Compte Rendu de projet :

Calcul de la surface accessible au solvant d'une protéine

MARGERIT William : 71601721
PRETET Maël : 71604568
M2BI

Mercredi 16 septembre

Table des matières

1 Introduction

On cherche ici à implémenter une méthode permettant de calculer l'accessibilité d'une protéine. En effet, l'accessibilité est un descripteur important dans l'étude des protéines : on peut penser à son lors utilité lors de simulation de dynamique moléculaire, ou encore pour ce qui est de la représentation en 3D des protéines, sur des logiciels comme PyMOL. L'obtention de ce descripteur est aujourd'hui obtenue par des programmes informatiques tel que ASC[2] Naccess[1] getarea[3] (qui sera utilisé plus tard en tant que programme comparatif).

2 Méthodes

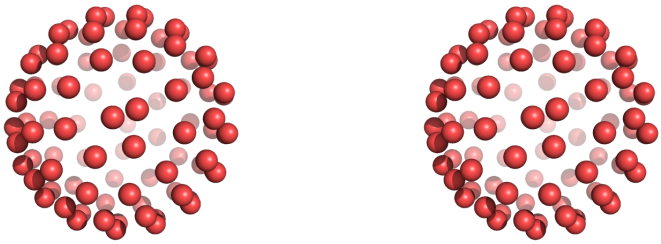
Une surface est dite inaccessible lorsque il n'y a pas assez d'espace autour d'elle pour accueillir une sonde. Cette sonde possède le rayon d'un atome d'oxygène, pour approximer celui d'une molécule d'eau (solvant utilisé le plus couramment). Ici, on va donc chercher à savoir si entre deux atomes, représentés par des sphères, on peut permettre le passage de cette sonde. Pour modéliser ces sphères, chaque atome de la protéine est résolu en 96 points positionnés de façon uniforme à une distance égale au rayon de Van der Walls de l'atome sommé au rayon de la sonde.

2.1 Sphère

Pour positionner de façon uniforme les 96 points sur la surface de la sphère, nous avons utilisé une implémentassions de l'algorithme de Saff et Kuijlaars[4]. Celui-ci permet de positionner N points sur la surface d'un sphère.

Vous pouvez observer la répartition des points en sphère sur les figures 1a, 1b.r

2.2 Accessibilité



(a) (b)

FIGURE 1

Chaque atome est donc résolue en 96 points représentant chacun $1/96^{eme}$ de la surface de la sphère. Pour évaluer la surface accessible d'un atome nous vérifions si chaque point composant sa sphère est ou non dans la sphère d'un autre atome.

Nous illustrons ces propos avec les schémas suivants. Comme on peut le voir sur la figure 2, sur la première situation aucun point de l'atome 1 (A1) n'est contenu dans la sphère de l'atome de l'atome 2 (A2) : les deux atomes ont donc 100% de leur surface accessible. En revanche dans la deuxième situation (figure 2) les points p1, p2 et p11 de l'atome A1 sont contenus dans la sphère de l'atome A2. De plus, les points p6 et p7 de l'atome A2, sont contenus dans la sphère de l'atome A1. Donc $3/11^{eme}$ des points de l'atome 1 sont inaccessibles et $2/11^{eme}$ des points de l'atome 2 sont occupés.

Vous pouvez observer que sur cet exemple que l'approximation de la surface en points présente un défaut de précision. On notera que d'après B.Lee et F.M.Richards, résoudre la surface de sphère d'accessibilité d'un atome ne présente plus de meilleur résultats à partir d'un seuil de 96 points, ce qui explique notre choix fait plus tôt.

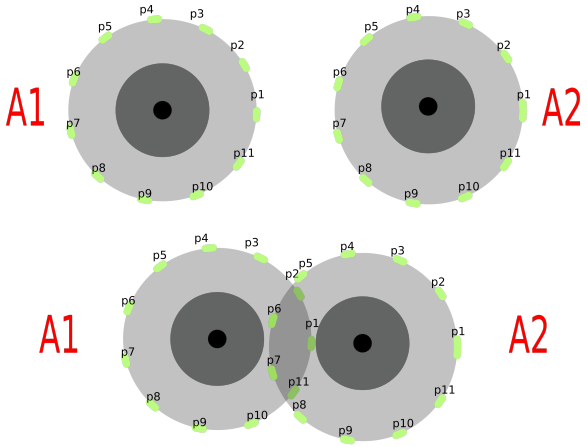


FIGURE 2 – Représentation 2D des sphères de contacte avec en noir le centre de la sphere

3 Résultats

Pour évaluer la qualité de notre modèle nous avons comparé nos résultats avec le programme [1] [3] qui calcule avec exactitude la surface de la protéine accessible au solvant. Pour cela, nous avons choisi de comparer les surfaces calculées totales des acides aminés à l'aide d'un test de comparaison de moyennes. Nous posons les hypothèses H0 : $m_1 = m_2$ et H1 : $m_1 \neq m_2$ au risque $\alpha = 5\%$. Les résultats sont les suivants :

Code pdb	p value
4yu3	0.27
1tjl	0.58
5t77	0.64

Au risque $\alpha = 5\%$, les trois p-values sont supérieures à 0.05. On peut donc observer que pour nos trois protéines tests, les surfaces calculés pour chaque acide aminé ne sont en moyenne pas significativement différentes entre notre programme et celles calculées par getarea. On peut donc juger que notre programme semble calculer efficacement les surfaces accessibles des acides aminés (et donc par extension de la protéine dans son ensemble).

4 Conclusion

En conclusion, nous pouvons dire que notre programme, en utilisant la technique de résolution de la surface d'accessibilité en 96 points de contact répartis de façon uniforme, nous obtenons des résultats similaires à l'algorithme getarea [3]. Ainsi la prochain étape de notre programme serait de convertir le code dans un langage de programmation plus performant, après avoir fini son optimisation.

Références

[1] B.Lee and F. Richards. The interpretation of protein structures : Estimation of static accessibility. *J. Mol Biol.*, 55 :379–400, 1971.

1. http ://curie.utmb.edu/area_man.html

[2] F. Eisenhaber and P. Argos. Improved strategy in analytic surface calculation for molecular systems : Handling of singularities and computational efficiency. *Journal of Computational Chemistry*, 14(11) :1272–1280, 1993.

[3] R. Fraczkiewicz and W. Braun. Exact and efficient analytical calculation of the accessible surface areas and their gradients for macromolecules. *Journal of Computational Chemistry*, 19(3) :319–333, 1998.

[4] E. B. Saff and A. Kuijlaars. Distributing many points on a sphere. *The Mathematical Intelligencer*, 19 :5–11, 1997.

5 Annexe

5.1 Logiciel utilisés

5.1.1 github

Pour synchroniser notre travail nous avons utilisé la plateforme github ainsi que git.

5.1.2 text-editor

Pour développer le logciel nous avons utilisé emacs et sublime text.

5.1.3 taiga.io

Pour se répartir les tâches nous avons utilisé l'application web taiga.io.

5.2 développement

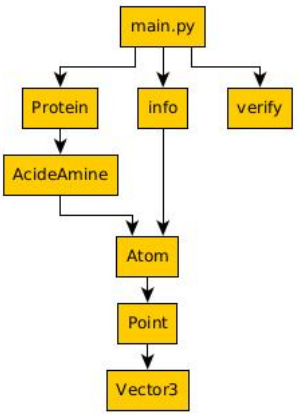


FIGURE 3 – Représentation de l'organisation des différente classes du code.

5.2.1 Optimisation

Pour ce projet nous avons voulu favoriser notre propre implémentation de codage plutôt que d'avoir recours à des bibliothèques python. Ainsi, nous n'avons pas profité de la rapidité de certaines bibliothèques (numpy). Au vu du temps imparti, certaines implémentations visant à augmenter la

vitesse d'exécution n'ont pas pu être terminées. Enfin, nous avons essayé d'implémenter du multi-threading pour accélérer le temps de calcul, mais ce dernier donnait le résultat inverse. L'implémentation du multi-threading aurait donc demandé une modification plus profonde de la structure du code.

Nous avons décidé de conserver la partie de code réservée au multi-threading, commentée, ainsi qu'une classe dédiée et non utilisée dans l'optique d'une amélioration possible.

5.3 Exemple d'utilisation du programme

Avec sortie des résultats dans un fichier texte (depuis le répertoire src) :

```
$ python main.py ../data/1tjl.pdb --file_out ../Results/out_1tjl_no_thr.txt
```

Avec sortie des résultats dans le terminal

```
$ python main.py ../data/1tjl.pdb
```