

Rapport de stage au MMSB – Février à Avril 2019

Sommaire

Introduction.....	2
Générale.....	2
Problématique et but de mon stage.....	2
Matériels et méthodes.....	2
Programme Python.....	2
Logigramme du programme.....	3
Premières requêtes et arborescence de travail.....	3
Initialisation.....	4
Domaines protéiques.....	4
Homologie.....	5
Alignements multiples.....	5
WebLogo.....	6
Calcul du taux de conservation.....	7
Intersection entre des fichiers PDB et de la séquence protéique d'Uniprot.....	7
Résultats.....	8
Tableau de synthèse.....	8
Principaux domaines.....	8
WebLogo.....	9
Évolution de la conservation.....	10
Intersection.....	10
PDB / Chiméra vs Uniprot.....	10
Graphique de correspondance des AA avec leur niveau de conservation.....	10
Conservation de domaines.....	11
Conclusion et Discussions.....	12
Remerciements.....	12
Références.....	12
Codes sources.....	13

Introduction

Générale

La résistance bactérienne face aux antibiotiques est expliquée en partie par la capacité de cette dernière à expulser les molécules antibiotiques de son cytoplasme par l'intervention de pompes. Dans le cas de la tétracycline, ces pompes sont contrôlées par des gènes qui sont eux même régulés par des promoteurs protéiques dit TetR. Les protéines TetR sont très présentes chez les procaryotes, et sont les répresseurs des éléments de résistance à la tétracycline. Elles interagissent avec l'ADN comme répresseur de gènes intervenant dans la résistance aux antibiotiques. En effet, la région N-terminale de la protéine forme une structure caractéristique en hélice et se lie à l'ADN, alors que la région C-ter est plus spécifique à la tétracycline. La liaison de la tétracycline à TetR réduit l'affinité du répresseur pour les sites opérateurs du promoteur du gène de résistance à la tétracycline (tetA)¹.

La famille TetR est un domaine d'étude très vaste². Nos collaborateurs Jean Michel Jault et de son équipe* mènent leurs recherches sur 4 protéines dont leur code accession sont les suivants : P04483, P0ACS9, P0A0N4 et Q9AIU0. Ces protéines nous serviront de points de départ pour notre analyse.

Problématique et but de mon stage

La problématique de mon stage s'axe dans les recherches de l'équipe du *mmsb* sur les protéines de la famille TetR.

Mon objectif est de concevoir un programme informatique codé en *Python* qui permettra à terme d'effectuer des opérations automatisées sur des protéines de la famille TetR. Ce programme est destiné principalement aux biologistes qui au cours de leurs expériences peuvent être amenés à effectuer des recherches complémentaires sur des protéines d'intérêt. À terme les utilisateurs pourront visualiser très rapidement les informations clés de la protéine, notamment les domaines protéiques conservés entre protéines de même famille.

L'utilisateur pourra également choisir quelles données issues du programme lui semble utile pour ses travaux. Mon script Python, baptisé *Proteinus*, propose une visualisation des résultats sous forme de graphique. Mais l'utilisateur peut très bien choisir une autre méthode de visualisation des résultats puisque *Proteinus* enregistre les données brutes dans des répertoires.

Matériels et méthodes

Programme Python

Le programme a été conçu au sein d'un notebook *Jupyter*. Il peut donc s'intégrer tel quel avec d'autres notebook déjà existant ou en l'exportant en tant que package/programme python classique. Pour rappel, les notebook *Jupyter* permettent de mélanger des cellules de codes python avec des cellules de textes en langage *Markdown*. Dans tous les cas, mon programme fonctionne en ligne de commande.

Pour utiliser les fonctions proposées par *Proteinus*, l'utilisateur doit au préalable, lors de l'initialisation, renseigner un code d'identification protéique (par exemple : *P04483*). Attention, ce programme a été conçu pour fonctionner avec des codes d'identifications issus de la base UniProt, à défaut de quoi le programme ne pourra

* <http://mmsb.cnrs.fr/equipe/atpasegtphase-bacteriennes-resistance-aux-antibiotiques-et-nouvelles-enzymes/>

pas s'initialiser. La récupération des données principales comme les séquences protéiques, les noms des fichiers PDB... proviendront de la fiche UniProt de la protéine.

Logigramme du programme

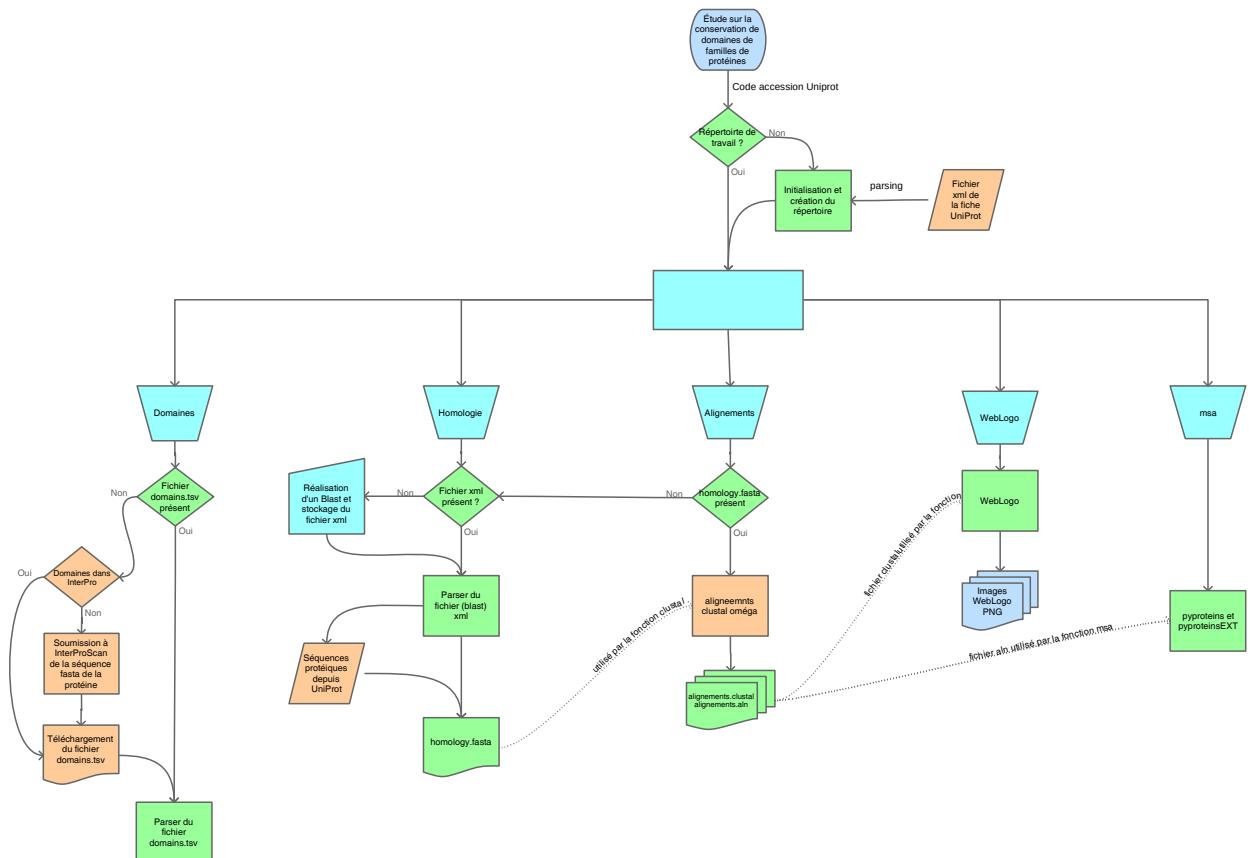


Figure 1: Logigramme du programme Proteinus, présentant les différents pipelines internes.

Comme l'illustre le logigramme (figure 1), la première information à obtenir, si on exclut le code d'identification de la protéine, est sa séquence protéique. C'est à partir de cette dernière que l'on peut effectuer des recherches et des analyses complémentaires comme la recherche de domaines ou encore réaliser divers alignements.

Certaines requêtes du programme seront soumises à UniProt via le protocole REST d'UniProt. Pour rappel le REST (pour *Representational State Transfer*) est un protocole de communication entre deux machines : client et serveur. Ce protocole est basé lui-même sur le protocole HTTP (pour *Hypertext Transfer Protocol*) reposant sur un ensemble de requêtes entre serveur et client³. Ainsi le protocole REST fonctionne en soumettant des url à un serveur qui en retour renvoie les informations demandées sous formes de textes ou en xml.

Premières requêtes et arborescence de travail

La première requête du programme a donc pour but de télécharger la fiche Uniprot de la protéine si cette dernière n'a pas été déjà téléchargée lors d'une utilisation précédente. Cette fiche se présente sous la forme d'un fichier xml qui peut ensuite être parcourue (*parse*) à la recherche d'éléments d'intérêt, notamment la séquence protéique.

Les éléments obtenus seront stockés au sein de différents attributs d'un l'objet python appelé : *protein*.

Le fichier xml nouvellement récupéré (si téléchargé) est alors stocké dans un répertoire créé dans le but de récupérer toutes les données nécessaires (et résultats) pour la bonne exécution du programme.

L'arborescence des répertoires est constituée de la manière suivante. À la racine on trouve un dossier nommé en fonction du code d'identification de la protéine. Au niveau supérieur de ce dossier on trouve un répertoire **Data** qui contiendra tous les fichiers en lien avec la protéine. Cette arborescence est créée et gérée par *Proteinus*.

Attention, il est important de noter que le programme est sensible au renommage manuel des répertoires (et des fichiers). Si le programme ne retrouve pas le chemin d'un fichier ou d'un répertoire (par exemple dans le cas d'un renommage), *Proteinus* ré-importera les données depuis les bases de données en lignes et recréera donc les répertoires de travail (et dans certains cas les fichiers nécessaires).

Initialisation

Le fonctionnement de *Proteinus* repose sur une protéine d'intérêt. Un utilisateur souhaitant utiliser les fonctionnalités du programme devra au préalable se munir de son code d'identification (*accession code*) d'UniProtKB, ceci étant nécessaire à l'initialisation du programme. La commande Python suivante permet d'initialiser le programme :

```
[In 1] P04483 = protein(name = 'P04483')
```

Avec `P04483` le code accession de la protéine.

Le premier processus du programme associée à l'initialisation (*figure 1*), consiste à vérifier l'existence d'un répertoire de travail, dans le but d'accélérer les tâches et de pouvoir stocker les résultats. Lors de cette étape deux conditions sont testées. La première est de vérifier l'existence du répertoire de travail. Pour rappel ce répertoire de travail est constitué d'une petite arborescence décrites ci-dessus.

Si cette condition est vérifiée alors, *Proteinus* utilisera ce répertoire et y exploitera les ressources déjà présentes. Si la condition n'est pas respectée, *Proteinus* créera alors les dossiers nécessaires à son bon fonctionnement.

Une fois les répertoires créés, le programme devra interroger les bases de données UniProt pour récupérer la fiche UniProt au format xml via le protocole REST. La séquence protéique sera ensuite récupérée depuis ce fichier, grâce à une fonction de parsing de fichier xml. Les éléments issus du parsing sont accessibles par ligne de commandes (comme la séquence des acides aminées (AA), le nom du gène.... cf. notebook *Jupyter*)

Le fait de privilégier le téléchargement de la fiche xml permet de pouvoir accéder à d'autres annotations sans être contraint d'interroger les bases de données à chaque requête, cela permet ainsi d'accélérer l'exécution du programme et de limiter les connexions aux ressources en lignes.

Domaines protéiques

La recherche de similitudes peut utiliser les domaines protéiques comme indication de conservation sur une ligne protéique. Pour effectuer cette recherche nous avons à notre disposition *InterProScan*, un outil en ligne permettant d'analyser une séquence protéique à la recherche de signatures particulière. Ces signatures correspondent à des motifs protéiques et sont comparées aux informations contenues dans différentes bases de données comme *ProSite*, *Pfam*, *Prints*, *Panther*...⁴

Pour utiliser *InterProScan*, deux façons s'offrent à nous. La première est de soumettre la séquence protéique à analyse en ligne. Cette analyse peut prendre quelques minutes, en fonction de la taille de la séquence. La seconde est d'interroger *InterPro* en lui soumettant directement le code accession de la protéine. Si notre protéine a déjà fait l'objet de recherches approfondies et donc de plusieurs annotations, les domaines protéiques auront été renseignés et donc directement accessible.

Dans les deux cas, le fichier de sortie téléchargé depuis *InterPro* sera un fichier tsv. Ce dernier possède l'en-tête suivant :

```
Source Database ; Signature Accession ; Signature Description ; Start Position ; Stop Position ; Score ; Status ; Date ; InterPro Entry ID ; InterPro Entry Name ; GO annotations ; Pathway annotations
```

La commande à effectuer pour soumettre la séquence protéique à *InterProScan* ou à *InterPro* est :

```
[In 2] P04483.scanDomains()
```

Une fois le fichier téléchargé, un parsing est effectué par *Proteinus* pour récupérer une partie des informations de ce fichier. Seuls les *signatures accession*, *positions start* et *stop* ainsi que *InterPro Entry Name*, sont conservées et intégrées à l'objet *protein*. De plus, ces informations nous seront utiles pour la suite de notre recherche (*cf. WebLogo page 6*).

L'intégration des informations citées plus haut s'effectue au sein d'un dictionnaire de dictionnaires Python. La première clef du dictionnaire est associée à un dictionnaire ; cette clef est celle donnant la description du domaine (*InterPro Entry Name*). Le second dictionnaire possède deux clefs, la première est *interval* qui retourne une liste de tuple correspondant pour chacun aux positions *start* et *stop* du domaine. La seconde est *name* et renvoie la *signature accession* du domaine. Tout ceci est illustré par le code suivant :

```
[In 3] P04483.domains

[Out ] { 'DNA-binding HTH domain, TetR-type, conserved site': {'interval':
[('21', '52')], 'name': 'PS01081'},
'DNA-binding HTH domain, TetR-type': {'interval': [('10', '55'), ('30', '53'),
('9', '22'), ('3', '63')], 'name': 'PS50977'},
'Tetracycline repressor TetR, C-terminal': {'interval': [('69', '201')], 'name':
'PF02909'},
'Tetracycline transcriptional regulator, TetR': {'interval': [('21', '44'),
('130', '149'), ('95', '118'), ('183', '197'), ('75', '94')], 'name': 'PR00400'},
'Homeobox-like domain superfamily': {'interval': [('5',
'67')], 'name': 'SSF46689'},
'Tetracyclin repressor-like, C-terminal domain superfamily': {'interval':
[('69', '204')], 'name': 'SSF48498'},
'G3DSA:1.10.10.60': {'interval': [('1', '66')], 'name': 'G3DSA:1.10.10.60'},
'G3DSA:1.10.357.10': {'interval': [('67', '203')], 'name': 'G3DSA:1.10.357.10'}}
```

Il est possible par la suite de s'intéresser à un domaine d'intérêt en le spécifiant par exemple :

```
[In 4] P04483.domains['DNA-binding HTH domain, TetR-type']

[Out ] {'interval': [('10', '55'), ('30', '53'), ('9', '22'), ('3', '63')], 'name':
'PS50977'}
```

Homologie

La recherche de séquences homologues s'effectue de manière manuelle avec l'interface Blast en ligne. La recherche de séquence homologue implique l'implémentation de fonctions permettant de réaliser des alignements locaux et donc d'avoir à disposition une base de données de séquences à aligner, maintenue à jour.

Par conséquent, il est nécessaire d'enregistrer un fichier de résultat de Blast au format xml, dans le répertoire *Data*. Pour les 4 protéines d'études, la recherche d'homologie c'est fait avec BLASTp version 2.9.0⁵, sur la base SwissProt et comme paramètres principaux : un word size à 6 et un seuil de score max à 1.

À l'appel de la fonction *homology*, *Proteinus* parcourt le fichier xml pour récupérer les codes accessions des séquences qui se sont alignées avec notre séquence d'études. En parallèle, pour chaque code accession, la séquence protéique est téléchargée et écrite dans un fichier fasta (*homology.fasta*). Ce fichier fasta sera utilisé par les autres fonctions du programme, comme celle de l'alignement multiple (*page 6*).

La méthode de téléchargement des séquences protéiques, s'effectue en exploitant le protocole REST d'UniProt (détailé en introduction page 2).

La commande correspondante est :

```
[In 5] P04483.homology()
```

Il est possible de spécifier un nom de sortie pour le fichier contenant les séquences homologues. Par défaut, ce nom est **homology.fasta**.

Alignements multiples

Afin de transposer les positions des domaines trouvées par *InterPro* sur les séquences homologues de notre protéine, nous allons réaliser un alignement multiple. Ce dernier sera effectué avec *Clustal Oméga*⁶. Ce programme *Clustal* est un ensemble de programme informatique permettant de réaliser des alignements multiples. Le programme *Clustal* possède de nombreuses variations⁷. Parmi ses variations nous avons choisi la version oméga (Ω) qui est la version la plus couramment utilisée dans le domaine de la bio-informatique.

ClustalQ est utilisable en ligne ou en récupérant les sources du programme sur le site de l'EBI*. *ClustalQ* est également disponible au sein d'un package Python (cf. *notebook*).

Nous utiliserons la deuxième méthode ici. Le script d'alignement multiple est un simple script Python qui va soumettre notre fichier fasta (contenant les séquences homologues de notre protéine) passé en entrée. Le script `clustalo.py` effectue des requêtes aux serveurs de l'EBI (via le protocole HTTP) qui se chargeront d'effectuer l'alignement multiple et nous retourner les résultats.

En sortie nous récupérerons un fichier d'alignement clustal qui sera sauvegardé dans le répertoire *Data* de la protéine. De plus, ce fichier clustal sera copié dans un autre fichier dont l'extension est `.aln` pour pouvoir être pris en charge par une librairie de gestion d'alignement multiple (cf. *Calcul du taux de conservation page 7*). Cette copie est nécessaire pour la suite de l'analyse et est automatisé par *Proteins*.

La commande type est la suivante :

```
[In 6] P04483.clustal(out = '', fasta = '')
```

Avec `out` désignant un préfixe pour le nom de sortie de l'alignement. Par défaut, cette valeur ne vaut rien. Et `fasta` étant le nom du fichier contenant les séquences homologues, par défaut le fichier est `homology.fasta`. Sauf si un nom de sortie spécifique a été renseigné en utilisant la commande [In 5].

WebLogo

La réalisation d'un weblogo va nous permettre uniquement de visualiser les conservations des domaines au sein de la protéine.

Comme pour *ClustalQ*, le script source peut être récupéré depuis les internets mais pour des raisons pratiques, nous utiliserons le package *WebLogo* (version 3.7.1) disponible sur le site de [pypi.org*](https://pypi.org/project/weblogo/) et installable en local.

Il existe deux façons de réaliser les *WebLogo* : soit sur l'intégralité de la séquence protéique, soit en se concentrant uniquement sur les domaines du moment que l'on connaît leurs coordonnées.

WebLogo utilise une formule d'entropie pour établir la hauteur des lettres à afficher. Cette formule⁸ sera réutilisée par la suite pour calculer le taux de conservation.

$$R_i = \log_2(20) - (H_i + e_n)$$
$$H_i = - \sum_{b=a}^t f_{b,i} \times \log_2(f_{b,i}) \quad H_i \text{ est souvent appelé l'entropie de Shannon ; } f_{b,i} \text{ est la fréquence de l'AA } b \text{ à la position } i.$$
$$e_n = \frac{1}{\ln 2} \times \frac{19}{2n} \quad e_n \text{ est un correcteur, } n \text{ correspond à la longueur de la séquence protéique.}$$
$$\text{height} = R_i \times f_{b,i} \quad \text{height, aussi appelé bit, est une unité permettant de mesurer la hauteur des lettres à afficher dans le weblogo.}$$

La commande type associée à cette fonction est :

```
[In 7] P04483.WebLogo(paramCustom, start = START, stop = 0, label = '',
nameOut = 'WebLogo.png')
```

Avec `paramCustom` qui indique si l'utilisateur veut utiliser des paramètres personnalisés pour réaliser un *WebLogo*, par défaut ce paramètre vaut `False`, ce qui signifie que les paramètres par défaut (définie dans le

* <https://raw.githubusercontent.com/ebi-wp/webservice-clients/master/python/clustalo.py>

* <https://pypi.org/project/weblogo/>

programme python) seront employés. Le paramètre `start` indique la position du premier AA de la séquence à utiliser. Par défaut ce paramètre vaut 1. Le paramètre `stop` indique le dernier AA de la séquence à utiliser. Par défaut ce paramètre vaut 0, ce qui signifie qu'il faille utiliser le dernier AA de la séquence protéique. Le `label` est un paramètre optionnel qui permet d'ajouter un sous-titre au graphique de sortie. Enfin `nameOut` indique le nom de sortie de l'image générée.

Ces paramètres s'adaptent automatiquement si l'utilisateur souhaite réaliser un weblogo sur les domaines de la protéique. C'est-à-dire que *Protéinus* utilisera les coordonnées de/des intervalle(s) des domaines. Dans ce cas-là, le *WebLogo* sera réalisé en commençant par le premier acide aminé du domaine, se terminera par le dernier. De plus, le label et le nom de sortie de l'image seront le nom de ce domaine.

Attention, étant donné que la fonction *weblogo* peut-être amenée à utiliser les coordonnées des domaines protéiques, il est fort recommander d'exécuter la commande [In 2] avant de procéder à la réalisation d'un weblogo.

Calcul du taux de conservation

WebLogo nous permet, ici, uniquement de visualiser les conservations. Dans une démarche plus scientifique, il est nécessaire de pouvoir calculer l'entropie (de Shannon) à partir de la formule utilisée par *WebLogo*.

Il existe une librairie développée au sein de l'équipe qui possède un module de gestion des alignements multiples. Cette librairie est composée de deux modules (*pyproteins* et *pyproteinsExt*)^{9,10} disponibles sur la plateforme *GitHub*, elle a été créée par Guillaume Launay.

Ces modules possèdent de nombreux script dont l'un (*msa.py*) parvient à parser le fichier d'alignement clustal (celui dont l'extension est aln, mais qui pour rappel est une parfaite copie du fichier clustal). À l'issue de cette opération il en ressort une matrice contenant la fréquence des acides aminés.

Nous allons récupérer cette matrice et y effectuer les calculs d'entropie. Il existe deux façons de procéder. La première consiste à supprimer toutes les colonnes de la matrice contenant un ou plusieurs gap. La seconde consiste à considérer le gap comme un acide aminé, de ce fait on pourra obtenir la fréquence des gap.

La sortie générée par la première méthode retournera une liste qui contiendra la valeur de l'entropie pour chaque acide aminé.

La sortie de la seconde méthode ne retourne rien. En effet les résultats sont directement stockés dans un attribut de l'objet *protein*.

Lorsque l'on demande l'affichage des résultats, la fréquence de chaque acide aminé, de la colonne, de l'alignement est renvoyée, ainsi que de la valeur de l'entropie associée.

Les commandes associées à la première méthode sont :

```
[In 8] oMsa = msaLib.Msa(fileName = P04483.clustalFile)
[In 9] oMsa.entropyS
[Out ] [0.37677016125643675,
         0.37677016125643675,
         0.37677016125643675,
         0.37677016125643675,
         ...
         ...]
```

Les commandes associées à la seconde méthode sont :

```
[In 10] P04483.msa()
[Out ] pos 1 is M 1 / 8 = 0.125
        pos 1 is - 7 / 8 = 0.875
        entropy = 0.37677016125643675 with height = 3.394078880200541
        ...
        ...]
```

Il est possible d'effectuer un graphique permettant de visualiser l'évolution de l'entropie en fonction de la longueur de la séquence. De plus il est possible de borner ce graphique en fonction des coordonnées des domaines conservés. Pour réaliser un graphique il faut utiliser le module *matplotlib*.

Intersection entre des fichiers PDB et de la séquence protéique d'Uniprot

Afin de permettre d'augmenter la qualité de visualisation des résultats sur la conservation des domaines, nous allons comparer les positions des acides aminés données par le ou les fichiers PDB et ceux de la séquence référencée dans Uniprot. Ou encore, ceux issus d'un fichier texte produit par le logiciel *Chiméra*.

Les fichiers PDB (*Protein Data Bank*) est un format standard pour la représentation des macromolécules issue d'expérience de cristallographie (rayon-X ou RMN)⁵.

Chiméra est un programme (utilisable avec interface graphique ou en ligne de commande) permettant de visualiser et analyser les structures moléculaires¹¹. De nombreux outils sont disponibles au sein du logiciel pour réaliser des analyses bio-informatiques. *Chiméra* à la possibilité de prendre en entré les fichiers PDB.

Parmi les nombreux outils du logiciel, il est possible de sélectionner des acides aminés en fonction de plusieurs critères, par exemple ceux en interactions avec la tétracycline et de récupérer cette liste d'AA dans un fichier texte.

Voici les commandes de *Chiméra* pour enregistrer un fichier texte contenant les AA situés à moins de 5 Å de la tétracycline :

```
select ~#0:tdc & #0:tdc za<5  
writesel my_selection.txt namingStyle simple
```

Proteinus dispose d'une fonction permettant d'afficher la séquence protéique en indiquant par un jeu de couleurs les acides aminés communs entre ceux du fichier PDB, ceux de *Chiméra* (théoriquement toujours identiques) et ceux de la séquence UniProt.

La fonction de *Proteinus* d'intersection va d'abord réaliser un parser du fichier PDB à l'aide de fonctions présentent dans la librairie *pyproteins*.

À l'issue de cette étape nous obtenons un dictionnaire résumant les acides aminés dont les positions matchs avec ceux inscrit dans le fichier PDB et la séquence UniProt.

```
[In 11] intersection(P04483.seq, fichier pdb, autre fichier de positions (chiméra  
par exemple), verbose=True)  
[Out ] msrlidKSKVINSALLELLNEVGIEGLTTRKLAQKLGVQPTLYWHVKNKRALLDALAIEMIDRHHTHECPLEGESW  
QDFLRLNNAKSERCALLSHRDGAKVHLCTRTEKCYETLENQLAFLCQQGFSIENAAYALSANGHFTLGCVLEDQEHQAKE  
ERETPTTDSMPLRQAIELFDHQGAEPAFLGLELIIICGLEKQLKCESgs
```

Perfect Match
PDB Match with Uniprot
PDB & Chimera Match, but no UniProt
No match

Encore une fois il est possible de visualiser ces résultats « colorés » sous formes de graphiques avec comme valeurs les différentes valeurs d'entropies en fonction de la position de la valeur d'entropie au sein de la séquence.

Résultats

Tableau de synthèse

	P04483	P0ACS9	P0A0N4	Q9AIU0
Description	TetR d' <i>E. coli</i> (spé. Tétracycline)	AcrR d' <i>E. coli</i>	QasR de <i>Staphylococcus aureus</i>	TtgR de <i>Pseudomonas putida</i>
Organisme	Escherichia	Escherichia	Staphylococcus	Pseudomonas
Longueur	207	215	188	210
Gene	tetR	acrR	qacR	ttgR
Homologues	8	100	100	100
Nombre de domaines	8	8	9	8
Type d'antibiotique	tétracycline	inconnu	inconnu	chloramphenicol ou tétracycline
Fichiers PDB	13	2	14	6
Couverture max	204	215	188	210

Tableau 1 de synthèse des résultats obtenus par ligne de commandes

Le tableau 1 présente les différents résultats obtenus lors de l'exécution du programme *Proteinus*. Les longueurs des séquences sont assez proches les unes des autres.

Les 4 gènes appartiennent à la famille TetR. Seule P04483 possède très peu de séquences homologues en comparaison avec les 3 autres. Il sera peut-être possible de réduire ces nombres en sélectionnant plus méticuleusement les séquences homologues.

Les protéines possèdent des fichiers PDB qui permettent de couvrir l'ensemble des protéines.

Principaux domaines

La liste des domaines protéiques disponible pour P04483 est résumé par le tableau 2. Ce tableau a été obtenu en utilisant les résultats de la commande [In 3].

Les 4 protéines à notre disposition possèdent toutes un domaine de fixation à l'ADN situé en partie N-ter de la protéine. Ce domaine reconnu comme étant le lieu de fixation à l'ADN possède une longueur de 61 acides aminés (AA) pour les 4 protéines.

La protéine P04483, possède également un domaine responsable de l'interaction avec la tétracycline. Ce domaine n'est pas un domaine continu, comme ce fut le cas pour celui de la fixation à l'ADN. En effet d'après ces données, 7 domaines sont responsables de la reconnaissance de l'antibiotique. À l'exception d'un domaine, ces domaines sont situés en partie C-ter de la protéine.

Le domaine de reconnaissance de la tétracycline est également retrouvé dans les autres protéines. Ce qui nous confirme notre affirmation donnée en introduction sur la superfamille des protéines TetR. De plus, cette observation nous permet de continuer notre analyse de recherche de similarités entre ces protéines. La visualisation de ses domaines conservés pourra nous indiquer plus précisément quels sont les AA impliqués dans les interactions protéines/ADN et protéines/antibiotiques.

Protein	Nom	Nombre de signatures	Start	Stop	Longueur
	DNA-binding HTH domain, TetR-type	5	3	63	61
	Tetracycline repressor TetR, C-terminal	1	69	201	133
			21	44	24
P04483	Tetracycline transcriptional regulator, TetR	5	75	118	44
			130	149	20
			183	197	15
	Homeobox-like domain superfamily	1	5	67	63
	Tetracyclin repressor-like, C-terminal domain superfamily	1	69	204	136
	DNA-binding HTH domain, TetR-type	5	10	70	61
P0ACS9	Transcription regulator MAATS, C-terminal	1	84	202	119
	Tetracyclin repressor-like, C-terminal domain superfamily	1	84	200	117
	Homeobox-like domain superfamily	1	1	68	68
	DNA-binding HTH domain, TetR-type	5	1	61	61
P0A0N4	Transcription regulator QacR, C-terminal	1	57	188	132
	Homeobox-like domain superfamily	1	3	65	63
	Tetracyclin repressor-like, C-terminal domain superfamily	1	73	186	114
	DNA-binding HTH domain, TetR-type	5	10	70	61
Q9AIU0	Transcription regulator MAATS, C-terminal	1	84	203	120
	Tetracyclin repressor-like, C-terminal domain superfamily	1	84	201	118
	Homeobox-like domain superfamily	1	2	76	75

Tableau 2: de synthèse des principaux domaines de notre jeu de données

WebLogo

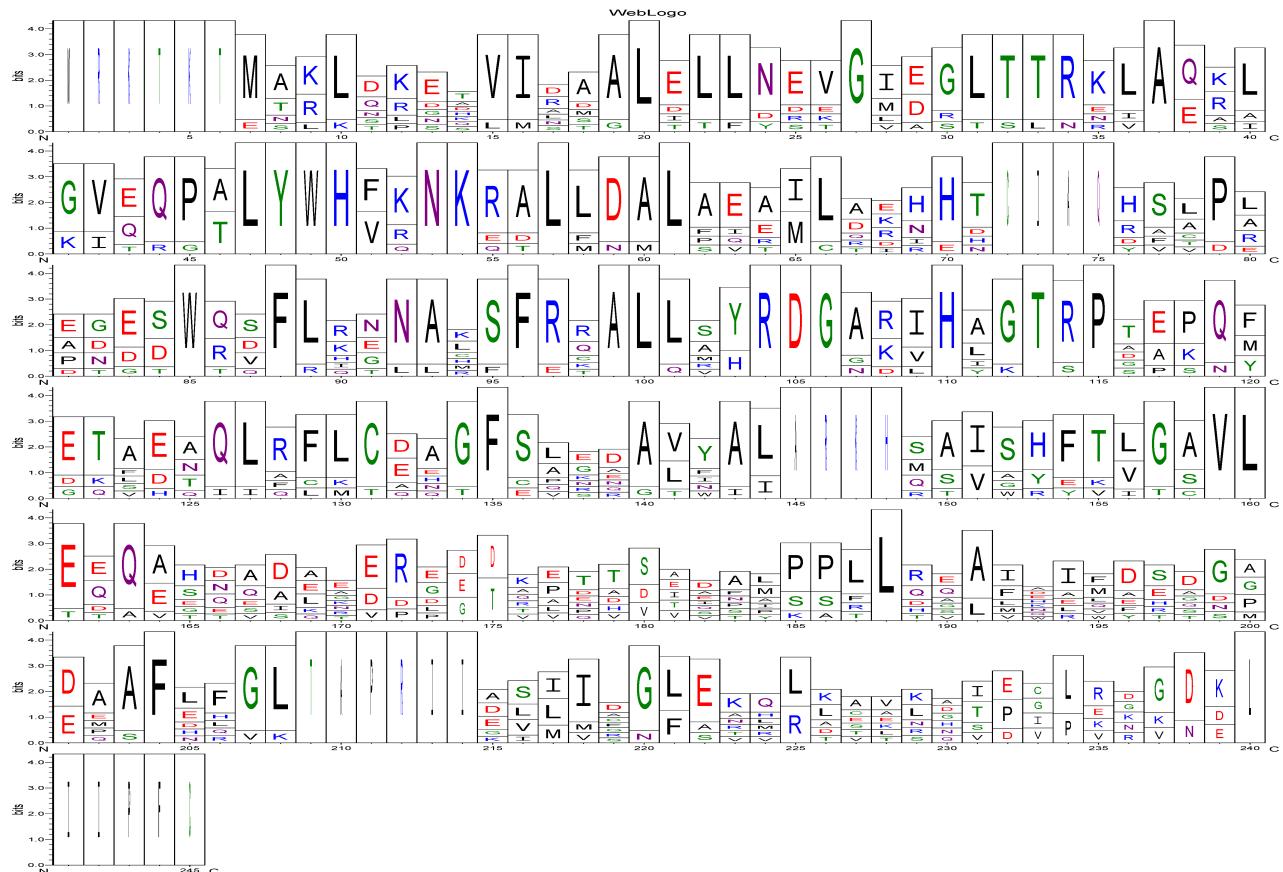


Figure 2: WebLogo obtenu à partir de l'alignement réalisé sur les séquences homologue de la protéine P04483

Cette visualisation permet de visualiser rapidement les AA qui sont fortement conservés. Ainsi on peut remarquer que la partie N-ter de la protéine est davantage conservé que sa partie C-ter.

En croisant ces constatations avec la position des domaines protéiques, nous pouvons en conclure que le domaine de fixation à l'ADN est mieux conservé que le domaine de fixation de la tétracycline.

Évolution de la conservation

L'exécution de la commande [In 10] permet d'enregistrer les valeurs, des entropies calculées, dans un attribut de l'objet *protein*. Cet attribut est `entropy`. Nous pouvons donc tracer un graphique (*Figure 3*) à partir de ces valeurs.

```
[In 12] plt.plot(P04483.entropy)
[In 13] plt.ylabel('entropyShannon', fontsize=30)
[In 14] plt.xlabel('position', fontsize=30)
[In 15] axhline(0, color='purple')
[In 16] plt.show()
```

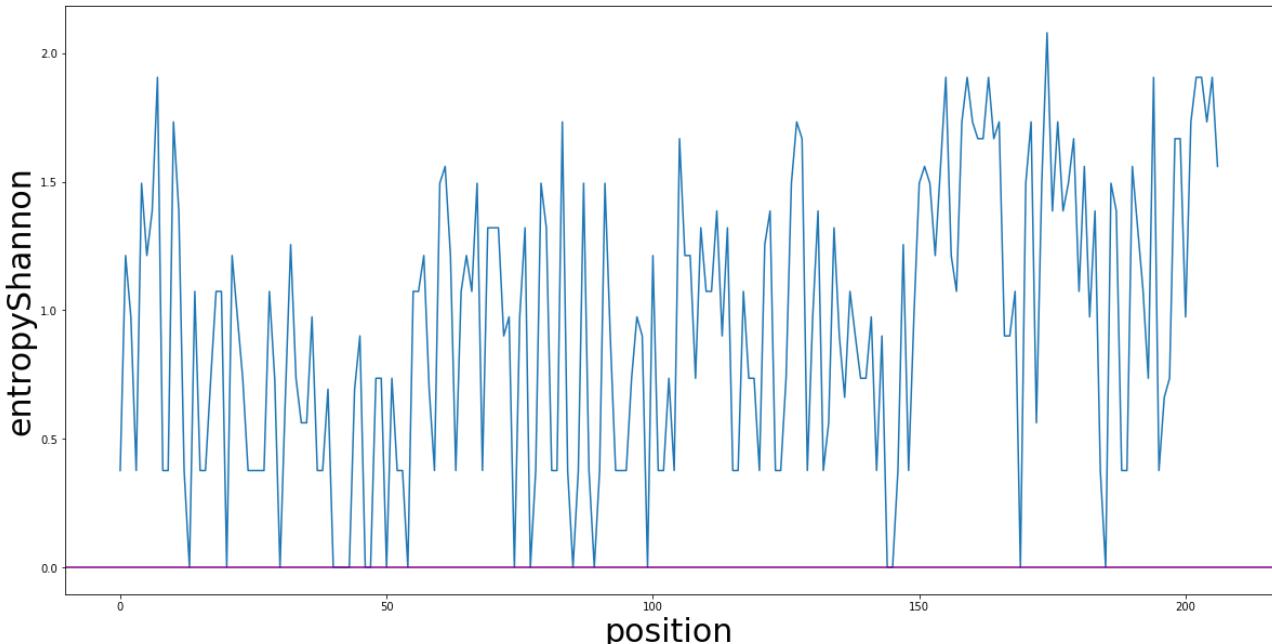


Figure 3: Graphique montrant l'évolution de l'entropie de Shannon en fonction de la position de l'acide aminé au sein de la protéine P04483.

À nouveau nous pouvons constater que la majorité des acides aminés conservés se situe dans les 100 premiers AA de la protein. Rappelons que d'après le tableau 2 il s'agit du domaine de fixation à l'ADN.

Intersections

Afin de répondre à la problématique générale il est intéressant de visualiser les AA qui sont en interaction avec les molécules environnant de la protéine (ADN ou antibiotique).

Nous allons à présent utiliser les informations contenues dans les fichiers PDB et les transposer sur nos résultats précédents.

PDB / Chiméra vs Uniprot

Correspondance entre les positions des AA donnés par le fichier PDB et ceux de la sélection opérée par le logiciel Chiméra. Les AA seront ensuite colorés en fonction de leur correspondance avec séquence Uniprot.

```
[In 17] intersection(P04483.seq, "/home/antoine/Documents/Stage2019/Stage/2vkv.pdb",
                     "/home/antoine/Documents/Stage2019/Stage/chimeraTet_selection.txt")
```



Figure 4: Intersection entre les données du fichier PDB (2vkv) et de la sélection opérée par le logiciel Chiméra avec la séquence UniProt de la protéine P04483

La Figure 4 permet de visualiser avec un jeu de couleur les AA d'intérêt.

Les AA indiqué sur fond blanc et en minuscule sont des AA qui ne figurent pas dans le fichier PDB et n'appartiennent pas à la sélection faite par Chiméra.

En vert il s'agit des AA qui sont retrouvés à la même position dans les fichiers PDB, par Chiméra et sur la séquence UniProt.

En orange il s'agit d'un match partiel. C'est-à-dire que la position d'AA donnée uniquement par le fichier PDB correspond également à la position de la séquence UniProt.

En bleu, seules les positions de l'AA donnée par Chiméra ou du fichier PDB sont identiques mais pas avec celle de la séquence d'UniProt. Attention l'AA affiché est celui de la séquence d'origine (UniProt).

Enfin en rouge nous retrouvons les AA dont les positions entre le fichier PDB est différent de celle fourni par Chiméra et de celle fournie par le fichier PDB.

Graphique de correspondance des AA avec leur niveau de conservation

Il est également possible de représenter les AA de la Figure 4 sur le graphique montrant l'évolution de l'entropie (Figure 3). Les AA choisis pour être affichés sont ceux dont les conditions correspondent aux couleurs verte, bleue et orange de la Figure 4. De plus il est également possible d'afficher ces AA avec des points de couleurs identifiant leur appartenance à une fonction biologique donnée.

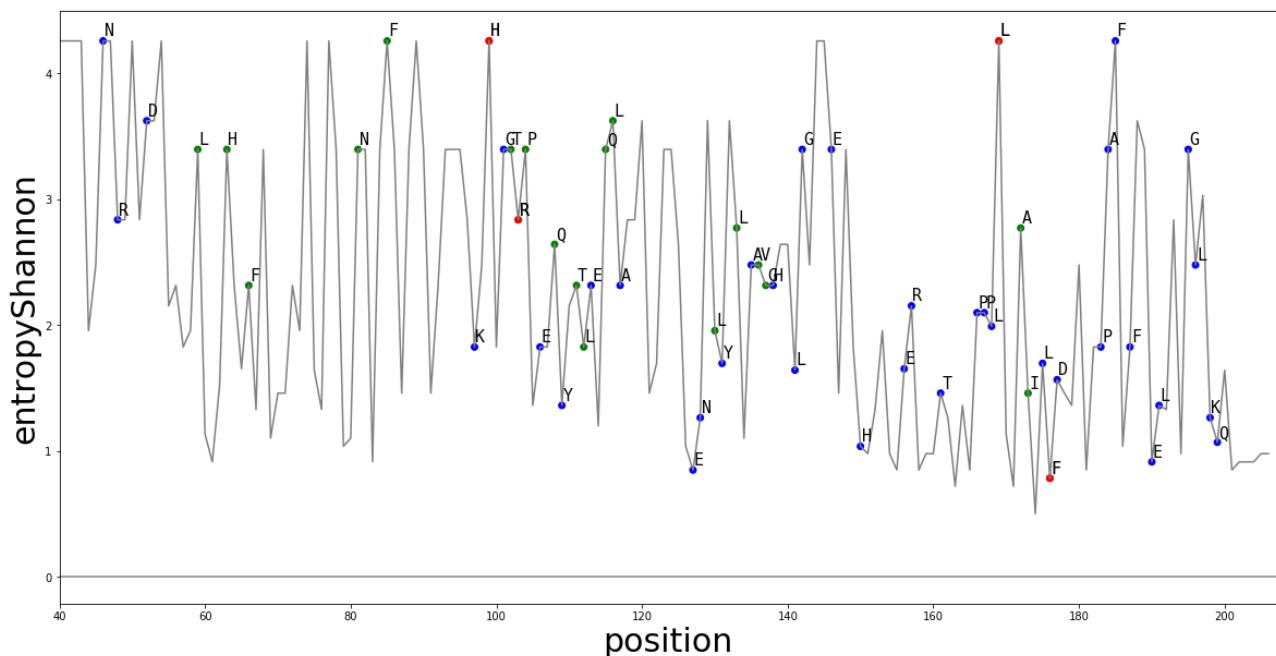


Figure 5: Graphique montrant le niveau de conservation d'un acide aminé en fonction de sa position sur la séquence protéique de P04483. Les acides aminés d'intérêt ont été affichés sur le graphique. En vert, il s'agit des AA interagissant avec la tétracycline ; en bleu ceux responsables de la formation du dimer de la protéine. Le rouge est la combinaison du bleu et du vert.

La Figure 5 indique que les AA responsables de la formation du dimer sont plus nombreux que ceux interagissant avec la tétracycline.

Cependant, il n'est pas possible à ce stade de pouvoir conclure sur le niveaux de conservation de ces AA.

Conservation de domaines

Boxplot des entropies des AA impliqués dans les domaines d'interactions avec la tétracycline, responsable de la formation du dimer et des autres AA.

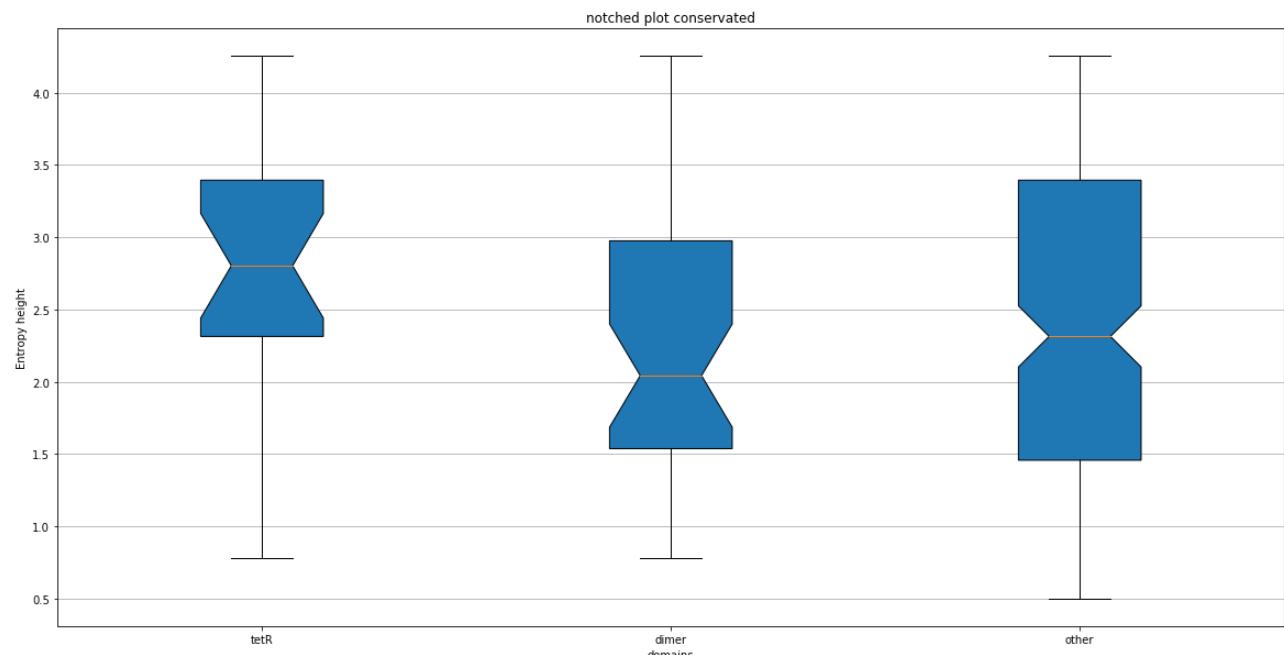


Figure 6: Boxplot des niveaux de conservations entre les acides aminés des domaines d'intéraction avec la tétracycline (TetR), ceux impliqués dans la formation du dimer et des autres acides aminés de la protéines.

La Figure 6 indique que les AA impliqués dans la formation du dimer sont les plus conservées. En revanche, ceux impliqués dans l'interaction avec la tétracycline sont moins conservés.

Conclusion et Discussions

À l'issue de ce rapport nous pouvons dire que le programme *Proteinus* répond à notre demande d'analyse rapide des protéines.

En effet grâce à l'ensemble de ses fonctions nous avons pu déterminer et visualiser des domaines protéiques d'intérêt biologique. Comme les domaines de fixations à l'ADN, d'interaction avec la tétracycline et de formation du dimer. De plus, nous sommes en mesure de représenter graphiquement leurs niveaux de conservation et de pouvoir cibler avec précision les AA les plus conservés au sein des domaines d'intérêts.

L'analyse statistique des niveaux de conservations nous révèle que le dimer protéique est une partie de la protéine qui est conservée au sein des protéines homologues.

Des études expérimentales axées sur le dimer protéique pourraient fournir des informations caractéristiques sur la nature physico-chimique de ce domaine protéique et pourrait peut-être aboutir sur une nouvelle approche de lutte contre cette protéine.

Remerciements

Je remercie particulièrement ma maître de stage Juliette Martin pour m'avoir accepté dans son équipe, également pour m'avoir fait confiance dans mon travail. Je la remercie également pour sa disponibilité et son écoute pendant toute la durée de mon stage. J'ai apprécié ses connaissances et ses conseils qui m'ont aidé à surmonter les difficultés.

Je remercie ensuite Guillaume Launay, vers qui je pouvais me tourner pour demander conseil pour le master de bio-informatique et pour avoir des conseils et des retours sur la programmation en Python. Enfin je n'oublie pas les membres de l'équipe du *mmsb* et du *mobi* avec lesquels partager une pause café, ou des conseils, étaient toujours très conviviale.

Références

1. tetR - Tetracycline repressor protein class B from transposon Tn10 - Escherichia coli - tetR gene & protein. Available at: <https://www.uniprot.org/uniprot/P04483>. (Accessed: 26th April 2019)
2. Cuthbertson, L. & Nodwell, J. R. The TetR Family of Regulators. *Microbiol Mol Biol Rev* **77**, 440–475 (2013).
3. Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST). Available at: https://www.ics.uci.edu/%7Efielding/pubs/dissertation/rest_arch_style.htm. (Accessed: 26th April 2019)
4. Jones, P. *et al.* InterProScan 5: genome-scale protein function classification. *Bioinformatics* **30**, 1236–1240 (2014).
5. Altschul, S. F. *et al.* Protein database searches using compositionally adjusted substitution matrices. *FEBS J.* **272**, 5101–5109 (2005).
6. Sievers, F. *et al.* Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Mol. Syst. Biol.* **7**, 539 (2011).
7. Clustal. *Wikipedia* (2019).
8. Crooks, G. E., Hon, G., Chandonia, J.-M. & Brenner, S. E. WebLogo: A Sequence Logo Generator. *Genome Res.* **14**, 1188–1190 (2004).
9. G.Launay. *toolbox to manipulate sequence related data. Contribute to glaunay/pyproteins development by creating an account on GitHub.* (2019).
10. G.Launay. *Contribute to glaunay/pyproteinsExt development by creating an account on GitHub.* (2019).
11. Pettersen, E. F. *et al.* UCSF Chimera--a visualization system for exploratory research and analysis. *J. Comput. Chem.* **25**, 1605–1612 (2004).

Codes sources

L'intégralité du code de Protéinus est disponible sur la plateforme *Github* à l'adresse suivante :
<https://github.com/AntoineHeurtel/TetR-Binding-Motif>

Le code de la librairie *pypyroteins* est disponible à cette adresse : <https://github.com/glaunay/pypyroteins>

Le code de la librairie *pypyroteinsExt* est disponible à cette adresse : <https://github.com/glaunay/pypyroteinsExt>