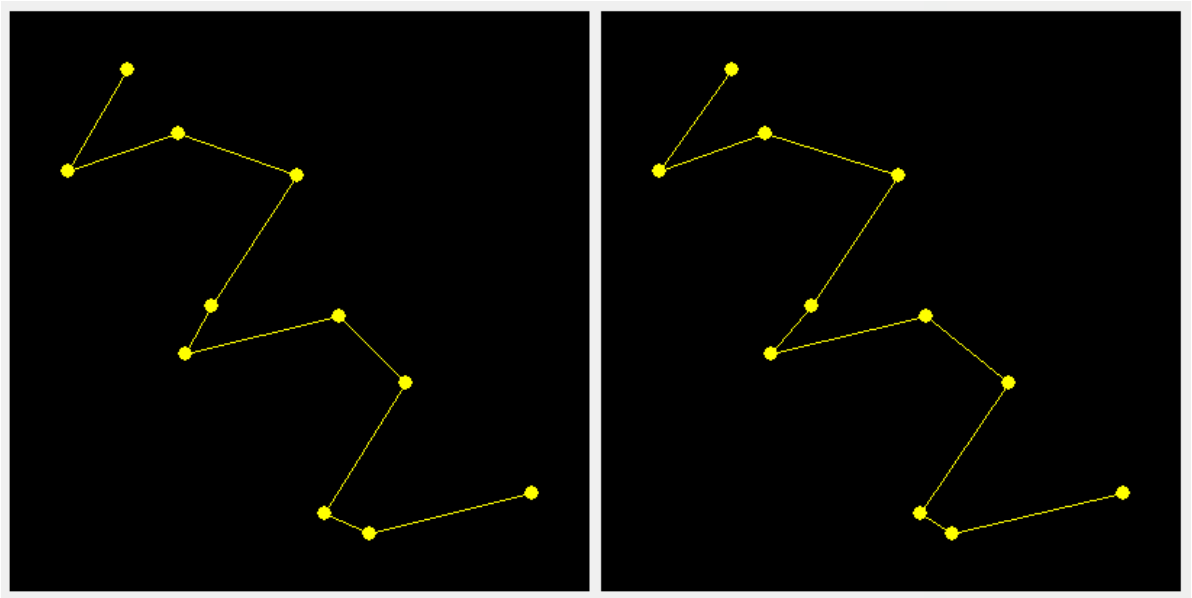


Fig. 6.1. Estereodiagrama elaborada con el software YASARA (arriba) y Pascal (abajo). La imagen de la derecha esta rotada 5º



7. Desarrollo de una aplicación informática para alinear dos Cα en el plano Z

7.1. Objetivos y metodología

El objetivo de esta actividad es desarrollar una aplicación capaz de alinear dos Cα en el mismo plano y dibujar la estructura resultante tras dicha transformación. Esta es una de las aplicaciones que se ha desarrollado a lo largo del curso durante las clases teóricas.

7.2. Aplicación informática

Para el desarrollo de la aplicación, se implementaron las funciones `PlotXY()`, `GiroOX()`, `GiroOY()` y `traslacion()`, desarrolladas anteriormente. Además, se implementó la siguiente función `alinear_eje_Z()` que será la función más importante de la aplicación, pues efectuará el alineamiento.

```
function Alinear_eje_Z(puntos: TPuntos): TPuntos;
{
Función para trasladar un conjunto de puntos tal que dos de ellos
queden alineados en el eje z
:parámetros:
   :param puntos: Puntos que se van a trasladar
:salida:
   :TPuntos: puntos ya trasladados
}
  var
     salida: TPuntos;
     p1, p2: TPunto;
     a, b, c, d1, d2, alfa, phi, senoalfa, senophi: real;
  begin
     setlength(salida, high(puntos)+1);
     p1:= puntos[0];
     salida:= traslacion(-p1.x, -p1.y, -p1.z, puntos);
     p2:= salida[high(salida)];
     a:= p2.X;
     b:= p2.Y;
     c:= p2.Z;
     d1:= sqrt(sqr(b)+ sqr(c));
     d2:= sqrt(sqr(a) + sqr(b)+ sqr(c));
     senoalfa:= a/d2;
     senophi:= b/d1;
     phi:= arcsin(senophi);
     alfa:= arcsin(senoalfa);
     if c<0 then phi:= -phi else alfa:= -alfa;
     salida:= GiroOX(phi, salida);
     salida:= GiroOY(alfa, salida);
     result:= salida;

end;
```

7.3. Demostración

En el siguiente GIF se puede observar el funcionamiento de la aplicación. El alineamiento difiere un poco con el de YASARA (hecho manualmente), pero hay que tener en cuenta que un alineamiento manual es menos riguroso y que en la imagen de YASARA los Cα y los enlaces se ven muy distintos a como están dibujados en nuestra aplicación y hay un gradiente de sombras para indicar profundidad, e incluso alineando únicamente cuatro Cα, se ven algunas diferencias.

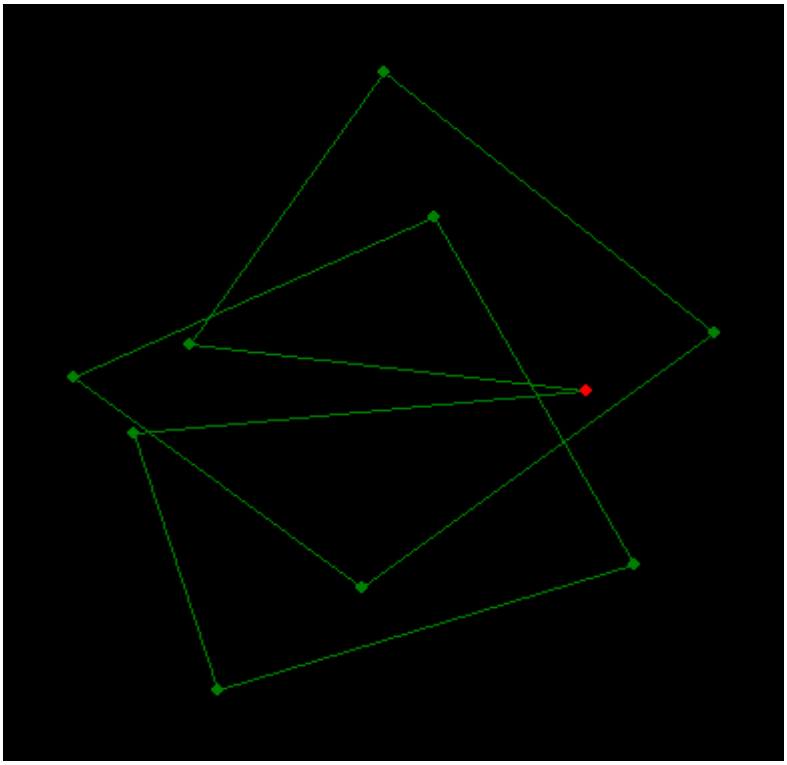


Fig. 7.1. Alineamiento de los Ca 50-60 por nuestra aplicación

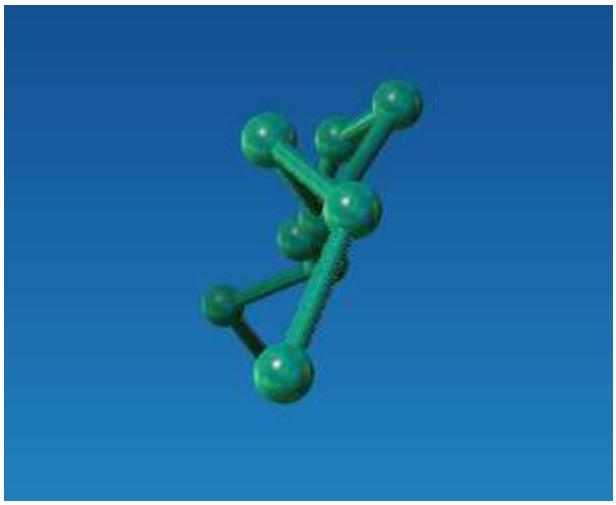


Fig. 7.2. Alineamiento manual en YASARA de los Ca 50-60

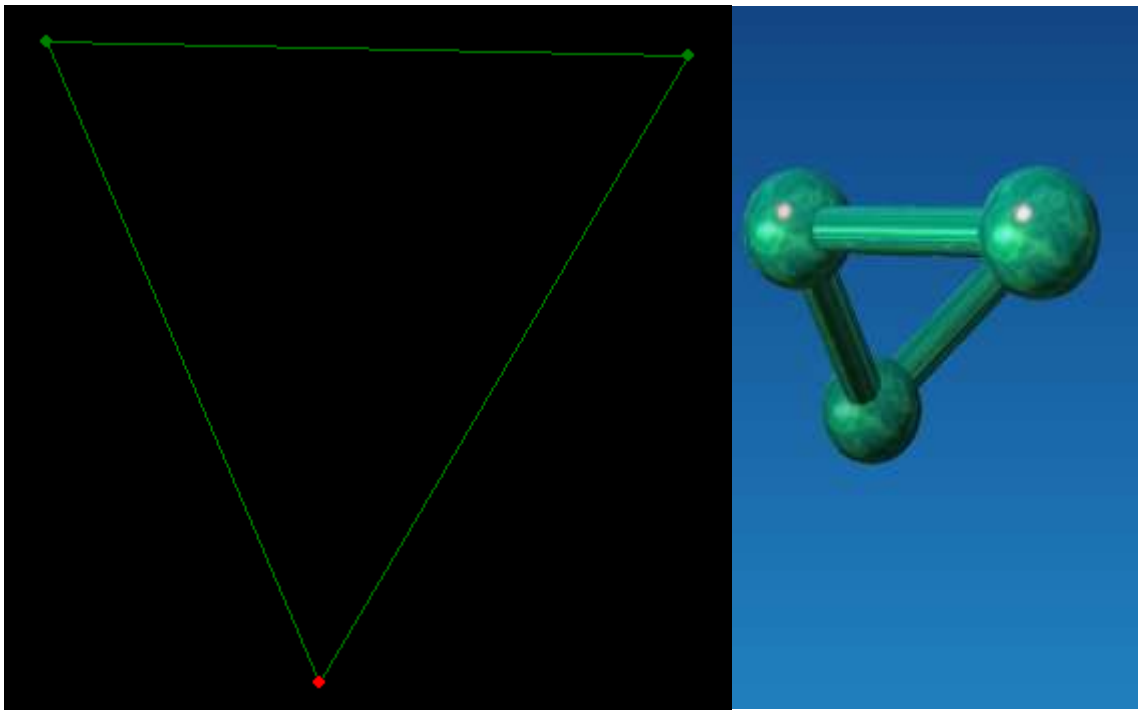


Fig. 7.3. Alineamiento de los Ca 1-4 en Pascal (izq) y YASARA (dch). Al ser una figura muy simple se puede comprobar que funciona correctamente mucho más fácilmente

8. Introducción de una mutación Phe→Tyr y cálculo de las nuevas coordenadas

8.1. Objetivos y metodología

El objetivo de esta actividad es "simular" una mutación de la primera fenilalanina por una tirosina. En cuanto a la metodología, va a cambiar con respecto a las actividades anteriores, ya que no usaremos el entorno Lazarus, sino que haremos los cálculos en una hoja de cálculo de Excel.

8.2. Cálculos

Planteamiento

Para entender los cálculos necesarios, primero vamos a explorar las estructuras de ambos aminoácidos.

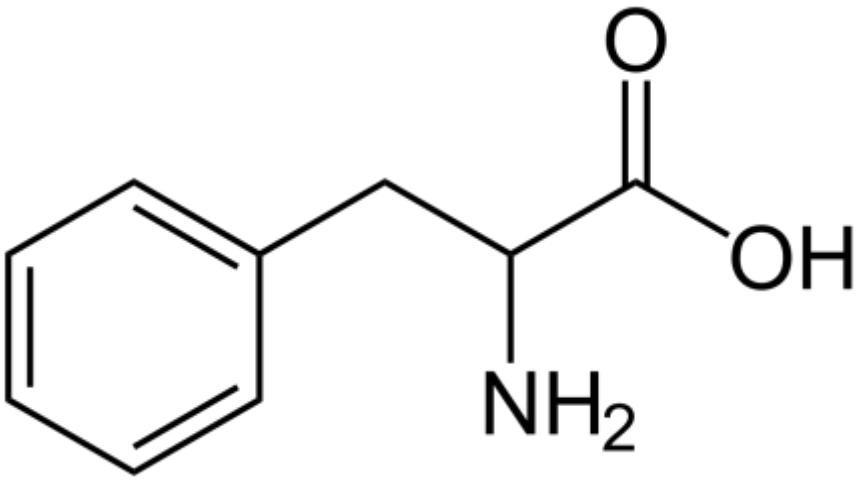


Fig. 8.1. Estructura de la fenilalanina [1]

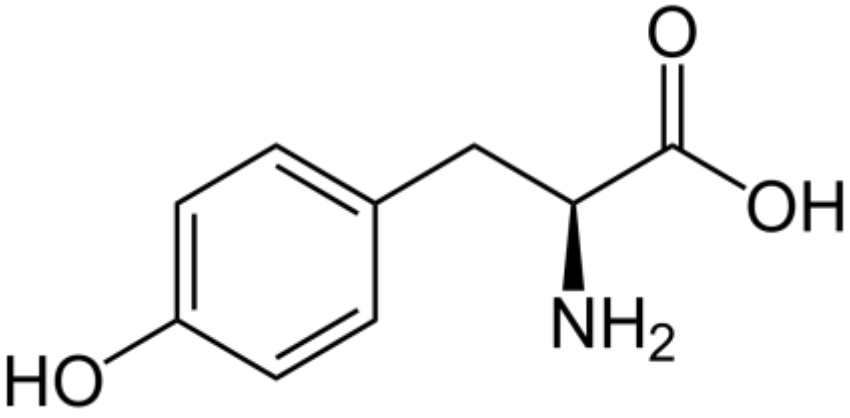
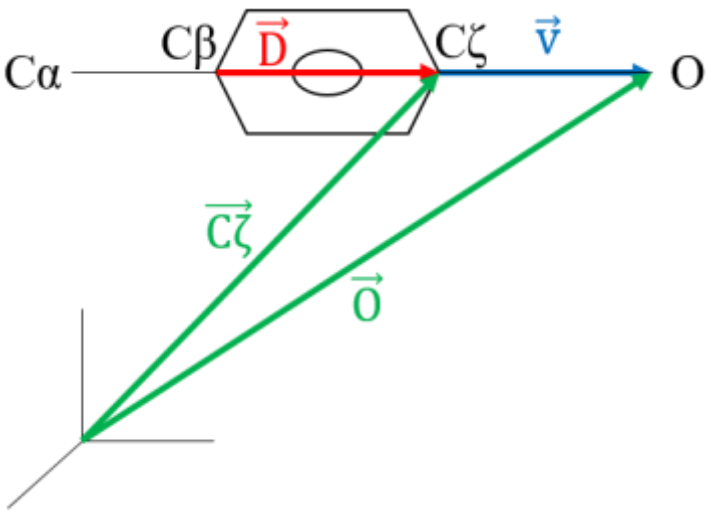


Fig. 8.2. Estructura de la tirosina [2]

Como podemos ver, la única diferencia entre ambos aminoácidos es la presencia de un grupo hidroxilo en el Cζ. Por tanto, la mutación inducida conlleva a la adición de dicho grupo –OH. Para calcular la posición de este grupo, es útil plantear el siguiente esquema:



De aquí se puede obtener:

$$||m|| = \overrightarrow{O} - \overrightarrow{C\zeta}$$

$$\vec{D} = \overrightarrow{C\zeta} - \overrightarrow{C\beta}$$

$$\vec{U} = \frac{D}{||D||}$$

$$\vec{V} = \vec{U} * ||m||$$

Dónde ||m|| es el módulo promedio de la distancia entre Cζ y O de cinco tirosinas escogidas al azar y $\vec{U}$ es un vector unitario. Tras obtener $\vec{V}$, podemos calcular las coordenadas del grupo hidroxilo (de un átomo de oxígeno, en realidad, pero la diferencia que aporta el átomo de H no es muy grande) como:

$$\vec{O} = \vec{V} + \overrightarrow{C\zeta}$$

Cálculos

Cálculo de ||m||					
Tirosina	**Átomo**	**X**	**Y**	**Z**	**m**
1	Cζ	33,186	51,173	51,087	1,37386571
	O	31,957	50,848	50,566	
2	Cζ	41,016	46,410	68,923	1,37097155
	O	40,313	45,383	69,498	
3	Cζ	46,939	44,903	59,884	1,37124797
	O	46,220	45,675	59,008	
4	Cζ	65,714	36,357	67,663	1,3745672
	O	65,485	35,932	68,950	
5	Cζ	64,361	59,884	52,834	1,37391812
	O	64,072	61,037	52,145	
				||m|| =	**1,3729141**

Tabla 8.1. Cálculo de ||m||

Tras explorarse la proteína mediante el visor de PDB, se encontró que no había ninguna fenilalanina en las zonas en las que la proteína interactúa con ligandos (que serían las zonas más interesantes de explorar, pues es altamente probable que una mutación en esas zonas resulte en una proteína no funcional), por lo que se eligió al azar la fenilalanina A 545 para introducir la mutación.

Cálculo de las coordenadas del nuevo átomo			
PHE A 545 --> TYR A 545	**X**	**Y**	**Z**
Cζ	38,353	33,829	81,066
Cβ	38,672	36,518	77,793
D	-0,319	-2,689	3,273
|D|		4,248	
D/|D|	-0,075	-0,633	0,770
V	-0,103	-0,869	1,058
O	**38,250**	**32,960**	**82,124**

Tabla 8.2. Coordenadas del nuevo átomo

Tras hacer los cálculos en Excel, podemos modificar manualmente el archivo PDB y visualizarlo con YASARA para explorar el cambio. En la comparación se puede ver que, aunque los ángulos son diferentes, distancia entre el residuo y la hélice que queda a la derecha (respecto al punto de vista del lector) podría haberse visto modificada. Aunque es puramente especulativo, este cambio estructural se podría transmitir a otras zonas de la proteína y, dependiendo de los cambios que se produzca, podría quedar no funcional.

Comparación

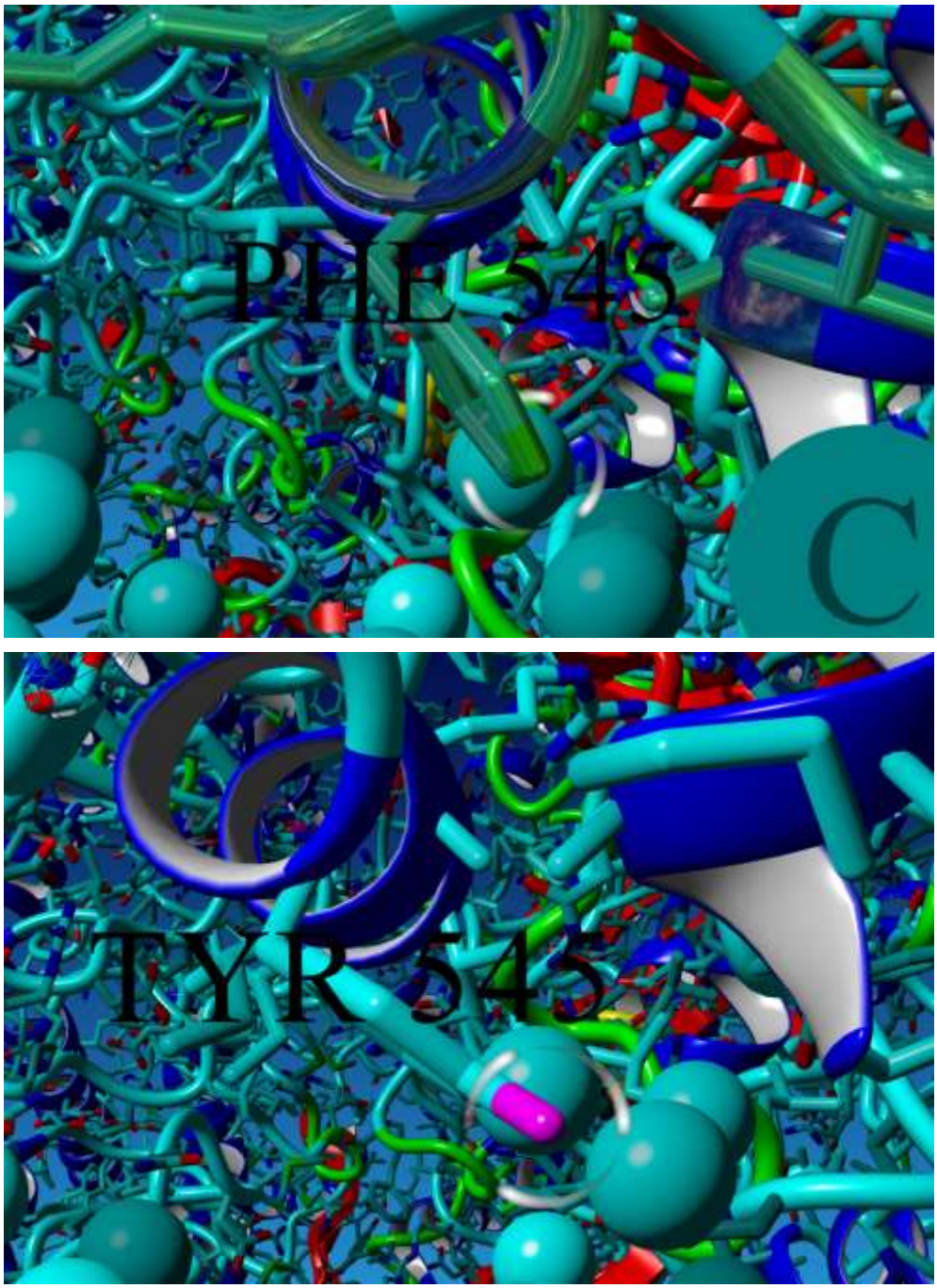


Fig 8.3 Comparación de la estructura con una Phe (arriba) y una Tyr (abajo) mediante el software YASARA

Referencias

[1] *Fenilalanina*. (2022, January 6). Wikipedia. https://es.wikipedia.org/wiki/Fenilalanina

[2] *Tirosina*. (2020, June 27). Wikipedia. https://es.wikipedia.org/wiki/Tirosina