TEP Florian
N'étudiant : 22118207

M2 Biologie Informatique
Programmation et gestion de projets

Rapport : Assignation et détection des parties transmembranaires d'une protéine

INTRODUCTION :

Les protéines membranaires représentent environ un quart de la totalité des séquences protéiques issues des gènes humains et assurent des fonctions clés pour l'organisme, notamment dans la communication et le transport entre les cellules et le milieu extra-cellulaire. De ce fait, ces protéines représentent des cibles d'intérêts majeurs dans le développement de nouveaux médicaments qui cherchent à agir directement sur leur fonctionnement. Cependant, les structures des protéines membranaires sont encore mal résolues aujourd'hui, principalement car réaliser leur cristallisation en environnement aqueux représente un challenge technique important.

Les nouvelles technologies de biologie structurelle ont néanmoins permis d'apporter de nouvelles informations sur les coordonnées atomiques de nombreuses protéines membranaires. Toutefois, il reste très difficile de déterminer la position de la bicouche lipidique par rapport à la structure de ces protéines en conditions naturelles, car les possibilités d'obtenir des données expérimentales sur les membranes sont limitées. Pour reconstituer l'orientation des protéines au niveau des bicouches lipidiques, il faut donc se reporter à des méthodes prédictives proposées par des algorithmes dédiés tels que TMDET, ANVIL ou encore MEMEBED. L'utilisation de tels méthodes permet d'aboutir à une annotation des structures des protéines membranaires et de leurs segments transmembranaires, ce qui contribue à alimenter et mettre à jour les bases de données qui leurs sont consacrées, à l'image de PDB_TM ou OPM.

Dans ce contexte, l'objectif du travail présenté ici était d'établir un outil qui permettrait de déterminer les zones transmembranaires d'une protéine. Celui-ci est disponible à l'adresse : https://github.com/flotep/Projet_assignation_parties_transmb

La méthode proposée repose sur une approche géométrique pour suggérer la position, l'inclinaison, et l'épaisseur optimale de la membrane par rapport aux segments d'une protéine sur la seule base des coordonnées atomiques et de l'accessibilité au solvant de ses Calphas.

MATERIEL ET METHODES

Calcul de la surface accessible au solvant avec le programme DSSP

L'outil proposé traite les fichiers d'entrées en format PDB de la façon suivante. D'abord, il lit le fichier de la protéine et le passe au programme DSSP (*Define Secondary Structure of Proteins*)

pour calculer l'accessibilité au solvant de chacun de ces résidus à partir des structures secondaires qui sont assignées en fonction des coordonnées atomiques. Parmi ces résidus ne sont considérés que ceux pouvant interagir avec la bicouche lipidique, dont la valeur relative de la surface accessible (Relative ASA) est supérieure à un seuil défini à hauteur de 20% minimum.

Extractions des coordonnées des Calpha et calcul du centre de masse.

Les résidus de la protéine conservés sont ensuite divisés de manière binaire en deux catégories : hydrophobes pour les acides aminés {Phe, Met, Ile, Leu, Val, Trp, Ala, Cys, Gly, Ser, His}, et hydrophiles pour les acides aminés {Arg, Asp, Lys, Glu, Asn, Gln, Pro, Thr, Tyr}. Pour chaque résidu, seules les coordonnées atomiques des Calphas sont récupérées à partir du fichier PDB d'origine. Le centre de masse de la protéine est alors calculé à partir de ces Calphas et leurs coordonnées sont transformées par rapport à ce centre qui correspond donc au point (0,0,0) de notre repère.

Détermination de vecteur normaux pour orienter la membrane.

A partir de ce centre de masse est ensuite générée une surface sphérique sur laquelle sont échantillonnés des dizaines de points de manière aléatoire. Chaque point permet de créer un vecteur normal qui définit une position initiale et une direction que va prendre le plan de la membrane, orthogonal au vecteur selon l'équation : ax + by +cz +d =0. L'objectif étant de quadriller avec suffisamment de précision toutes les directions possibles.

Par la suite, la position du plan de la membrane va être déplacée par tranche de 2 Angstrom le long de chaque vecteur normal tant que le nombre de résidus hydrophobes présent dans le plan est supérieur à 0. Dans le cas contraire, on estime que le plan que l'on regarde est situé en dehors de la protéine.

Scoring pour trouver les meilleures positions de la membrane.

Le but de cette approche est de calculer grâce à la fonction objective, pour chaque position de la membrane, le rapport entre le nombre d'atomes Calphas issus de résidus hydrophobes (M) à l'intérieur (Mi) et à l'extérieur (Me) du plan, par rapport au nombre d'atomes Calphas issus de résidus hydrophiles (S) à l'intérieur (Si) et à l'extérieur (Se) de ce même plan, selon la formule présentée dans *Postic et al , 2015* :

$$C = \frac{M_i \times S_e - S_i \times M_e}{\sqrt{(M_i + S_i) \times (M_i + M_e) \times (S_i + S_e) \times (S_e + M_e)}}$$

Les acides aminés M sont prédits comme étant interne à la membrane tandis que les acides aminés S sont attendus à l'extérieur de cette membrane. La formule utilisée comme méthode de scoring dans cet algorithme cherche donc à trouver le plan de bicouche lipidique qui maximise la valeur de C.

Epaississement des meilleures positions de la membrane pour trouver le plan optimal.

Une fois que les scores de tous les plans possibles de la membrane ont été calculées, on conserve la position qui maximise C dans chaque direction. Pour chacune de ces positions, l'épaisseur de la membrane (initialement égale à 14 Angstrom) est augmentée par tranche de 1 Angstrom tant que la valeur de C augmente.

La position avec l'épaisseur de membrane pour laquelle le score C est le plus élevé donnera donc le plan optimal de la bicouche lipidique pour la protéine étudiée, l'équation du plan de chaque couche pouvant être récupérée en donnée de sortie.

Visualisation de la membrane avec la structure de la protéine.

Les sorties de cet algorithme permettent également une visualisation directe du plan optimal de la membrane par rapport à la structure de la protéine. D'abord, les coordonnées de tous les atomes du fichier PDB d'origine sont transformées par rapport au centre de masse précédemment calculé. Ensuite, ce fichier est édité en ajoutant des hétéroatomes dont les coordonnées ont été générées sur chaque plan de la bicouche lipidique de manière aléatoire. Ainsi, deux « tapis » de molécules sont ajoutés à la structure de la protéine et permettent d'identifier la position de la membrane en ouvrant le nouveau fichier PDB avec un logiciel de visualisation tel que PyMOL.

RESULTATS ET DISCUSSION :

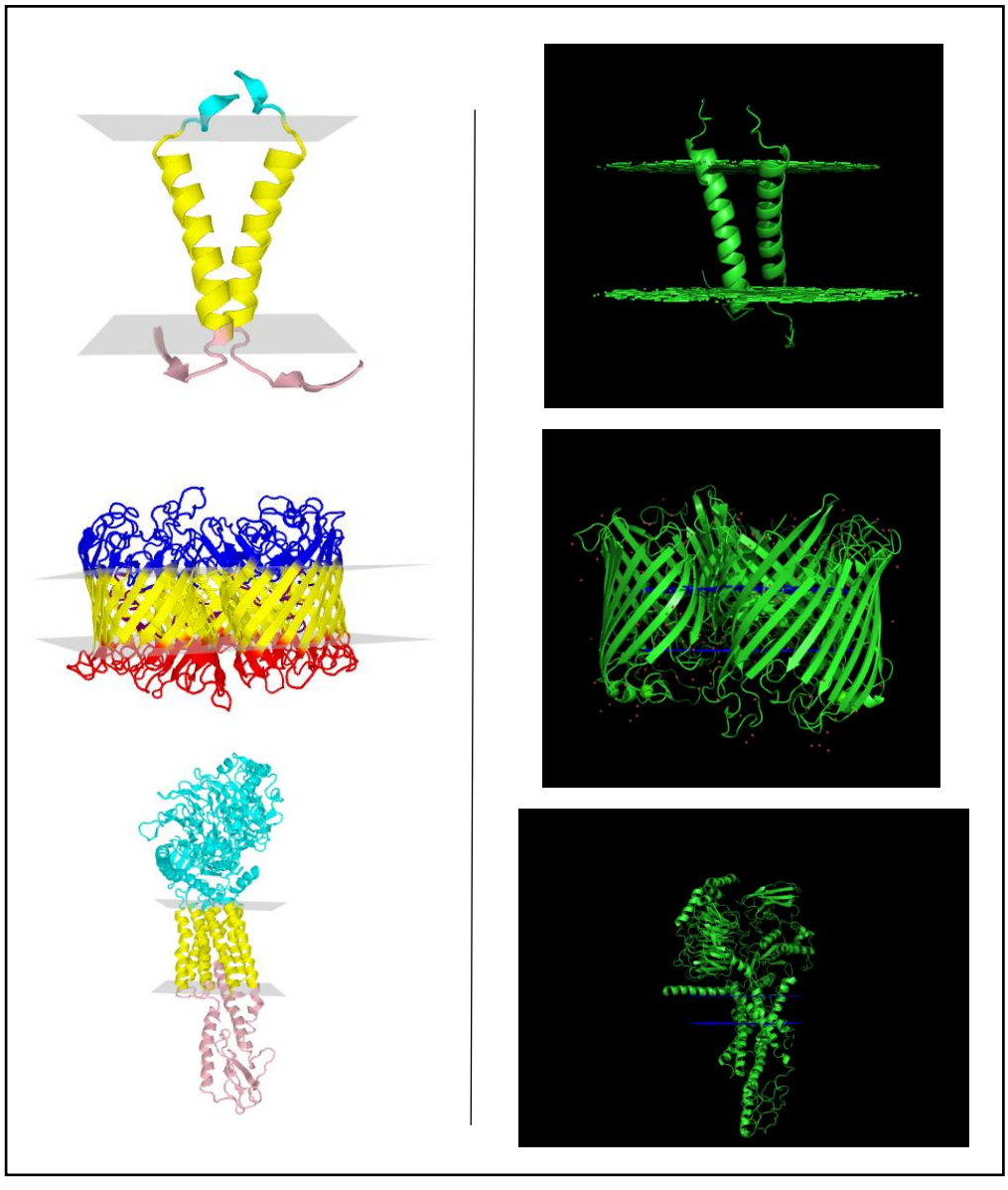

Figure. Localisation de la membrane dans la base de données PDB_TM (gauche) comparé à la localisation obtenue avec l'outil développé pendant ce projet (droite). ID des protéines de haut en bas : 2n90, 1a0s et 6whc. Les plans de la membrane sont représentés en bleues pour 1a0s et 6whc sur les images de droite.

L'outil proposé permet dans certains cas de déterminer la position de la bicouche lipidique par rapport à la structure des protéines transmembranaires avec une précision relativement proche de celle indiquée dans la base de données PDB_TM. Cette méthode se révèle performante pour l'étude des protéines dont la résolution est suffisamment haute, sans présence de chaine discontinue ou de structure complexe tel que des cavités. L'unique utilisation des coordonnées atomiques des Calphas semble être une approche efficace pour obtenir une approximation satisfaisante.

L'implémentation de la fonction objective, relativement flexible, permet également de tester aisément de nouveau système de scoring pouvant être présenté dans la littérature afin de comparer les résultats obtenus.

La représentation de la membrane par l'ajout d'atomes dans le fichier PDB édité est acceptable pour constater sa position par rapport à la protéine, mais nécessite préférablement l'utilisation d'un logiciel de visualisation permettant de colorer chacun des plans (PyMOL suggéré).

La rapidité d'exécution du programme peut représenter un frein dans son utilisation, et dépend principalement du nombre de direction quadrillée dans l'espace par la génération des points sur la sphère. Cette durée est également susceptible de s'allonger lorsque la structure des protéines se complexifie.

En optimisant le temps d'exécution de cet outil, celui-ci pourrait représenter une première approche fiable pour prédire la topologie de la bicouche lipidique dans la structure des protéines transmembranaires. Néanmoins, l'algorithme ne permet pas de différencier les protéines membranaires et globulaires, il serait donc nécessaire de savoir au préalable si la protéine comporte des régions transmembranaires.

ANNEXE

```python
#Argument Parser pour récupérer les arguments entrés par l'utilisateur.
argparser = argparse.ArgumentParser(description="Extract chain from a PDB file")
argparser.add_argument('infile', help="Path to input file (PDB)")
argparser.add_argument('outfile', help="Path to input file (PDB)")
argparser.add_argument('--model',type =int, default=1, help=
"Modèle de la protéine étudié")
argparser.add_argument('--vector_nb',type =int, default=30, help=
"Nombre de vecteurs échantillonnés aléatoirement")
args= argparser.parse_args()

PDBFile=args.infile
OutputFile=args.outfile
PDBName=os.path.basename(args.infile)
PDBid=PDBName.split(".")[0]

#Lis le fichier PDB et le passe au programme DSSP, contient la valeur relative de la surface accessible (ASA) de chaque CA
p = PDBParser()
structure = p.get_structure(PDBid, PDBFile)
model_protein = structure[args.model-1]
dssp = DSSP(model_protein, PDBFile, dssp='mkdssp')

# Créer un dictionnaire DSSP à partir du fichier PDB, contient les Residus IDs de chaque CA.
dssp_tuple=dssp_dict_from_pdb_file(PDBFile)
dssp_dict = dssp_tuple[1]
id_tuple = [x[1] for x in dssp_dict]
id=[y[1] for y in id_tuple] #Récupère les Résidus IDs dans une liste


df_DSSP = pd.DataFrame(dssp) #Créer un DataFrame à partir du fichier DSSP de départ.
df_column=[0,1,3]
df_DSSP = df_DSSP.iloc[:,df_column]
#Garde seulement les IDs par défaut, le nom des acides aminés et les valeurs de relative ASA.
df_DSSP.columns=['#','amino_acid','Relative_ASA']
df_DSSP['#']=id
#Remplace les IDs par défaut du fichier DSSP par les Residus IDs stockées dans le dictionnaire DSSP.
#print (df.loc[[6]])
print(df_DSSP)

threshold = 0.20
#Seuil minimum de la relative ASA pour considérer qu'un résidu est accessible au solvant.
residue_hydrophobic=['F','G','I','L','M','V','W','A','C','S','H']
#Classification binaire des résidus selon Postic et al., 2015
residue_hydrophilic=['D','E','K','N','P','Q','R']


#Conserve seulement les CA si leur résidu est accessible au solvant et classe les hydrophobes et les hydrophiles dans deux listes séparées
accessible_ca = df_DSSP.loc[(df_DSSP['Relative_ASA']>threshold)]
accessible_ca_hydrophobic = accessible_ca.loc[accessible_ca['amino_acid'].str.contains('|'.join(residue_hydrophobic))]
accessible_ca_hydrophilic = accessible_ca.loc[accessible_ca['amino_acid'].str.contains('|'.join(residue_hydrophilic))]

#Récupère les résidus IDs des CA accessibles au solvant pour les résidus hydrophobes et hydrophiles.
accessible_residue_hydrophobic= accessible_ca_hydrophobic['#'].tolist()
accessible_residue_hydrophilic= accessible_ca_hydrophilic['#'].tolist()



#Lecture du fichier PDB en entrée pour récupérer les coordonnées de tous les CA accessibles au solvant.
#Ces coordonnées ne sont pas accessible depuis le fichier DSSP ou le dictionnaire DSSP.
io = PDB.PDBIO
struct = p.get_structure(PDBid,PDBFile)
model = structure[args.model-1]
#Ne parcours que le modèle de la protéine souhaité, par défaut le 1er modèle.

#Liste les coordonnées x,y,z des CA des résidus hydrophobes et hydrophiles.
ca_hydrophobic_coord_list = []
ca_hydrophilic_coord_list = []

for chain in model:
    for residue in chain:
        res_id=residue.get_full_id()[3][1]
#On ne récupère que le ID du résidu parmi tous les IDs obtenus avec le get_full_id
        if res_id in accessible_residue_hydrophobic :
            ca=residue["CA"]
            ca_hydrophobic_coord=ca.get_coord()
#Récupère uniquement les coordonnées des CA.
            ca_hydrophobic_coord_list.append(ca_hydrophobic_coord)

        elif res_id in accessible_residue_hydrophilic :
            ca=residue["CA"]
            ca_hydrophilic_coord=ca.get_coord()
            ca_hydrophilic_coord_list.append(ca_hydrophilic_coord)

#Calcul le centre de masse de la protéine avec les coordonnées de tous les CA des résidus accessibles au solvant.
ca_coord_list = np.array(ca_hydrophilic_coord_list + ca_hydrophobic_coord_list)
com=center_of_mass(ca_coord_list)


#Transforme les coordonnées de tous les CA par rapport aux coordonnées du centre de masse (qui représentera le point (0,0,0))
transformed_coord_hydrophilic=transform_coordinates(ca_hydrophilic_coord_list,com)
transformed_coord_hydrophobic=transform_coordinates(ca_hydrophobic_coord_list,com)

#Détermine les vecteurs (a,b,c) à partir de l'échantillonnage de points sur sphère.
vecteurs=sample_spherical(args.vector_nb,3)
#Création d'un plan orthogonal au vecteur passant par le point (0,0,0).
orthogonal_planes=get_orthogonal_planes(vecteurs,x,y,z,d)
```

```python
optimal_position = []
optimal_score_direction =[]

#Boucle "for" pour trouver le meilleur score obtenu dans chaque direction, et à quelle position il a été obtenu.
for i in orthogonal_planes :
    list_score=[]
    position=0 #The initial position is always 0

    #Donne les équations des deux plans de la membrane à la position 0.
    membrane1, membrane2 = get_membrane_planes(i)


    #Calcul le score obtenu avec la membrane à la position 0.
    score, nb_hydrophobic_atom_inside=objective_function(membrane1, membrane2, transformed_coord_hydrophilic, transformed_coord_hydrophobic)
    list_score.append(score)
    #print (atoms inside)


#Déplace la membrane dans la direction du vecteur par tranche de 2 A tant que le nombre de CA hydrophobes entre les deux plans > 0.
    while nb_hydrophobic_atom_inside > 0:
        position += 2

        #Donne les équations des plans de la membrane à la position donnée.
        membrane1, membrane2 = get_membrane_planes(i,pos=position)

        #Calcul le score obtenu avec cette membrane à la position donnée.
        score, nb_hydrophobic_atom_inside=objective_function(membrane1, membrane2, transformed_coord_hydrophilic, transformed_coord_hydrophobic)
        list_score.append(score)


#Trouve le score maximum obtenu pour une membrane dans cette direction, et à quel index de la liste il se trouve.
    max_score_direction=max(list_score)
    idx_max_score = list_score.index(max_score_direction)


    #Récupère le score maximal et l'index associé pour chaque direction.
    optimal_position.append(idx_max_score)
    optimal score direction.append(max_score_direction)

#print (optimal position)

#Liste les équations des plans optimales après épaississement dans chaque direction, et leur score associé.
optimal membrane1 list=[]
optimal_membrane2_list=[]
optimal score=[]

#Boucle "for" pour trouver le meilleur score obtenu après épaississement à la position optimale de chaque direction.
for plan,idx position,score in zip(orthogonal planes,optimal position, optimal_score_direction)  :
#for each membrane with the highest number of atoms, we increase their thickness 1 A by 1 A as long as the score increase
    add to pos=2 #correspond to the size of the pas during the sliding of the membrane
    thickness = 14
    max_score=score

    list mb1=[]
    list_mb2=[]
    list_score=[]


    thickness +=1 #Incrémentation de l'épaisseur de la membrane de 1 A par 1 A.
    position = 0 + add to pos*idx position
#Retrouve la position à laquelle la meilleur membrane a été trouvée dans cette direction.



#Calcul les nouvelles équations et le score des plans de la meilleur position après augmentation de son épaisseur.
    membrane1, membrane2 = get_membrane_planes(plan, thickness= thickness,pos= position)
    score,nb_hydrophobic_atom_inside=objective_function(membrane1, membrane2, transformed_coord_hydrophilic, transformed_coord_hydrophobic)

    list_mb1.append(membrane1)
    list_mb2.append(membrane2)
    list_score.append(score)



#Si le score de la membrane après épaississement est supérieur au score précédent, continue d'épaissir la membrane tant que c'est le cas.
    if score>=max_score:
        while score >= max score :
            max_score=score
            thickness += 1
            membrane1, membrane2 = get_membrane_planes(plan, thickness=thickness,pos=position)
            new_score,nb_hydrophobic_atom_inside=objective_function(membrane1, membrane2, transformed coord hydrophilic, transformed coord hydrophobic)

            list mb1.append(membrane1)
            list_mb2.append(membrane2)
            list score.append(new score)

            if new score >= score :
                max_score = score
                score = new score


#Si le nouveau score est inférieur au précédent, récupère les équations et le score de la membrane précédente.
            else :
                membrane1, membrane2 = get membrane planes(plan, thickness=thickness,pos= position)
#if the score doesnt increase for the new membrane, get the equation of the previous membrane + its score
                optimal_membrane1 list.append(list_mb1[-2])
                optimal_membrane2_list.append(list_mb2[-2])
                optimal_score.append(list_score[-2])
                break


#Si le premier épaississement n'a pas amélioré le score, garde les équations et le score obtenu sans épaississement.
    else :
        membrane1, membrane2 = get_membrane_planes(plan,pos= position)
#keep the original thickness if the incremation doesn't increase the score from the beginning
        score, nb_hydrophobic_atom_inside=objective_function(membrane1, membrane2, transformed_coord_hydrophilic, transformed_coord_hydrophobic)
        optimal_membrane1_list.append(membrane1)
        optimal membrane2 list.append(membrane2)
        optimal_score.append(score)


#Trouve le score maximal et l'index associé obtenus après épaississement de toutes les membranes à la meilleure position de chaque direction.
max score membrane=max(optimal score)
idx_score_optimal = optimal_score.index(max_score_membrane)


#Retourne les équations des plans de la membrane optimale pour la protéine.
optimal_membrane1=optimal_membrane1_list[idx_score_optimal]
optimal membrane2=optimal membrane2 list[idx score optimal]
print (optimal_membrane1, optimal_membrane2)
```

```
#Récupération de chaque argument au sein des équations.
#Doit utiliser findall car les équations sont composées de symbole sympy, l'ordre des arguments retournés par .arg[] change entre les équations.
str_mb1_equation=str(optimal_membrane1)
str_mb2_equation=str(optimal_membrane2)
str_mb1_parameters= reg.findall(r"(?:\+\s*|\-\s*)?[-+]?\d*\.*\d+", str_mb1_equation)
str_mb2_parameters= reg.findall(r"(?:\+\s*|\-\s*)?[-+]?\d*\.*\d+", str_mb2_equation)
mb1_parameters=[float(str(i).replace(' ','')) for i in str_mb1_parameters]
mb2_parameters=[float(str(i).replace(' ','')) for i in str_mb2_parameters]


a,b,c=mb1_parameters[0],mb1_parameters[1],mb1_parameters[2]
d1=mb1_parameters[3]
d2=mb2_parameters[3]

normal_vec=(a,b,c)

#Génération de points (x,y,z) random situés dans le plan de la 1ere couche de la membrane.
r=d1/c
refpoint=(0.0,0.0,r)
plan1=Plane(normal_vec,d1,refpoint,30)
random_plane_points = plan1.create_random_points(1000)
#https://pypi.org/project/random-geometry-points/

#Génération de points random (x,y,z) situés dans le plan de la 2ere couche de la membrane.
r2=d2/c
refpoint2=(0.0,0.0,r2)
plan2=Plane(normal_vec,d2,refpoint2,30)
random_plane_points_2 = plan2.create_random_points(1000)

#Lecture du fichier PDB en entrée puis transformation des coordonnées de chaque atome par rapport au centre de masse.
parser = PDB.PDBParser()
io = PDB.PDBIO()
struct_transform = parser.get_structure(PDBid,PDBFile)
model_transform =struct_transform[args.model -1]
rotation_matrix = PDB.rotmat(PDB.Vector([0, 0, 0]), PDB.Vector([0, 0, 0]))
#Matrice neutre, pas de rotation des coordonnées souhaitée.
transform=(-com[0],-com[1], -com[2])

for chain in model_transform:
    for residue in chain:
        for atom in residue:
            atom.transform(rotation_matrix, transform)

#Retourne un fichier PDB avec les coordonnées de tous les atomes transformées.
io.set_structure(model_transform)
PDBTransform= os.path.join(OutputFile,'{0}_model{1}_transform.pdb'.format(PDBid, args.model))
io.save(PDBTransform)


#Lis le fichier contenant les nouvelles coordonnées avec le module Biotite.
atom_array = strucio.load_structure(PDBTransform)

#Pour chaque point généré aléatoirement sur la 1ere couche de la membrane , créé un hétéroatome possédant ses coordonnées.
for i in random_plane_points :
    atom = struc.Atom(
        coord = [i[0],i[1],i[2]],
        chain_id = "N",
        res_id = atom_array.res_id[-1] + 1,
#Le résidu id de l'atome est égale au dernier résidu ID du fichier +1
        res_name = "MB1",
#Chaque atome créée pour la 1ère couche possède le nom de résidu MB1.
        hetero = True,
        atom_name = "CA",
        element = "C"
    )
    atom_array += struc.array([atom])
#Ajout de l'hétéroatome à la suite des atomes présent dans le fichier PDB.

#Pour chaque point généré aléatoirement sur la 2eme couche de la membrane, créé un hétéro atome possédant ses coordonnées.
for j in random_plane_points_2 :
    atom = struc.Atom(
        coord = [j[0],j[1],j[2]],
        chain_id = "N",
        res_id = atom_array.res_id[-1] + 1,
        res_name = "MB2",
#Chaque atome créée pour la 1ère couche possède le nom de résidu MB2.
        hetero = True,
        atom_name = "CA",
        element = "C"
    )
    atom_array += struc.array([atom])
#Ajout de l'hétéroatome à la suite des atomes présent dans le fichier PDB.

#Retourne un fichier PDB édité avec les hétéroatomes représentant chaque couche de la membrane ajoutés à la structure de la protéine.
PDBEdited=os.path.join(OutputFile,'{0}_model{1}_edited.pdb'.format(PDBid,args.model))
strucio.save_structure(PDBEdited, atom_array)
```