Enseignants référents :
Mr GAUTIER – Mme ROBBE

Réalisé par :
Edmond BERNE – Maud COLLOMB – Léa DUWEZ – Anaïs FLEURY

GB4

Année universitaire 2023-2024

SOMMAIRE :

I- Introduction :

a. Présentation projet :

Ce rapport a pour but de présenter le programme que nous avons réalisé grâce au langage de programmation Python. Il a pour objectif de récupérer toutes les informations importantes extraites d'une fiche PDB (Protein Data Base). Avec ces informations nous avons par la suite dû construire plusieurs graphiques afin d'apporter le maximum d'informations concernant cette séquence protéique à l'utilisateur.
Afin d'atteindre ce but, nous allons tout d'abord vous présenter le contexte biologique dans lequel s'inscrit ce projet. Nous présenterons ensuite les différentes stratégies adoptées concernant les grandes parties de notre programme. Nous expliquerons brièvement les résultats obtenus au cours de notre projet ainsi que leur interprétation biologique. Puis nous élargirons ce rapport sur les potentielles améliorations à rajouter à notre programme afin de l'optimiser au maximum.

b. Contexte biologique :

Notre programme a pour but d'extraire et d'analyser une séquence protéique issue d'un fichier PDB. Un fichier PDB est une fiche résumant les différentes propriétés protéiques d'une séquence. Afin d'analyser notre séquence protéique, nous avons donc extrait toutes les informations importantes concernant cette protéine (ex : 1CRN). Nous avons donc déterminé la séquence en acides aminés de la protéine, si elle est sécrétée ou non (protéine transmembranaire), si elle contient des ponts-disulfures, les coordonnées polaires des carbones alpha et des cystéines …

À partir de ces informations, nous avons donc pu construire plusieurs graphiques. Le tout premier comprend la comparaison des fréquences d'apparition des différents acides aminés dans notre séquence par rapport à une séquence de référence (SwissProt - banque de données biologiques de séquences protéiques où les informations sont saisies et vérifiées manuellement.). Cette comparaison des acides aminés permettra de nous fournir des informations pour comprendre la structure, la fonction et l'évolution de cette protéine.

Nous allons par la suite construire le profil d'hydrophobicité de cette protéine. Celui-ci nous permettra de connaitre le caractère hydrophile ou hydrophobe de la chaîne latérale des acides aminés de notre séquence. Grâce à ce graphique, nous pourrons déterminer la nature hydrophobe des régions de notre protéine à partir de la séquence en acides aminés de la fiche PDB.

Analyser les différentes cystéines nous permettra de faire un calcul de distance et donc de conclure si un pont-disulfure peut potentiellement se former entre deux cystéines. Les ponts-disulfures permettent de stabiliser la structure de la protéine et d'influencer la conformation des protéines. La formation de ponts-disulfures va fortement stabiliser les protéines lorsqu'elles sont excrétées dans un environnement extracellulaire.

Par la suite, les B-factor des protéines peuvent nous servir à identifier certaines classes d'acides aminés, car ils permettent des les colorer sous le logiciel Pymol. Cela peut s'avérer très utile pour observer la localisation d'un certain type d'acide aminé.

La matrice de contact nous permettra de savoir quel acide aminé est proche d'un autre dans la structure tertiaire de la protéine. Cette matrice permettra d'en apprendre plus sur la structure 3D de la protéine, sur les potentielles interactions entre acides aminés ainsi que les propriétés physico-chimiques associées.

II- Stratégies :

A) Interface graphique

Nous avons fait le choix de réaliser une interface graphique grâce aux modules pygame et pygame_GUI. La combinaison de ces deux modules permet de créer une fenêtre et des boutons simples à configurer et accessibles pour l'utilisateur. La stratégie de programmation correspondante à cette partie, basée sur les classes, ne sera pas traitée dans ce rapport. Le code a été annoté et expliqué directement dans le fichier Application.py.

Il est cependant important de noter qu'au début de ce fichier, nous récupérons son adresse pour l'utiliser en tant que chemin d'accès de référence. Cela règle les potentiels problèmes d'adressage entre les différents environnement python ou bien lors de l'exécution du programme directement depuis le système d'exploitation.

B) Fichier PDB

a. Récupération du contenu pdb et des informations sur la protéine

Afin de récupérer le contenu PDB, nous avons créé deux fonctions qui permettent soit de récupérer la fiche PDB disponible en ligne : fonction **importation_online()**, soit de la récupérer localement : fonction **importation_locale()**, c'est-à-dire lorsque celle-ci est stockée dans la base de données du système d'exploitation utilisé. Nous avons également pris en considération la possibilité que le fichier ne soit pas stocké localement, auquel cas un message d'explication est renvoyé à l'utilisateur du programme.

De plus, les informations importantes sur la protéine sont extraites par la fonction **info_imp()** en faisant apparaître : le header, le titre et la taille de la séquence, son identifiant Uniprot, la méthode expérimentale et la résolution.

Afin d'enregistrer la fiche PDB dans le dossier contenant les données, nous avons défini la fonction **enregistrement_pdb()** qui consiste à créer le chemin, avec le module os, vers le dossier contenant le fichier texte à enregistrer avec le code PDB et la fiche PDB.

Nous avons sélectionné les fiches PDB suivantes pour tester notre programme : 1CRN ; 1GC6.

b. Cystéines et ponts disulfures

Ensuite, pour déterminer si les cystéines de la protéine étaient liées ou non, nous avons établi une méthode générale afin de calculer les distances entre un type d'atome spécifié.

coordonnees(): Il s'agit de la première fonction qui vise à extraire les coordonnées (x,y,z) des atomes de soufre des cystéines (SG) ou bien des carbones alpha (CA). Nous avons choisi d'utiliser la même méthode pour ces deux atomes, car la matrice de contact des CA nécessite également ces informations. Dans la fiche PDB, il se peut que certains acides aminés existent sous différentes conformations (ACYS, BCYS…). Dans ce cas-là, le choix d'effectuer une moyenne entre les différents états de l'acide aminé a été réalisé afin de faciliter la création de la matrice de contact (indiqué sur la sortie). De plus, la différence de position entre les états est souvent faible, donc la moyenne est assez proche de tous les états.

En outre, certaines fiches PDB sont particulières. Il peut y avoir une ligne « ANISOU » entre les lignes des coordonnées. Nous ne les avons pas prises en compte. Enfin, pour les dimères (ou oligomères) protéiques, nous avons choisi d'ajouter une lettre (A, B, C etc…) après les noms des cystéines pour les différencier. Cela peut s'avérer très intéressant pour visualiser un pont disulfure entre les 2 parties du dimère.

Le résultat final est stocké dans un dictionnaire pour associer facilement l'atome et ses coordonnées.

calcul_distance() : Cette fonction utilise le dictionnaire précédent pour calculer les distances entre les atomes. Elle stocke le résultat dans un autre dictionnaire avec la paire d'atomes concernée en clé (associée à un flottant en valeur). Afin d'optimiser le processus, la fonction ne calcule pas la distance entre deux atomes identiques (0Å) et ne recalcule pas non plus la distance d'une paire opposée (exemple : atome1-atome2 et atome2-atome1)

recuperation_code_Uniprot/importation_online_uniprot/secreted :
Cet ensemble de fonctions vise à identifier si la protéine étudiée est sécrétée ou non. En effet, les pontdisulfures ne sont présents que sur des protéines extracellulaires. Si le programme réussit à trouver dans la fiche Uniprot que la protéine est sécrétée, un message avertira l'utilisateur que les résultats sont fiables. Dans le cas contraire, le message mettra en garde l'utilisateur que les résultats obtenus sont à prendre avec prudence (protéine intracellulaire ne peut avoir des cystéines pontées)

pontdisulfure() : Cette fonction finale classifie les paires de cystéines selon leurs distances en angström Å. Si elle est inférieure à 3Å, le couple sera stocké dans un dictionnaire précis et dans un autre dans le cas contraire. Cela permet une manipulation plus simple des données lors de l'affichage des résultats pour l'utilisateur. La fonction renvoie également le message en lien avec le lot de fonctions précédentes.

c. Fichier pymol

Afin d'obtenir une analyse structurale approfondie via Pymol, nous utilisons le protocole suivant :

Nous classons chaque acide aminé selon un critère spécifique grâce à la fonction : '**classification(AA, Classification, PDB, chemin_py, repertoire)**'
Cette fonction prend un acide aminé spécifique (AA), un critère de classification (Classification), la fiche PDB (PDB), le chemin vers le dossier où enregistrer le fichier (repertoire), et le répertoire courant (chemin_py). Dans un premier temps, les acides aminés sont répartis en fonction de leur polarité. Plusieurs groupes sont définis, comme les acides aminés polaires non chargés, polaires acides, polaires basiques, apolaires non aromatiques, et apolaires aromatiques. Un dictionnaire (dico_polarité) associe chaque groupe à une valeur. La fonction recherche à quel groupe appartient l'acide aminé spécifié et renvoie la valeur associée à ce groupe dans le dictionnaire. Ensuite, nous classons les acides aminés selon leur poids moléculaire grâce à un dictionnaire (poids_moleculaires_aa). La fonction renvoie le poids moléculaire de l'acide aminé spécifié multiplié par 4. Enfin, notre fonction appelle une fonction externe (tableau_bilan_AA) avec la fiche PDB pour obtenir un tableau de fréquences d'acides aminés. Un dictionnaire (code_acide_amine) associe chaque acide aminé à son code à une lettre. La fonction sélectionne la ligne correspondante à l'acide aminé dans le tableau de fréquences et renvoie la valeur arrondie de la fréquence.

Dans un second temps, nous réalisons la fonction '**fichier_pdb(PDB, Classification, code_pdb, repertoire, chemin_py)**', qui nous sert à créer un nouveau fichier PDB selon la classification des acides aminés obtenue grâce à la fonction précédente. Tout d'abord, elle ouvre un nouveau un fichier PDB, prêt à accueillir des modifications écrites. En analysant méthodiquement chaque ligne de la fiche PDB originale, la fonction identifie celles qui contiennent le terme 'ATOM' et extrait l'acide aminé associé à chaque ligne. Pour chaque

acide aminé ainsi extrait, elle a recourt à la fonction précédente : **‘classification()’** pour procéder à la modification du B-Factor de la ligne atomique en remplaçant la valeur existante par cette nouvelle classification. Les autres lignes, non atomiques, sont simplement copiées dans le nouveau fichier, préservant ainsi l'intégrité de la structure PDB. Enfin, la fonction clôture le fichier PDB nouvellement créé et le renvoie, prêt à être utilisé pour des analyses structurales plus approfondies.

C) Séquence FASTA

a. Extraire une séquence FASTA à partir d’une fiche PDB

Nous réalisons une fonction **(‘header()**’) qui a pour objectif de créer le header d'une séquence au format FASTA en utilisant les informations extraites de la fiche PDB. Elle parcourt la fiche PDB à la recherche des lignes contenant "DBREF" puis construit le header en commencant par “>” et en ajoutant l’identifiant Uniprot et l’organisme chez qui la protéine a été extraite. Nous poursuivons notre code par la fonction ‘**FASTA()**’ qui se charge d'extraire les acides aminés de la séquence à partir de la fiche PDB pour générer le format FASTA. Elle utilise les lignes "SEQRES" de la fiche PDB pour récupérer les résidus. Ces résidus sont ensuite convertis en codes d'acides aminés à une lettre pour créer la séquence séquence protéique. Pour améliorer la lisibilité, notre fonction insère un retour à la ligne tous les 80 acides aminés. Nous combinons le header et la séquence FASTA grâce à la fonction **‘fusion()**’**.** Dans le cas où la séquence n’existe pas, la fonction renvoie un message indiquant l'absence de séquence détectée dans la fiche PDB.

b. Profil hydrophobicité

Nous créons ensuite une fonction permettant de calculer le profil d’hydrophobicité et de potentiellement prédire des régions transmembranaires. Pour cela, nous utilisons l’échelle de Fauchere et Pliska (https://web.expasy.org/protscale/pscale/Hphob.Fauchere.html). Nous réalisons la moyenne de l’hydrophobicité sur une fenêtre glissante de 9 acides aminés. Plus la valeur est positive, plus les acides aminés de la région sont hydrophobes.

Pour ce faire, nous réalisons une fonction (‘**lecture_hydrophobicite()**’) permettant d’ouvrir un URL et de lire un fichier depuis le site internet protscale (actualisation de l’échelle au fil du temps). Notre programme lit ligne par ligne le fichier et les lignes que nous considérons utiles, ne contenant pas certains caractères spécifiques tels que des virgules, points-virgules et accolades sont ajoutés à la liste ‘list_fich’.

Ensuite, une seconde fonction **(‘extraction_data ()’)** nous sert à créer un dictionnaire contenant les valeurs d’hydrophobicité pour chaque acide aminé. Ainsi, les codes à 3 lettres des acides aminés sont les clés du dictionnaire et les valeurs sont les hydrophobicités.

La fonction ‘**hydrophobicité ()’** a pour input la séquence au format FASTA (‘PDB’) et nous renvoie une liste des hydrophobicités moyennes par intervalle de 9 acides aminés, pour cela, nous utilisons le code suivant : ‘fenetre = seq[i:i+9]’

Finalement, une dernière fonction ‘**graphique_hydro()**’ est utilisée pour faire une représentation graphique des résultats. Pour tracer le graphique, nous avons utilisé seaborn.lineplot où en abscisse sont indiquées les positions des acides aminés dans la séquence et en ordonnée les valeurs d’hydrophobicité.

c. Analyse composition

Nous réalisons plusieurs fonctions afin d'analyser et visualiser la composition en acides aminés d'une séquence FASTA.

Premièrement, une fonction **('composition_AA()**') nous sert à extraire la séquence en acides aminés à partir d'une fiche PDB, elle calcule la fréquence de chaque acide aminé, puis retourne un dictionnaire qui nous servira à une utilisation ultérieure dans la création de graphiques ou de tableaux Pandas.

Ensuite, nous réalisons une fonction **('tableau_bilan_AA()**') créant un tableau regroupant les fréquences en acides aminés de la séquence et les fréquences de référence à partir d'un fichier Excel (`valeurs_freqAA.xlsx`). Elle utilise la fonction précédente **`composition_AA()`** pour obtenir les fréquences de la séquence.

Puis, nous créons une fonction effectuant une analyse statistique de proportion avec le test de Fischer **('test_proportion()**'), pour comparer les proportions des fréquences en acides aminés entre la séquence et les valeurs de référence. Enfin, pour visualiser graphiquement nos résultats obtenus grâce aux fonctions précédentes, nous réalisons la fonction 'graphique_aa()' utilisant les bibliothèque : Matplotlib et Seaborn. Sont représentées en bareplot les fréquences des acides aminés. Les étoiles (*) ou "ns" (non significatif) sont ajoutées pour indiquer la signification statistique entre la séquence et les valeurs de référence.

D) Proposer un fichier regroupant les différentes analyses

Nous avons procédé à la création d'un fichier dans lequel nous importons différentes fonctions provenant des modules suivants : Composition_AA, InfoImp, coordonnees_atome, Profil_hydrophobicite.

Afin de faciliter la mise en page du fichier bilan .txt, nous avons au préalable créé une fonction **mise_en_page()** qui permet la création d'une mise en page simple pour le fichier. Cela le rend plus lisible pour l'utilisateur.

Fichier_bilan() : Elle appelle des fonctions externes telles **que tableau_bilan_AA, fusion, info_imp, hydrophobicite, et pontdisulfure**. Puis, elle ouvre un fichier bilan en mode écriture avec encodage utf-8 (garantit le bon format pour les différents utilisateurs). Elle écrit différentes sections du bilan, y compris la séquence FASTA d'intérêt, les informations importantes, l'analyse des proportions des acides aminés, la valeur de l'hydrophobicité de la protéine, et la présence de ponts disulfures.

En cas d'erreur, elle renvoie des messages appropriés.

E) Matrice de contact

Distance_carbone-alpha() : Cette fonction se base sur celles pour les pontdisulfures pour aboutir à un dictionnaire avec les pairs de carbones alpha et leurs distances.

Matrice_contact() : le but ici est de créer la matrice de contact grâce à la bibliothèque pandas. Nous avons utilisé cette dernière et non numpy, car pandas peut convertir ses dataframe directement en .xlsx ou en .csv contrairement à numpy. De plus, le dataframe pandas est proche de celui de R dans ses manipulations et dans son format ce qui le rend plus simple à convertir également en .rds.

Graph_matrice() : Le graphique a été tracé grâce à pyplot de matplotlib, car il existe une fonction qui prend très bien en charge ce type de données (contourf). Seaborn ne contient pas encore de fonction remplissant ce rôle parfaitement (sb.heatmap est limité au niveau visuel).

fichier_matrice() : Pour finir, cette fonction se charge d'enregistrer la matrice au format .xlsx, .csv directement avec pandas et en .rds par l'intermédiaire de pyreadR qui ne prend en charge que les dataframe pandas. Nous avons récupéré le code de la fiche pdb afin de nommer le fichier avec ce dernier. Cela permet à l'utilisateur d'enregistrer autant de matrice qu'il le souhaite tout en les différenciant facilement.

III- Résultats et analyse :

La séquence protéique correspondant au fichier PDB « 1CRN » au format FASTA est la suivante :
« >P01542|CRAM_CRAAB
TTCCPSIVARSNFNVCRLPGTPEAICATYTGCIIIPGATCPGDYAN »
Cette séquence comporte 46 résidus. Nous savons d'après le fichier bilan de la protéine que cette séquence protéique appartient à une plante, plus particulièrement à l'organisme : *Crambe hispanica subsp. Abyssinica*.

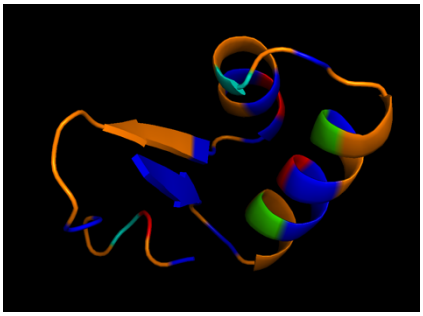
Cette protéine est une protéine hydrophobe, en effet nous pouvons voir que cette séquence est riche en acides aminés hydrophobes ce qui la rend beaucoup moins soluble dans l'eau. Sur 46 résidus, la séquence comprend 19 résidus hydrophobes. Nous pouvons voir grâce au profil d'hydrophobicité que la partie de la protéine 22 résidus à 33 est particulièrement hydrophobe.

Grâce aux coordonnées polaires des cystéines, nous avons pu calculer la distance entre ses différents résidus. Nous savons donc que 3 liaisons entre cystéines existent dans cette protéine. Elles ont entre elles une distance inférieure à 3 angströms :

- La cystéine 3 est liée avec la cystéine 40 avec une distance de 2,04 Angström.
- La cystéine 4 est liée avec la cystéine 32 avec une distance de 2,03 Angström.
- La cystéine 16 est liée avec la cystéine 26 avec une distance de 2,05 Angström.

La présence de pont disulfure dans une protéine est importante car ils vont contribuer de manière significative à la stabilité de la protéine ainsi qu'à sa structure tridimensionnelle. En effet, une liaison covalente (forte) entre deux groupements thiol de deux résidus de cystéine va se former. La présence de ces liaisons va maintenir la protéine dans une certaine conformation et donc influencer l'activité biologique de cette protéine.

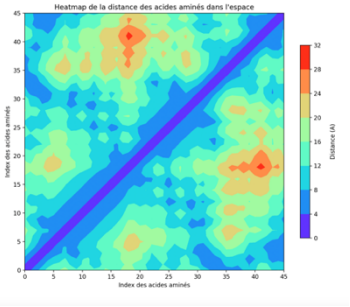
Savoir qu'une protéine est sécrétée ou non va nous apporter de nombreuses informations sur cette protéine. Nous pouvons voir dans la fiche Uniprot de cette protéine qu'elle est bien sécrétée. Elle va donc passer du milieu intracellulaire au milieu extracellulaire. Cette protéine va donc pouvoir assumer sa fonction biologique une fois libérée dans le compartiment extracellulaire, nous pouvons donc avoir affaire ici à par exemple : une hormone, une enzyme digestive, un facteur de croissance, … La sécrétion d'une protéine veut aussi dire qu'elle va exercer sa fonction à distance du site de production de celle-ci.



En regardant PyMOL avec une coloration par polarité des résidus, nous pouvons distinguer de nombreuses structures secondaires. Nous pouvons voir 2 feuillets beta et 4 hélices alpha :

- Feuillets beta : des résidus 2 à 6 et des résidus 36 à 38.
- Hélices alpha : résidus 7 à 17, résidus 23 à 30.

La présence de toutes ces structures secondaires va laisser place à la structure tertiaire et donc la présence de nos ponts disulfures afin de renforcer la conformation de la protéine.



La matrice de contact est une représentation qui va mettre en avant les paires d'acides aminés qui sont en contact les uns avec les autres à l'intérieur de la structure tridimensionnelle de la protéine. On peut voir, dans les zones bleues là où la distance est minimale, que la fin de la séquence protéique se rapproche du début de la séquence. Nous pouvions déjà voir cela grâce à la représentation en 3D de la protéine sur PyMOL. Cette protéine est assez repliée sur elle-même.

IV - Remarques :

Organisation du travail :

Nous avons créé un Github commun (https://github.com/Edmondbrn/Projet-python-GB4) afin de partager les programmes écrits par chacun. Cela permet de mieux synchroniser le travail de chacun et d'avoir une interface propre pour le partage du code. De plus, nous avons réalisé plusieurs séances de travail en commun afin de faire le bilan sur l'avancée du projet régulièrement. À la fin de ces dernières chaque personne recevait une tâche à réaliser pour la prochaine fois.

Difficultés rencontrées :

Étant donné la différence de niveau entre chaque membre du groupe, il a été difficile de concilier / se répartir correctement les tâches, certains membres étant plus rapides au codage que d'autres. De plus, nous disposons chacun d'un système d'exploitation des données différent, à savoir Linux, MacOS, Windows... Il a donc fallu adapter le programme pour chaque système, en incrémentant des instructions dans notre terminal. En termes de codage, nous avons eu des problèmes pour la matrice de contact, notamment les coordonnées des atomes car les fiches PDB ne sont pas les mêmes et certains acides aminés étaient présents en double, sous différents états.

Également, un soin particulier a été donné au programme afin de le rendre compatible pour tous les systèmes d'exploitation (Windows, MacOS, Linux). Nous avons vraiment voulu rendre l'utilisation de ce programme le plus simple possible pour l'utilisateur, comme en atteste les fichiers .bat et .sh qui installent automatiquement les bibliothèques nécessaires. La rédaction du README a aussi demandé un certain travail pour fournir un tutoriel d'installation le plus clair possible.

Gestion des erreurs :

Le programme est construit de sorte à pouvoir résister à de nombreuses erreurs. Tout d'abord, l'interface graphique limité au maximum les erreurs d'input de l'utilisateur grâce aux boutons. Lorsque l'utilisateur entre le code pdb de la fiche, le programme vérifie si la fonction

associée à la récupération retourne la fiche ou le message d'erreur. Dans ce cas, la fenêtre invite l'utilisateur à recharger une nouvelle fiche. Enfin, si la fiche renseignée (en local) est tronquée, le programme affichera un message d'avertissement à l'utilisateur. Cependant, si la fiche PDB est tronquée au niveau des coordonnées des atomes (ATOM), le programme cessera de fonctionner à cause du dépassement d'index qui n'a pas été prévu.

V – Conclusion :

Notre programme répond aux objectifs initiaux qui étaient : analyser une séquence protéique au format PDB. Pour cela, nous avons créé plusieurs fonctions nous permettant de détecter les ponts-disulfures, percevoir un profil d'hydrophobicité, réaliser une matrice de contact, représenter la protéine via Pymol,etc.
Perspectives : Nous aurions pu coder un BLAST à partir de la base de données SwissProt pour trouver des protéines similaires par rapport à notre fiche PDB. Cela nous aurait permis de pousser plus loin l'analyse de la protéine en étudiant des protéines homologues à cette dernière.
Globalement, nous sommes plutôt satisfaits du travail que nous avons pu fournir au sein de ce projet Python. En effet, le programme reste tout de même accessible tout en répondant à toutes les consignes imposées.
Enfin, nous aurions aussi pu transformer notre fichier application.py en fichier .exe avec le module Pyinstaller afin de le rendre indépendant de Python et encore plus simple d'utilisation.