Biopython & BioStructures
Evaluation de la manipulation de fichier pdb en python et Julia

2020-2021

IMBERT Pierre

Table des matières

Introduction

Les protéines sont des macromolécules biologiques présentes dans toutes les cellules vivantes. Elles sont constituées d'une chaîne polypeptidique dont les acides aminés consécutifs les composant ont tendance à se replier en modules plus ou moins compacts, appelés domaines. Les structures tertiaires abordées dans cette étude tertiaire correspondent à l'arrangement dans l'espace et à la succession de structures locales appelées structures secondaires, elles-mêmes constituées d'un ou plusieurs domaines. Le repliement d'une protéine est un processus complexe dont la bonne conformation assure la fonction biologique de la protéine. Sachant que le nombre de protéines disponible dans la Protein Data Bank (PDB) [6] augmente de manière exponentielle - il est à l'heure d'aujourd'hui d'environ 175 000 - il est essentiel d'utiliser des modules/packages pour travailler sur ces protéines. Ils permettent i) de traiter des fichiers pdb - téléchargement et chargement - et ii) de les analyser - visualisation, alignement de séquences, de structures, carte de contacts, etc. Cela doit évidemment être réalisé de manière efficace. Ce rapport présente une application et évaluation de l'un de ces modules et de l'un de ces packages : le module python Biopython [2] & le package Julia BioStructures [3].

Biopython est un ensemble d'outils de bio-informatique implémentés en python. Python (https://www.python.org/) est un langage de programmation interprété en libre accès largement utilisé autour du monde et dont la popularité ne cesse de croître d'après le nombre de recherches du terme python (le langage) par rapport à Java, C++, etc. selon Google Trends. Parmi les modules disponibles, on s'intéresse essentiellement à Bio.PDB qui permet le traitement et l'analyse de fichier PDB. La version de Biopython évaluée est la 1.78, publiée en septembre 2020 et qui est la dernière sortie à ce jour, le 19 mars 2021. Précisons que les programmes mis en place tournent sous la version Python v3.7.8.

BioStructures est le premier package implémenté en Julia ; il est destiné au traitement et à l'analyse de fichier PDB. Julia (https://julialang.org/) est un langage de programmation récent ; la première version publique est sortie en 2012. Il est assez populaire comme en attestent plus de 25 millions de téléchargements. Sa version 1.0 est sortie en 2018. Julia est présenté comme un langage de programmation de haut niveau et performant, qui s'affranchit des limitations propres à d'autres langages de haut niveau, i.e. l'écriture d'un code prototype dans un premier langage, par exemple python, avant de le réécrire dans une "version performante" avec un autre langage, par exemple C ou C++. Julia présente une syntaxe et des structures de données

relativement proches de python, que ce soit i) le parcour des éléments d'une liste à l'aide d'une boucle for (il est possible de parcourir une liste à l'aide de l'index mais ce n'est pas une nécessité), ii) la manipulation de liste ; l'ajout d'un élément (append), l'accès au premier élément - liste[1] en Julia (index à partir de 1) et liste[0] en python, le tri de listes, etc., iii) les listes en compréhension, iv) les dictionnaires de données, etc. Si vous désirez avoir un aperçu plus approfondi des spécificités de Julia, veuillez vous référer au jupyter notebook *julia-cheat-sheet.ipynb*, basé sur le livre Julia 1.0 programming [1]. Le notebook est disponible sur https://github.com/Pierre-damase/Projet-julia/blob/master/julia-cheat-sheet.ipynb. La version de BioStructures évaluée dans le cas présent est la 0.11.6, publié fin décembre 2020, basé sur Julia 1.4.1. En ce qui concerne l'alignement de séquences, nous avons utilisé le package Julia BioAlignements (version 2.0.0) comme c'est indiqué dans la documentation fournie par BioStructures (https://biojulia.net/BioStructures.jl/stable/documentation/).

Le rapport suivant présente une étude comparative d'analyses de bio-informatique structurale entre Biopython et BioStructures. Le but est de présenter dans un premier temps les manipulations réalisables avec ces deux outils et d'expliquer comment les mettre en place. Dans un deuxième temps il s'agit de les évaluer en réalisant un benchmark.

Matériel & Méthodes

Les différentes analyses mises en place sont le i) téléchargement et la lecture de fichier PDB à partir du PDB ID, ii) la visualisation de structures, iii) l'alignement de séquences, iv) l'alignement de structures, v) et les cartes de contacts. Vous pouvez retrouver l'ensemble des programmes sur le lien github suivant: https://github.com/Pierre-damase/Projet-julia.

Prérequis

L'utilisation de Miniconda3 (https://docs.conda.io/en/latest/miniconda.html) est fortement recommandée pour l'utilisation des différents programmes mis en place, le module python prot_py et les jupyter notebooks. L'environnement conda utilisé est disponible sur le github *Projet-julia* et peut être initialisé à l'aide de la commande *conda env create --file julia.yml* ; par la suite pour l'activer il suffit d'utiliser la commande *conda activate julia*. Les différents modules utilisés sont notamment i) le module *Bio.PDB* de *Biopython* pour manipuler les fichiers PDB et

les structures, ii) *numpy (https://numpy.org/) & pandas* (https://pandas.pydata.org/) pour la manipulation des données, iii) *matplotlib* [4] *& seaborn* [5] pour réaliser les graphiques du benchmark, iv) *nglview* (https://github.com/nglviewer/nglview) pour une visualisation interactive de protéines dans un jupyter notebook, v) *argparse*, un parser d'arguments en lignes de commandes (https://docs.python.org/3/library/argparse.html), et vi) *Bio.Pairwise2* pour l'alignement de séquences (https://biopython.org/docs/1.75/api/Bio.pairwise2.html).

En ce qui concerne Julia, son installation préalable est requise via la commande *sudo apt install julia*. De plus, Julia n'ayant pas d'équivalent à miniconda, l'ensemble des packages doit être installé manuellement. Pour cela, il faut exécuter la commande *julia* pour pouvoir utiliser de manière interactive Julia dans le terminal soit le REPL (read-eval-print loop). Toute installation de package en Julia se fait à l'aide du gestionnaire de package Pkg (https://docs.julialang.org/en/v1/stdlib/Pkg/) activé avec la commande *using Pkg*. Ensuite, la commande *Pkg.add("NOM")* installe le package NOM. L'ensemble des packages Julia nécessaires sont: i) *IJulia* pour pouvoir utiliser Julia de manière interactive avec un jupyter notebook (https://github.com/JuliaLang/IJulia.jl), ii) *ArgParse*, un parser d'arguments en lignes de commandes (https://github.com/carlobaldassi/ArgParse.jl) proche du module python *argparse*, iii) *BioAlignments* pour l'alignement de séquences, iv) *BioStructures* pour le téléchargement/lecture de fichier PDB, la manipulation de structures, l'alignement de structures, les cartes de contacts, v) *Bio3DView* pour une visualisation interactive de protéines dans un jupyter notebook (https://github.com/jgreener64/Bio3DView.jl), vi) *NaturalSort* pour trier une liste de strings dans l'ordre naturel, i.e. ["fichier-1", "fichier-2", "fichier-10"] et non pas ["fichier-1", "fichier-10", "fichier-2"] (https://github.com/JuliaStrings/NaturalSort.jl), et vii) *Suppressor* pour la suppression des warnings utilisé uniquement lors du benchmark (https://github.com/JuliaIO/Suppressor.jl).

Pour la version python, a été mis en place un module appelé **prot_py**. Son utilisation nécessite d'avoir au préalable initialisé et activé l'environnement conda *julia*, ou de disposer a minima d'une distribution python 3.7.8 sur son ordinateur et d'avoir installé l'ensemble des modules requis. Pour l'exécuter, il faut être situé dans le dossier ./Projet-julia/Code et exécuter la commande suivante:

```
python -m prot_py -i ID -e ARG -a ALIGN -c CUTOFF
```

L'option -m permet d'exécuter prot_py en tant que module python.

Les arguments nécessaires sont i) *ID* pour l'id du ou des fichiers PDB à étudier - si les fichiers PDB renseignés ne sont pas présents dans le dossier ./Projet-julia/Code/data/pdb ils sont automatiquement téléchargés puis lus sinon ils sont directement lus - et ii) *ARG* pour l'étude à réaliser - *view* pour la visualisation, *align* pour l'alignement de séquences, *rmsd* pour l'alignement de structures et le calcul du rmsd et *maps* pour générer la carte de contact. Dans le cas d'*align* & *rmsd*, deux PDB ID doivent être renseignées.
Les arguments facultatifs sont i) *ALIGN* pour indiquer le type d'alignement de séquences à réaliser, soit *global* ou *region*, se référer à la partie alignement de séquences, et ii) *CUTOFF* pour indiquer le cutoff à appliquer à la carte de contacts, compris entre 6 et 12 Angstrom (vaut 10A par défaut).

Pour la version Julia, les différentes méthodes mises en place sont disponibles dans le package **Prot** situé dans le dossier ./Projet-julia/Code. Une fonction *main.jl* est également disponible pour pouvoir exécuter les méthodes à l'aide de la ligne de commande suivante:

```
julia main.jl -i ID -e ARG -a ALIGN -c CUTOFF
```

Les arguments nécessaires et facultatifs sont identiques à ceux du module **prot_py**.

Ont également été mis en place deux jupyter notebooks, *prot-py.ipynb & prot-jl.ipynb*, l'un pour la version python et l'autre pour la version Julia respectivement. Ces deux notebooks mettent en place des exemples interactifs d'application du module python **prot_py** et du package Julia **Prot** pour un jeu de protéines donné. Le notebook *prot-py.ipynb* permet également la génération des sous-structures utilisées pour le benchmark ainsi que de la réalisation des graphiques du benchmark, i.e. la lecture des données du benchmark, le traitement des données puis la génération des graphiques.

Les données

Les données utilisées sont des structures issues de la Protein Data Bank (PDB). Il a été choisi de ne considérer que les fichiers au format PDB, fichiers organisés en deux parties. La première est l'entête qui fournit des informations sur la structure - le nom de la protéine, les chaînes dont elle est constituée ainsi que leur nom, l'organisme dont provient la protéine, la fonction, la méthode expérimentale employé pour déterminer la structure, la séquence en acides aminés de la protéine, etc.

```
HEADER    OXYGEN TRANSPORT                        08-DEC-97   1A01
TITLE     HEMOGLOBIN (VAL BETA1 MET, TRP BETA37 ALA) MUTANT
COMPND   2 MOLECULE: HEMOGLOBIN (ALPHA CHAIN);
COMPND   3 CHAIN: A, C;
[...]
COMPND   6 MOLECULE: HEMOGLOBIN (BETA CHAIN);
COMPND   7 CHAIN: B, D;
[...]
SOURCE   2 ORGANISM_SCIENTIFIC: HOMO SAPIENS;
SOURCE   3 ORGANISM_COMMON: HUMAN;
[...]
KEYWDS    OXYGEN TRANSPORT
EXPDTA    X-RAY DIFFRACTION
[...]
```

La deuxième partie du fichier pdb indique les coordonnées de chaque atome de la protéine. Sont représentées ci-dessous les coordonnées des atomes de la valine 1 situé sur la chaîne A. Dans l'ordre est indiqué: i) le numéro de l'atome, ii) le nom de l'atome, iii) le code à 3 lettres du résidu dont fait partie l'atome, iv) la chaîne d'appartenance du résidu, v) le numéro du résidu dans la protéine, vi) les coordonnées x, y et z de l'atome, vii) le facteur d'occupation de l'atome, i.e. la probabilité de trouver l'atome à cette position donnée (1.00 pour 100%), viii) le facteur de température, et ix) l'élément chimique de l'atome.

```
[...]
ATOM      1  N   VAL A   1      19.478  29.066  43.555  1.00 40.28           N
ATOM      2  CA  VAL A   1      20.245  30.016  42.747  1.00 39.74           C
ATOM      3  C   VAL A   1      21.747  29.694  42.886  1.00 35.09           C
ATOM      4  O   VAL A   1      22.165  29.145  43.911  1.00 35.76           O
ATOM      5  CB  VAL A   1      20.034  31.481  43.171  1.00 50.59           C
ATOM      6  CG1 VAL A   1      20.116  32.423  41.971  1.00 56.83           C
ATOM      7  CG2 VAL A   1      18.801  31.673  44.020  1.00 63.10           C
[...]
```

Biopython et BioStructures permettent également de lire des fichiers au format mmCIF. Cependant, il a été choisi de ne pas implémenter la lecture de tels fichiers ni d'en évaluer le chargement.

Dans l'exemple intéractif, 4 protéines sont étudiées: l'hémoglobine humaine (code 1A01) et bovine (code 2QSP), la capside du virus de l'hépatite B (code 2G33) et un assemblage de structures (code 1HTQ). L'hémoglobine est une protéine présente dans le sang des vertébrés au sein des globules rouges; elle assure le transport de l'Oxygène. L'hémoglobine est constituée de deux sous-unités, alpha et beta, codées par deux gènes. Les phylogénies de ces

gènes ont permis de se rendre compte que les événements de duplication (séparation d'un gène ancestral en deux, ici alpha et beta) précèdent des événements de spéciation (séparation d'un ancêtre commun en nouvelles espèces) [7]. 1A01 et 2QSP sont deux structures obtenues par x-ray diffraction, de résolution 1.80A et 1.85A respectivement et de masse 64 kDa. La troisième structure considérée est celle de la capside du virus de l'hépatite B (HBV) qui correspond à l'enveloppe protéique du virus. Cette enveloppe est constituée de 240 copies d'une protéine appelée HBc, soit 120 dimères. 2QSP a été obtenue par x-ray diffraction, a une résolution de 3.96A et a une masse de 67 kDa. 1HTQ est une structure cristallographique multicopie d'une glutamine synthétase détendue. La structure de la macromolécule a été obtenue par x-ray diffraction et a une résolution de 2.40A. De plus, cette structure a une masse de 1298 kDA.

La structure des données chargées avec Biopython et BioStructures est équivalente, le deuxième s'étant inspiré du premier : i) une structure constituée de modèles, ii) un modèle lui-même constitué de chaînes, iii) les chaînes formées d'un ensemble de résidus, et iv) les résidus qui sont une suite d'atomes caractérisés notamment par leur coordonnées x, y et z.

Visualisation

Biopython et BioStructures permettent la visualisation de structures, soit en appelant des programmes externes tels que PyMol (https://pymol.org/2/), soit de manière interactive dans un jupyter notebook. Dans le deuxième cas, il est possible de changer le style de visualisation (cartoon, surface, etc.) et la couleur de la structure. La personnalisation reste cependant très limitée par rapport à PyMol par exemple. Il est important de noter que Biopython et BioStructures ne cherchent en aucun cas à recréer un autre logiciel de visualisation de structures chimiques mais à manipuler et analyser des structures. On peut ainsi expliquer ce manque de personnalisation, notamment sur la sélection de chaînes et/ou d'éléments spécifiques.

Avec Biopython, la visualisation se fait de la manière suivante à l'aide du module nglview :

```
import nglview as nv  # Importer le module
view = nv.show_biopython(structure)
view.add_representation('surface', color='red', selection='protein')
view
```

La méthode *add_representation* permet de spécifier représentation et couleur de la structure.

Avec BioStructures, la visualisation se fait ainsi à l'aide du package Bio3DView :

```
style = Style("sphere")
viewstruc(structure[1], style=style)
```

Il est également possible de spécifier un style de représentation en Julia.

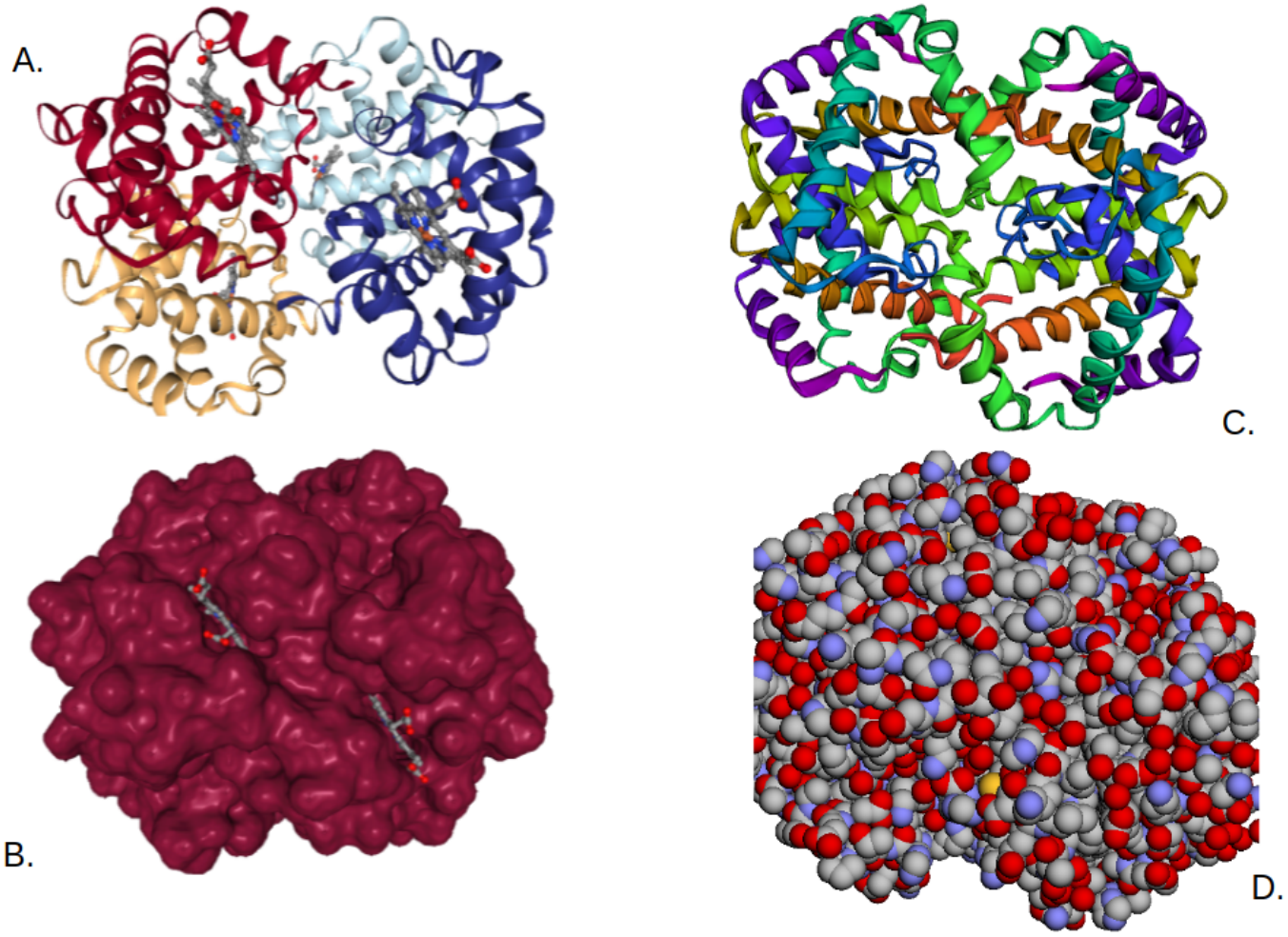

Figure 1: Visualisation de structures pour l'hémoglobine humaine 1A01 de manière interactive dans un jupyter notebook avec : i) **Biopython: A**. représentation cartoon & **B**. représentation surface et couleur rouge bordeaux, et ii) **BioStructures: C**. représentation cartoon & **D**. représentation sphère. Ces représentations sont disponibles dans les deux jupyter notebooks prot-py.ipynb pour Biopython et prot-jl.ipynb pour BioStructures.

Lien github https://github.com/Pierre-damase/Projet-julia/tree/master/Code des jupyter notebook.

Alignement de séquences

L'alignement de séquences consiste à comparer des séquences de macromolécules de telle sorte que l'on mette en lumière les similarités entre ces séquences. Ainsi, une non correspondance entre deux acides aminés à une position donnée peut renseigner sur la présence d'une mutation, et un "gap" peut indiquer de manière plus spécifique une mutation par insertion ou délétion dans l'une des séquences. Que ce soit pour la version python ou Julia, la même méthode d'alignement a été choisie. Dans les deux cas, il n'a été réalisé que des alignements globaux, i.e. des alignements qui déterminent la meilleure concordance entre tous les acides aminés des deux séquences. De plus, la matrice de score BLOSUM62 a été utilisée; elle renseigne sur la probabilité qu'une paire de résidus observée soit biologiquement vraisemblable et non le fruit du hasard. En cas de nouveau gap, signe d'insertions ou de délétions dans une des deux séquences, un score de -10 est considéré. Si c'est la suite d'un gap précédent, le score n'est alors que de -1. Deux types d'alignements sont possibles, soit un alignement global de deux protéines, soit un alignement global mais des régions, i.e. alignement de la chaîne A de la protéine 1 avec celle de la protéine 2, alignement de la chaîne B de la protéine 1 avec celle de la 2, et ainsi de suite. Le 2ème cas de figure peut par exemple être intéressant dans le cas de l'Hémoglobine pour déterminer si deux chaînes issus de deux espèces différentes sont similaires.

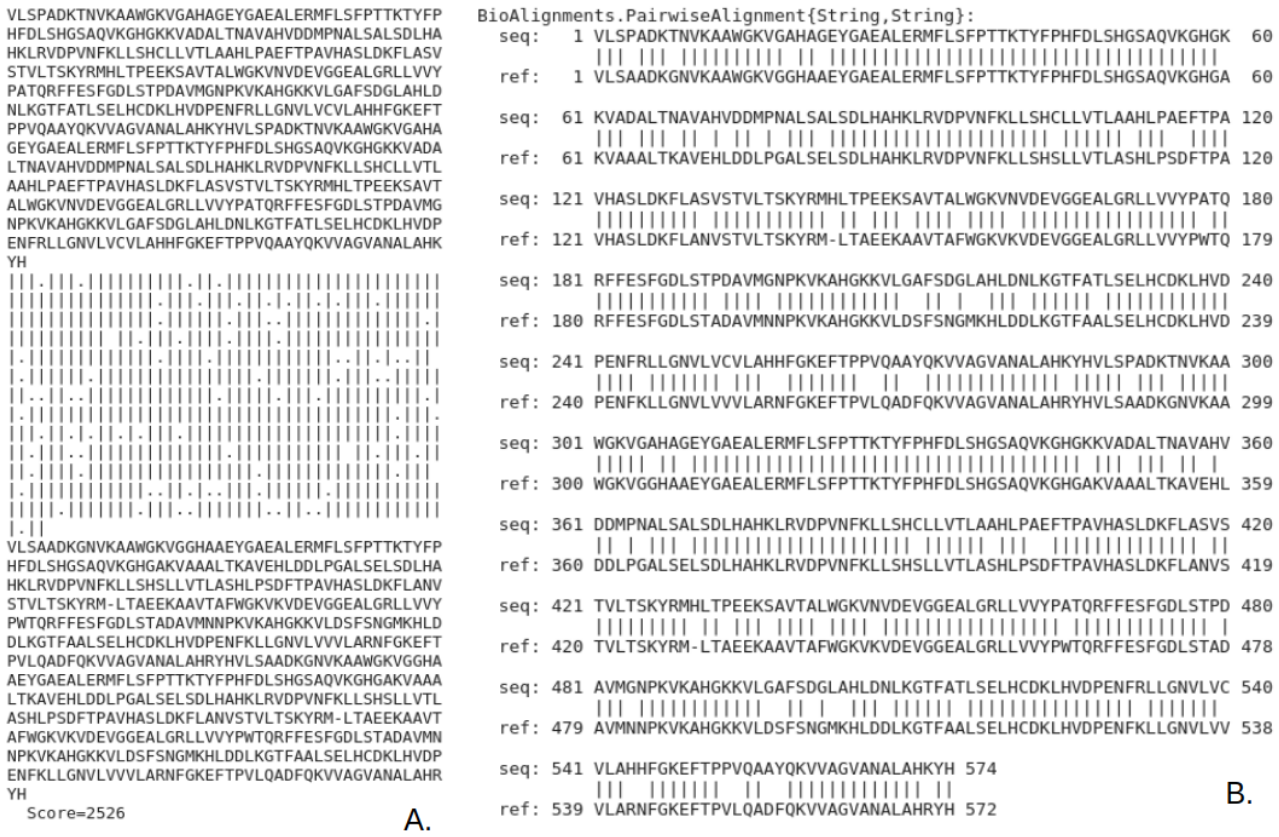

Figure 2: Alignement deux à deux de l'Hémoglobine humaine et bovine. **A**. alignement obtenu avec le module python *Bio.Pairwise2*, **B**. alignement obtenu avec le package Julia BioAlignments.

Alignements de structures

Le nombre de structures résolu sur la PDB est environ de 175 000 à ce jour et le nombre de séquences disponibles dans genbank est de l'ordre du milliard [8]. De plus, les structures protéiques ont été classées selon leurs similitudes structurales. Ainsi, à ce jour il y a environ 2000 superfamilles de repliements protéiques déterminées pour plus de 100 000 entrées PDB dans la dernière version de la classification structurale SCOPe [9] (2021). Le nombre de familles de protéines ayant des repliements différents est donc étonnamment bas par rapport à la diversité des séquences observées. Ces données représentent un argument fort en faveur de l'hypothèse que les structures de protéines, et donc des repliements qu'elles adoptent, sont bien plus souvent conservées au cours du temps que les séquences. Ceci peut se traduire notamment par des protéines dont la séquence est éloignée mais qui présentent un repliement similaire et ont donc une fonction équivalente. L'alignement de structures représente donc une étape essentielle d'analyses de bio-informatique structurale.

Concrètement, l'alignement de structures consiste à comparer les structures tertiaires de macromolécules de sorte que l'on mette en lumière les similarités dans le repliement adopté par ces structures. L'alignement de structures fait intervenir le RMSD (cf équation 1) qui est la distance moyenne entre les atomes de la structure 1 et ceux de la structure 2.

$$(1)\ RMSD = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(r_i^{s1} - r_i^{s2})^2}$$

Soit r^{s1} et r^{s2} deux ensembles constitués de N atomes, chaque atome étant caractérisé par leur position x, y et z. Dans le cas du programme Julia, seuls les carbones alpha (Cα) des acides aminés sont considérés. Dans le cas du programme python, si le résidu i de la structure 1 et 2 est identique, l'ensemble de la chaîne principale (backbone) du résidu est considérée pour l'alignement. Sinon, seul le carbone alpha est considéré. Cette considération intervient dans la sélection des atomes car les deux ensembles r^{s1} et r^{s2} doivent être de taille identique N. Il est à noter que, par simplification lors de la sélection des atomes, l'atome de carbone de la glycine, bien qu'il soit non chiral, est également noté Cα par Biopython et BioStructures. De plus, il est possible de ne considérer que les carbones beta (C$\boldsymbol{\beta}$) pour le calcul du RMSD; cela n'a pas été implémenté et peut faire office d'une extension. Concernant l'alignement, Biopython et BioStructures font intervenir une méthode de "superimposition" qui modifie les coordonnées d'un ensemble d'atome 1 pour les superposer à celles de l'ensemble 2. Dans le cas de Biopython, cette étape a également pour but de modifier les coordonnées de telle sorte que le RMSD soit minimisé.

Pour l'alignement de l'hémoglobine humaine (1A01) avec la bovine (2QSP), on obtient un rmsd de 2.9A avec le programme python et 2.6A avec le programme Julia. Les deux programmes renvoient donc un RMSD très proche, à plus ou moins 0.3A. De plus, un RMSD faible, inférieur à 3A, renseigne sur deux protéines dont la structure est très proche. Observer un tel RMSD pour l'alignement de l'hémoglobine humaine et bovine semble donc cohérent, sachant que ces deux protéines sont issues d'un même ancêtre commun.

Carte de contacts

La carte de contacts d'une protéine renseigne sur la distance entre chaque paire de résidus d'une structure donnée. Par simplification, il n'est pris en compte que les carbones alpha. La distance euclidiennes entre deux atomes est définie comme suit :

$$(2)\ \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2 + (z_B - z_A)^2}$$

(x_A, y_A, z_A) et (x_B, y_B, z_B) représentent respectivement les coordonnées des atomes A et B. Pour construire une carte de contacts, il faut donc au préalable générer la matrice de distance euclidienne entre tous les atomes qui est de taille N*N. La taille de la structure a donc potentiellement un impact important sur la complexité en espace de l'algorithme, i.e. la quantité d'espace mémoire utilisée. Ensuite, une fois la matrice de distance formée, on peut la transformer en carte de contacts en appliquant un cutoff compris entre 6 et 12A. Si la distance entre les atomes A et B est inférieur au cutoff alors le contact vaudra 1, i.e. que les deux atomes sont proches dans l'espace. A l'inverse, si la distance entre les atomes A et B est supérieure au cutoff alors le contact vaudra 0, i.e. que les deux atomes sont éloignés dans l'espace. Ainsi, une carte de contacts permet d'avoir une représentation en 2 dimensions d'une structure de protéine en 3 dimensions. Par cela j'entends que la carte de contacts permet de mettre en lumière dans un plan 2D pour un cutoff donné les atomes qui sont proches et donc en interaction dans la structure 3D.

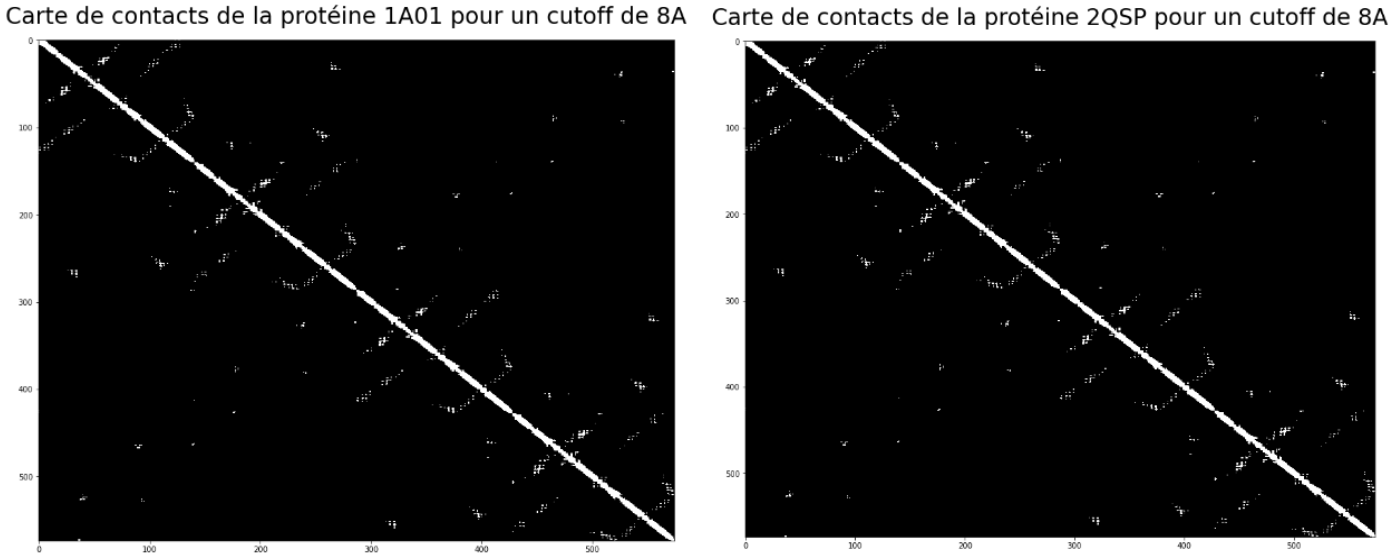

Figure 3: Carte de contacts pour l'hémoglobine humaine (à gauche) et l'hémoglobine bovine (à droite) pour un cutoff de 8 Angstrom. En blanc sont renseignées les paires d'atomes dont le contact vaut 1, et en noir celles où le contact vaut 0. La diagonale de la matrice caractérise les paires d'atomes identiques, soit A avec A, B avec B, et ainsi de suite, d'où la présence d'une diagonale blanche. De plus, on remarque que la carte de contacts de l'hémoglobine humaine et bovine sont globalement similaires. Ceci est cohérent avec l'alignement des structures de ces deux protéines dont le RMSD est inférieur à 3A, une valeur de RMSD qui indique que la structure des deux protéines est très proche.

Protocole de benchmark

Une problématique importante revient souvent en bio-informatique, le temps d'exécution d'un programme. C'est pourquoi, je vais vous présenter un benchmark des différentes méthodes que je vous ai présentées. Concernant le benchmark, il a été réalisé sous Ubuntu 20.04.2 LTS avec le processeur de 6ème génération Intel(R) Core(TM) i5-6600K CPU @ 3.50GHz et 24 Go de RAM DDR4 cadencée à 2400 MHz.

La première étape du benchmark est la création d'un ensemble de sous-structures dont la masse varie de 5 à 100 kDa. Cette fourchette de valeurs a été choisie car la majorité des protéines a une masse comprise entre 20 et 60 kDa. De plus, une protéine dont la masse est inférieure à 20 kDa est considérée comme petite; à l'inverse celle dont la masse est supérieure à 60 kDa est considérée grosse. Ainsi, la fourchette choisie, de 5 à 100 kDa, semble être la plus représentative du vivant. Pour la création des sous-structures, j'ai utilise 1CRN

(https://www.rcsb.org/structure/1CRN) comme structure modèle, une petite structure de 5 kDa environ qui est constituée d'une unique chaîne A. A partir de cette structure modèle, je génère 20 sous-structures de taille 5, 10, 15 et ainsi de suite jusqu'à 100 kDa. Chacune des sous-structures est constituée d'une unique copie d'1CRN (5 kDa), de deux copies d'1CRN (10 kDa), …, et de vingt copies d'1CRN (100 kDa).

Ensuite, une fois le jeu de données créé, le protocole de benchmark suivant est appliqué (programme python et Julia): il est réalisé 100 itérations de chacune des méthodes implémentées, soit le chargement de structures, l'alignement de séquences (global ou région), l'alignement de structures et les cartes de contacts. A chacune des itérations, le temps d'exécution moyen est sauvegardé et le résultat de la méthode écarté car il ne rentre pas en ligne de compte. En python, la méthode *process_time()* a été préférée à *time()* car plus précise. Ce sont deux méthodes du module time (https://docs.python.org/3/library/time.html). En Julia, la macro *elapsed* a été utilisée, (https://docs.julialang.org/en/v1/base/base/#Base.@elapsed).

Résultats & discussion

Les figures 4 et 4bis présentent les résultats du benchmark de Biopython et BioStructures. Tout d'abord, pour ce qui concerne le chargement des structures, les performances de python et Julia sont équivalentes (cd fig. 4A), i) pour python: de l'ordre du centième de seconde pour des structures de 10 kDa ou moins et de l'ordre du dixième de seconde pour des structures de 15 à 100 kDa, ii) pour Julia: de l'ordre du centième de seconde pour des structures de 35 kDa ou moins et de l'ordre du dixième de seconde pour des structures de 50 à 100 kDa. Globalement, la différence est donc minime. De plus, on remarque que les deux langages s'adaptent relativement bien à l'augmentation de la masse des structures.

Deuxièmement, concernant les alignements globaux de régions (cd fig. 4B), la performance de Julia est de l'ordre du millième de seconde, là où celle de python est plutôt de l'ordre du dixième de seconde pour des structures de 70 kDa ou moins et de l'ordre de la seconde pour des structures de 75 kDa ou plus. Globalement, même si Julia fait mieux, la différence de temps d'exécution reste dérisoire. En revanche, il semble que python s'adapte moins bien à l'augmentation de la masse des structures quand elle intervient.

Enfin, en ce qui concerne l'alignement de structures (cd fig. 4C), jusqu'à des structures de 50 kDa les performances de python et Julia sont identiques (de l'ordre du centième de seconde). Cependant, pour une structure de 55 kDa, on remarque une augmentation importante du temps d'exécution pour python, temps qui passe à l'ordre du dixième de seconde. Globalement, cela

reste un temps d'exécution très bon mais il est intéressant de noter que dans le cas des alignements de structures, python s'adapte moins bien à l'augmentation des données.

On peut également remarquer la présence d'artefacts pour le benchmark de Julia pour des protéines de 5 kDa dans le cas de l'alignement de régions et de structures. En effet, l'estimation de la médiane dans ces deux cas est plus élevée que pour des structures de 100 kDa ce qui n'est pas cohérent. De plus, les grandes barres indiquent une incertitude importante de de la médiane.

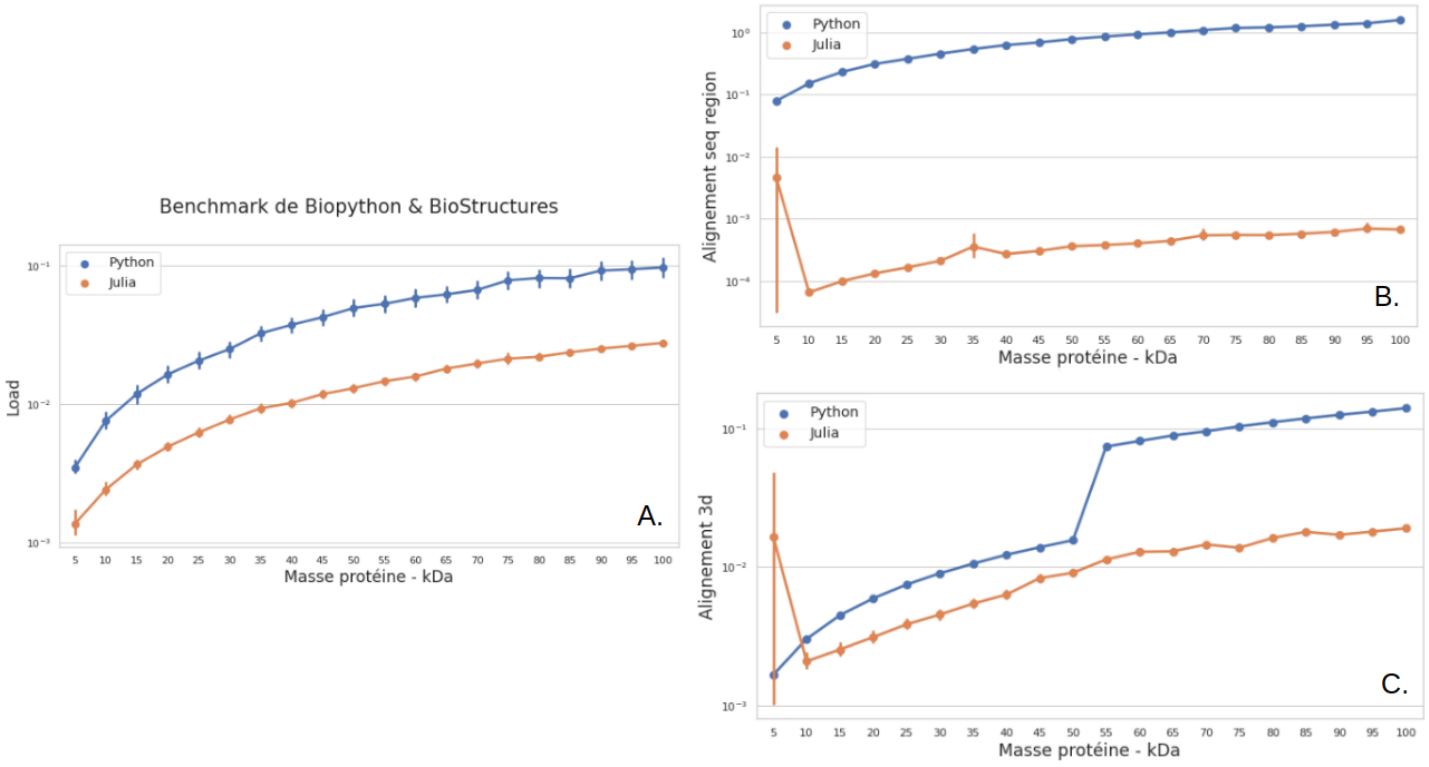

Figure 4: Benchmark de Biopython & BioStructures à partir du protocole expliqué dans la section Protocole de benchmark. Pour tous les graphiques, i) en bleu sont représentées les données obtenues avec Biopython et en orange celles obtenues avec BioStructures, ii) sur l'axe des abscisses est représentée la masse des protéines qui varie de 5 à 100 kDa, iii) sur l'axe des ordonnées est représenté le temps d'exécution en seconde en échelle logarithmique (logarithme base 10), iv) Les points représentent l'estimation de la médiane pour les 100 itérations d'une même méthode et les barres fournissent une indication de l'incertitude autour de cette médiane. **A**. Benchmark du chargement de structures. **B**. Benchmark de l'alignement global de régions (chaîne A de la structure 1 avec la A de la structure 2, chaîne B de la structure 1 avec la B de la structure B, et ainsi de suite). **C**. Benchmark de l'alignement de structures et calcul du RMSD.

Pour les deux méthodes suivantes (cf fig 4bis), soit les alignements globaux de séquences et les cartes de contacts, on remarque une différence importante de performance entre python et Julia. Différence d'autant plus grande, que l'axe des ordonnées est en échelle logarithmique. Pour Julia on remarque une bonne adaptation à l'augmentation de la masse des protéines, le temps d'exécution passant de l'ordre de 0.0001 seconde au dixième de seconde. Pour python, concernant les cartes de contacts (fig 4bis E), on remarque que jusqu'à une protéine de 45 kDa l'opération ne dépasse pas la seconde et qu'ensuite pour des protéines de 50 à 100 kDa le temps d'exécution de l'opération oscille entre 1 et 10 secondes. Cependant, dans le cas des alignements globaux, le temps d'exécution avec python augmente très rapidement, dépassant la seconde pour des protéines de 15 kDa, dépassant 10 secondes pour des protéines de 50 kDa ou plus, et ne cessant d'augmenter ensuite.

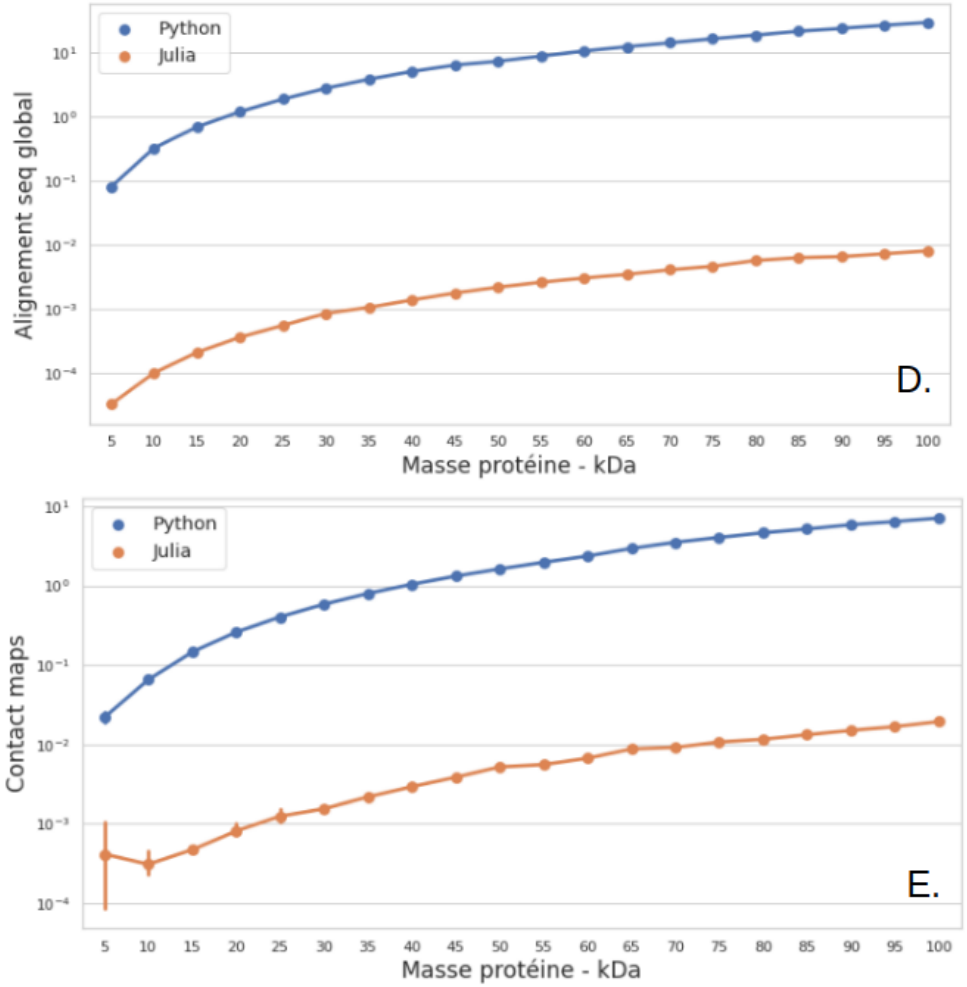

Figure 4bis: Suite de la figure 4. **D**. Benchmark de l'alignement global de séquences. **E**. Benchmark de la génération de cartes de contacts.

Conclusion et perspectives

Au vu des différents résultats, les différences de temps d'exécution entre Biopython et Biostructures sont minimes, essentiellement de l'ordre du centième ou du dixième de seconde. Ceci est valable pour le chargement des structures, l'alignement global des régions et l'alignement des structures. De plus, dans le cas des cartes de contacts, python reste adapté, malgré une moins bonne performance que Julia, si la masse de la protéine ne dépasse pas les 100 kDa. En revanche, dans le cas des alignements globaux de séquences, Julia est nettement plus performant que python, notamment dès que la masse de la protéine dépasse les 50 kDa. Ces résultats sont cohérents avec ceux du dossier github PDB Benchmark disponible à l'adresse suivante: https://github.com/jgreener64/pdb-benchmarks. En effet, comme observé dans les figures 4 et 4 bis, Julia est globalement plus rapide d'un centième voire dixième de seconde selon les méthodes évaluées.

On peut conclure que dans la majorité des cas Biopython est largement suffisant ; il faut quand même noter qu'il peut présenter des faiblesses par rapport à BioStructures lorsque la masse des protéines est supérieure à 50 kDa. Globalement, pour la majorité des tâches, Biopython est plus que suffisant. Comme précisé sur PDB Benchmark je conseillerais donc d'utiliser le langage/module avec lequel on est le plus familier.

L'analyse des modes normaux, non évoqué dans ce rapport, peut représenter une extension intéressante à ce projet. Ce genre d'analyses permet d'analyser les mouvements de biomolécules mais est relativement coûteux. Au vu des performances de Julia, il pourrait être intéressant de réaliser de telles analyses et de les évaluer par rapport au package R Bio3D par exemple (http://thegrantlab.org/bio3d_v2/index.php).

Bibliographie

1. Ivo Balbaert. Julia 1.0 Programming: Dynamic and high-performance programming to build fast scientific applications, 2nd Edition. Packt Publishing Ltd, 2018. ISBN: 1788990056, 9781788990059.
2. Hamelryck T, Manderick B. PDB file parser and structure class implemented in Python. Bioinformatics. 2003 Nov 22;19(17):2308-10. doi: 10.1093/bioinformatics/btg299. PMID: 14630660.
3. Joe G Greener, Joel Selvaraj, Ben J Ward, BioStructures.jl: read, write and manipulate macromolecular structures in Julia, Bioinformatics, Volume 36, Issue 14, 15 July 2020, Pages 4206–4207, https://doi.org/10.1093/bioinformatics/btaa502
4. J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.
5. Michael Waskom and the seaborn development team. mwaskom/seaborn. 2020 Sep. doi: 10.5281/zenodo.592845
6. Berman HM, Westbrook J, Feng Z, Gilliland G, Bhat TN, Weissig H, et al. The Protein Data Bank. Nucleic Acids Res. 2000 Jan 1;28(1):235–42.
7. Storz JF, Opazo JC, Hoffmann FG. Phylogenetic diversification of the globin gene superfamily in chordates. IUBMB Life. 2011;63(5):313-322. doi:10.1002/iub.482
8. Benson DA, Cavanaugh M, Clark K, Karsch-Mizrachi I, Lipman DJ, Ostell J, et al. GenBank. Nucleic Acids Res. 2013 Jan;41(Database issue):D36-42.
9. Fox NK, Brenner SE, Chandonia J-M. SCOPe: Structural Classification of Proteins—extended, integrating SCOP and ASTRAL data and classification of new structures. Nucleic Acids Res. 2014 Jan 1;42(D1):D304–9.