TIEO Sonia - M2BI

Sujet : Assignation des structures secondaire de protéines

Introduction:

Les protéines peuvent être décrites à partir de quatre organisations structurales. La structure primaire est un enchaînement linéaire d'acides aminés formant entre eux une liaison peptidique. Cette chaîne ou squelette peut se déformer (torsion ou repliement) donnant lieu à une structure secondaire. La structure secondaire la plus courante et la plus stabilisatrice est l'hélice alpha (α) : le squelette s'enroule sur lui même et se maintient grâce à des liaisons hydrogène entre les groupes NH et CO à environ tous les quatre acides aminés. **(Figure1)** Parmi les hélices, on trouve également les hélices 3_{10} qui sont stabilisées par des liaisons hydrogène tous les 3 résidus et les hélices pi (π) tous les 5 résidus.
Le feuillet bêta (β) ,une autre structure secondaire, est formé par au moins deux liaisons hydrogène entre atomes côte à côte, formant un plan plissé de chaînes polypeptidique (accordéon). Lorsque les chaînes sont dans le même sens, il s'agit de feuillet parallèle. Dans le cas contraire, c'est un feuillet antiparallèle.**(Figure1)**

Certaines protéines forment une structure tertiaire (enchaînement de structure secondaire) voire une structure quaternaire (enchaînement de structure tertiaires) lui conférant des propriétés et fonctions physiologiques.

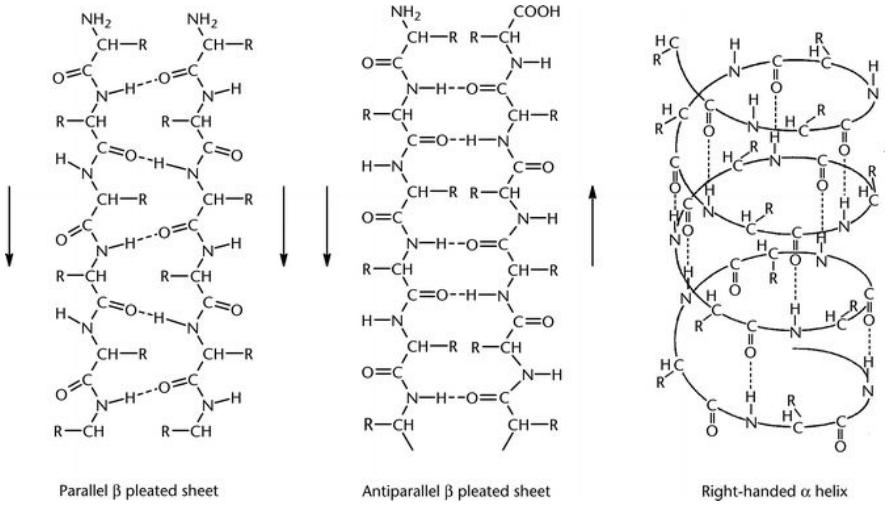

Figure 1 : (gauche à droite) Motifs d'un feuillet β parallèle, d'un feuillet β antiparallèle et d'une hélice α [source : https://www.nature.com/horizon/proteinfolding/background/figs/importance_f3.html]

Nous nous intéresserons ici aux structures secondaires et notre problématique est d'implémenter un algorithme afin d'assigner ces structures secondaires à partir d'un fichier PDB (Protein Data Bank) contenant les informations de la structure 3D d'une protéine (déterminé en général par cristallographie aux rayons X ou spectroscopie RMN).
Plusieurs méthodes ont été mis en place tel que STRIDE , DEFINE ou DSSP. Nous nous intéresserons au programme DSSP (hydrogen bond estimation algorithm) de Wolfgang Kabsch and

Chris Sander. (Kabsch and Sander 1983). Le programme codé en Python implémenté pour ce projet est appelé : *dssp_like.py*.

Matériel et Méthodes:

DSSP se base sur les arrangements de la liaison hydrogène entre le groupe carbonyle (C=O) et le groupe amide (NH) deux deux résidus (i et j) pour définir la structure secondaire. (Pauling and Corey 1951)

Une liaison hydrogène (Hbond) est décrite par l'interaction électrostatique des deux groupes, attribuant des charges partielles q_1 de +0,42*e* et - 0,42*e* pour C et O et q_2 de +0,20*e* et - 0,20*e* pour N et H. Ainsi l'énergie est définie par :

$$E= q_1 q_2 (1/r(0N)+ 1/r(CH) - 1/r(OH) - 1/r(CN))*f \ kcal/moleJ$$

Avec r(AB) la distance entre les atomes A et B (en angstroms) et le facteur f=332.

Soit Hbond(i,j) , une liason H entre i et j , Hbond(i,j) est défini par:

$$Hbond(ij)=: [E < -0.5kcal/mole].$$

L'assignation de DSSP est définie selon 8 états de trois types de structures principales et propose sa propre classification avec le code à 1 lettre suivante :

- G = hélice 3_{10} Le -CO du résidu *i* forme une liaison hydrogène avec le NH du résidu *i+3* (longueur minimale, 3 résidus) ;
- H : hélice α. Le -CO du résidu *i* forme une liaison hydrogène avec le NH du résidu *i+4* (longueur minimale, 4 résidus) ;
- I = hélice π. Le -CO du résidu *i* forme une liaison hydrogène avec le NH du résidu *i+5* (longueur minimale, 5 résidus) ;
- T = coude fermé par une liaison hydrogène (3, 4 ou 5 résidus) ;
- E = brin β étendu dans un feuillet parallèle ou antiparallèle (longueur minimale, 2 résidus) ;
- B = résidu isolé dans un pont β
- S = coude (sans liaison hydrogène).

Les hélices ou feuillets trop courts sont reconsidérés en T ou B, respectivement.

(Voir le détail de calcul de structures : (Kabsch and Sander 1983))

Le programme *dssp_like*, à partir des coordonnées cartésiennes des atomes de la protéine (position ATOM dans un fichier PDB) retourne en sortie un fichier .txt. Ce fichier contient les informations sur les résidus (nom, numéro) suivi de la structure correspondante utilisant la même nomenclature DSSP. Sur ce fichier, est indiqué le début, la fin et la nature du n-turns par des '>' , '<' et chiffres (n =3,4 ou 5) respectifs et le(s) partenaire(s) en interaction avec le résidu impliqué dans la formation pont (E,B). **(Annexe1-Tab1)**

Quelques précisions sur l'implémentation est abordé en annexe. **(Annexe2)**

Résultats et discussion:

Les sorties de dssp_like sont comparés à celle du programme mkdssp (https://www.mankier.com/1/mkdssp#) .

Pour les résultats nous prendrons en exemple la barnase *1bnr.pdb* car c'est une petite protéine simple de 110 résidus, à unique chaîne. Cependant, cette protéine est seulement composée de : H, E (feuillet parallèle et antiparallèle) , T , B et S.
(Pour détecter les hélices R et I, le programme est effectué sur *2qd4.pdb* , plus précisément avec le fichier *2qd4_withH.pdb* où les atomes H ont été rajouté avec *reduce* mais ce ne sera pas présenté dans les résultats - fichiers et exécutables fournis -).

Les sorties de mkdssp (1*bnr_dssp.txt*) et sorties de *dssp_like* (*1bnr_dssplike.txt*) sont dans le dossier et une partie de sortie de *1bnr_dssplike.txt* est en annexe **(Annexe1)**

Globalement *dssp_like* détecte les hélices H et les feuillets E au bon endroit. Le(s) hélices le(s) partenaire(s) BP1, BP2 ont été identifiés correctement : exemple pour les résidus 71 à 75 , les premiers partenaires respectifs sont les résidus de 56 à 52 et les seconds sont les résidus 91 à 87. J'ai néanmoins omis d'attribuer les feuillets E isolés en B. Ainsi, les E isolés obtenu dans le fichier de sortie seront noté E (comme pour le résidu 24 partenaire avec le résidu 50). **(Annexe1-Tab1)** La différence entre les feuillets parallèles et antiparallèles ne sont pas affichés.

Concernant les hélices, un problème est soulevé pour *dssp_like* car il y a toujours 2 résidus en plus impliqués dans H en plus par rapport à mkdssp, un en début d'hélice et un autre en fin d'hélice. A la place, mkdssp y trouve un T ou un S. (ce problème est noté pour les autres types d'hélices également). Prenons la première hélice décrite entre les résidus 7 et 17 selon mkdssp, avec mon programme *dssp_like* les résidus 6 et 18 participent à l'hélice alors qu'ils sont considérés comme des bends. **(Annexe1-Tab1)**

Un autre problème est alors détecté : la discrimination des structures secondaires. En effet, comme DSSP définit une structure secondaire d'un résidu par sa possibilité de former des liasons H, alors ce résidu pourrait intervenir dans plusieurs structures différentes. Pour *mkdssp*, le programme discrimine en prenant la meilleur valeur d'énergie de liaison. Cependant, pour le délai imparti, je n'ai pas pu trancher sur les structures de cette manière, j'ai conscience d'avoir imposé un ordre de priorité de structure très empirique et sans justification. L'ordre de priorité croissante suivante est choisi est: T, S, E, G, H, I , considérant qu'une hélice I est plus rare que les autres. Ainsi cette attribution est à revoir et à améliorer dans le cas de *dssp_like*.

Pour la représentation des T, j'ai réussi à représenter le début et la fin (par les chevrons et les n=3,4 ou 5) comme pour mkdssp sans le 'X' de mkdssp qui indique une coïncidence entre les signes '>' et '<' mais les turns ne sont pas indiqués au endroit et il y a des bends entre les turns. Par exemple pour les résidus 66 à 68 , les attributions respectivement : ' ',T,T,' ' , pour mkdssp. Avec dssp_like les attributions sont : 'T','S','S','T' . **(Annexe1-Tab1)** Une des source de problème possible est l'attribution par priorité de *dssp_like*.

Conclusion:

dssp_like ne produit pas encore exactement les mêmes sorties que *mkdssp*, il reste des changements dans le programme, notamment le choix de la structure secondaire entre plusieurs pour un résidu.

ANNEXE :

Annexe1 - Comparaisons des de sorties de mkdssp et de dssp_like sur 1bnr.pdb :

Annexe-Tab1 : Tableau tronqué du fichier de sortie *1bnr_dssp_like.txt* (avec dssp_like)
Avec RES: le numéro du résidu , AA: l'acide aminé impliqué(code à 3 lettres) , STRUCTURE : composé de la structure selon la nomenclature de DSSP, suivi de chevrons ou de chiffres indiquant la présence, le début et la fin des boucles T.
Les lignes surlignées en jaunes sont les structures des résidus déterminées différemment en comparaison avec la sortie de mkdssp *1bnr_dssp.txt* .

RES	AA	STRUCTURE	BP1	BP2
...	...			
6	THR	H >	0	0
7	PHE	H >	0	0
8	ASP	H >	0	0
9	GLY	H >	0	0
10	VAL	H <	0	0
11	ALA	H <	0	0
12	ASP	H <	0	0
13	TYR	H <	0	0
14	LEU	H <	0	0
15	GLN	H <	0	0
16	THR	H <	0	0
17	TYR	H <	0	0
18	HIS	H <	0	0
19	LYS	S	0	0
...	...			
24	TYR	E <	50	0
...	...			
64	PRO	S	0	0
65	GLY		0	0
66	LYS	T >	0	0
67	SER	S 3	0	0
68	GLY	S 3	0	0
69	ARG	T <	0	0
70	THR		0	0
71	TRP	E	91	56
72	ARG	E	90	55
73	GLU	E	89	54
74	ALA	E	88	52
75	ASP	E	87	0
76	ILE		0	0

Annexe1 - Comparaisons des de sorties de mkdssp et de dssp_like sur 1bnr.pdb :

Annexe2 - Quelques précisions sur le programme

(Toujours sur l'exemple de 1bnr.pdb)

Dans un premier temps, le programme récupère les coordonnées cartésiennes des atomes de la protéine (position ATOM dans un fichier PDB) et crée un dictionnaire afin de stocker tous les informations nécessaires.
Par exemple, pour le dictionnaire nommé dico_res , dico_res[1] permet de récupérer les infos suivantes :
{'N': {'res_num': 1, 'res_name': 'ALA', 'x': 12.138, 'y': 13.048, 'z': 1.934}, 'CA': {'res_num': 1, 'res_name': 'ALA', 'x': 12.241, 'y': 14.092, 'z': 2.993}, 'C': {'res_num': 1, 'res_name': 'ALA', 'x': 10.844, 'y': 14.413, 'z': 3.529}, 'O': {'res_num': 1, 'res_name': 'ALA', 'x': 10.637, 'y': 14.527, 'z': 4.721}}
Et pour accéder à l'atome C du résidu 1 : dico_res[1]['C']
{'res_num': 1, 'res_name': 'ALA', 'x': 10.844, 'y': 14.413, 'z': 3.529}

A partir de cette structure de départ, l'énergie de la liaison H pour chaque résidu est calculée avec tous les autres résidus pour chacun des types de structures.

Lorsqu'un résidu peut appartenir à une structure , il est stocké dans une liste propre à cette structure. Il y a donc des listes pour chaque structure recensant tous les résidus impliqués dans la structure et le(s) partenaire(s) en interaction avec le résidu impliqué.
Par exemple :la fonction list_nturn avec n=4 crée une liste avec tous les numéros des résidus impliqués dans la 4-turns (T)
Sortie pour cet exemple :
list_4turn = ([6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 26, 27, 28, 29, 30, 31, 32, 33, 34, 41, 42, 45, 46], [(6, 10), (7, 11), (8, 12), (9, 13), (10, 14), (11, 15), (12, 16), (13, 17), (14, 18), (26, 30), (27, 31), (28, 32), (29, 33), (30, 34), (41, 45), (42, 46)])
A partir de cette, on peut affiner et former une nouvelle liste pour l'hélice α composé de résidus successifs. (ici en l'occurrence, tous les résidus sont impliqués dans la formation d'hélice α)
h_alpha = [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 26, 27, 28, 29, 30, 31, 32, 33, 34, 41, 42, 45, 46]
C'est le même principe pour les autres structures.

Pour définir la structure "finale" affichée en sortie, le programme n'est pas bien abouti .
Le programme crée un nouveau dictionnaire (ex: dico_res_struct) à remplir. Il stocke les résidus et les clés suivantes : 'res_num' , 'res_nam' , 'type' (pour H, E, T, S …) , 'BP1' , 'BP2' , et trois clés pour les turns '3T' , '4T' , '5T' pour mettre les chevrons…
Une fonction est créé pour chaque structure, elle prend en compte la liste précédente et va remplir le dictionnaire en attribuant aux résidus de la liste la nomenclature de la structure correspondante.
Par exemple, pour le hélices α, comme le résidu 6 est dans la liste h_alpha, la fonction res_helixH va placer 'H' en item pour la clé 'type' :
dico_res_struct[6] = {'res_num': 6, 'res_name': 'THR', 'type': 'H', 'BP1': 0, 'BP2': 0, '3T': '', '4T': '>', '5T': ''}

dico_res_struct est donc modifié à chacun des passages des fonctions qui définissent les structures comme : res_betasheetE , res_helixH , res_helixpi …

Ainsi on voit que c'est empirique :la dernière fonction qui remplit le dictionnaire écrase, les précédents remplissages si le résidu intervient dans plusieurs structures. Ainsi le programme est non seulement pas abouti et pas efficace car il reparcourt les dictionnaires à chaque fois. Ceci est à améliorer.

Points à améliorer :

- Une meilleure factorisation du code, notamment pour la définition des fonctions antiparallel_bridge et parallel_bridge qui se ressemblent , et pour les fonctions de "remplissage" comme res_turnT.
- Une programmation orienté objet avec des classes
- Le système de définition des structures secondaires à partir de la meilleur valeur de Hbond
- La présentation du "main" du programme
- Subdivision du programme