Molecular Simulations Project 1: Replicate OPM. Setting transmembrane proteins in a predicted membrane.

Adrià Pérez and Alberto Meseguer

30 April, 2016

1 Tutorial

The program we have developed is named membranator.py. Membranator.py takes as input the .cif and .pdb files of the same protein and returns a .pdb file containing the same protein placed between two slabs of dummy atoms, representing the edges of a lipidic membrane. For its execution you must be on a directory containing the script. The script is written in python 3.5.1, here is a list of the necessary packages for running it properly:

- htmd
- biopython
- numpy

Here is an example of the usage of the program:

python membranator.py -i structures/ -o output -b 0

The arguments that this program takes are the following:

- -i(input): is the only compulsory argument for running the program. It must be a path to a directory which contains the two input files: the .pdb and the .cif one. Its important that the two files have the same name and only differ on the termination. If the directory contains this pair of files for more than one protein, the program will return an output for all structures having the two input files.
- -o(output): is the name of the output .pdb file. By default the file will be named output.pdb, so this argument is not mandatory.
- -b(beta sheet mode): if the protein has a beta sheet transmembrane domain, you can choose between different methods for performing its orientation. Each method is represented by a number:

  0 : $get\_sheet\_vector\_CA\_CA\_version$

  1 : $get\_sheet\_vector\_consecutive\_CA\_version$

  2 : $get\_sheet\_vector\_inertia\_tensor\_version$

  3 : $get\_sheet\_vector\_ring\_version$

  Each one of this methods is explained in the following section, on the division of "Obtaining the perpendicular vector to the membrane". By default, the program runs with the first method, the one corresponding to the 0 number. So this argument is not mandatory.

2 Explanation of the program

Once we introduce the proper input to the program, let's see how does it work. The program is subdivided in functions, so we will explain it function by function following the logical order within the program.

The first function which is called in the program is *Membrane_builder*, which will be the main function of the program. This will call most of the other functions of the program. The first thing this function does is creating a molecule object from the .pdb file. Then, it calls the function *get_secondary_structure*.

Getting the secondary structure of the protein: The objective of the *get_secondary_structure* function is to return a list of lists, in which each of these lists contains information about the transmembrane domains of the protein. This information is always organized in three fields in the following order: the chain to which the transmembrane domain belongs, the position of the sequence in which the domain starts and the position in the sequence in which the domain ends. For obtaining this information, we extract positions corresponding to secondary structures using the Bio.PDB.MMCIF2Dict package for python. This information can only be provided by .cif files, that is why we need having both the .pdb and the .cif files for running the program properly. Then, helices or sheets are considered transmembrane if their length is above 14 residues in helices and 8 residues in sheets. Finally, the information about these domains is stored.

Then, we have to determine whether the protein has an alpha helical or a beta sheet transmembrane domains. For doing this we compare the information about alpha helices and beta sheets. If one protein has no alpha helices we know for sure that the transmembrane domain will be a beta sheet, and vice versa. If the protein has both helices and sheets, we compute an hidrophobicity score for the two sets of domains. This score is computed obtaining the mean for the hydrophobicity value for each residue belonging to these secondary structures. The hydrophobic value for each residue is obtained from the amino acid hydrophobicity scale, defined in the paper "Experimentally determined hydrophobicity scale for proteins membrane interfaces"; written by Wimley and White(Nat Struct Biol; 3:842; 1996). As the computed score is a mean depending on the number of residues, it is not biased by the number of amino acids or by the number of secondary structure regions. Therefore, the structure type with higher hydrophobicity score is selected to be the one constituting the transmembrane domain.

Obtaining the perpendicular vector to the membrane: Once we have classified our protein depending on the structure of its transmembrane domain, we have to get the vector which defines the perpendicular direction to the membrane. Then, we will place the protein in the center of coordinates and rotate it, so the perpendicular vector to the membrane gets aligned with the Z-axis of the 3D space. On this process, the first step is to find this perpendicular vector to the membrane, this is carried by different funtions according to whether the protein has an helical or sheet transmembrane domain.

For alpha helices we obtain the vector by the *get_helix_vector* function. Here, the pro-

gram computes, for each alpha helix, the mean of all the vectors that go from the beta carbon to the oxygen within the protein backbone. Then, if the protein has several transmembrane helices we obtain the mean vector between them. It is important to remark that we are computing these vectors following the order of the protein sequence in an N-terminal C-terminal direction. This means that if we have two helices united by a loop, the resulting vectors of these two helices will point in opposite directions. That is why we assign an incrementing integer value to each transmembrane domain and we multiply by -1 all pair helix vectors, so vectors from all helices point in the same direction. This approach has a weakness, and it is that if we have in our structure an helix which is not transmembrane and it has been identifyed as that, then it interferes in the alternance of opposite helices leading to unaccurate results.

For beta sheets we have developed several methods which may be complementary. The idea is that the user can select the method for obtaining this vector by introducing an argument in the program command line. For more information look at the tutorial section. These are the different developed methods, each one corresponding to one of these functions:

1 *get_sheet_vector_CA_CA_version*. This function computes for each transmembrane beta sheet the vector between the two third residues of the sheet, starting for each one of its ends. Here we also find that if we compute these vectors depending on sequence position, vectors from contiguous sheets will point in opposite directions. For that we compute the opposite vector for vectors corresponding to a pair number of beta sheet, just as we did in the previous function. Then, all vectors are normalized and a mean is obtained from them, so the length of the sheets doesn't influence on the outcoming vector.

2 *get_sheet_vector_consecutive_CA_version*. This function computes for each transmembrane beta sheet the mean vector between all the vectors that go from one alpha carbon to the alpha carbon of the next amino acid in the sequence. This is only performed for residues which are 3 positions far from the ends of the sheet. Again, we compute inverse vectors corresponding to pair transmembrane sheets and we normalize them before computing the mean, which will be the outcoming vector.

3 *get_sheet_vector_inertia_tensor_version*. This function takes profit of the physic properties of rigid bodies. First of all, the program defines the I matrix. This matrix is the inertia tensor for a rigid body, which is containing the information for the mass distribution of this body. Here we can see this matrix

$$I = \begin{pmatrix} \sum_i m_i(y_i^2 + z_i^2) & -\sum_i m_i x_i y_i & -\sum_i m_i x_i z_i \\ -\sum_i m_i y_i x_i & \sum_i m_i(x_i^2 + z_i^2) & -\sum_i m_i y_i z_i \\ -\sum_i m_i x_i z_i & -\sum_i m_i y_i z_i & \sum_i m_i(x_i^2 + y_i^2) \end{pmatrix}$$

We fill this matrix with the coordinates of alpha carbons belonging to transmembrane domains, we asume a constant mass for all of them. For beta barrels this mass distribution is pretty similar to the one of a cylinder. Therefore, for cylnders it is known that the diagonalization of this matrix results in three eigenvalues, where one is significantly lower than the other two. The eigenvector associated to this eigenvalue

defines the direction in space corresponding the height of the cylinder. Considering the cylinder a beta barrel, this vector is the perpendicular one to the membrane. That is the computation performed by this function.

4 *get_sheet_vector_ring_version*. This function is computing the mean of the perpendicular vector to the aromatic rings placed in the transmembrane domains of the protein. For each transmembrane domain the aromatic residues (PHE, TYR, TRP and HIS) are identified. Then, a perpendicular vector to the ring is obtained by computing the vector product between two vectors obtained from three points (atoms) from the ring. As long as rings are flat structures this operation returns the perpendicular vector to the ring. Finally, we compute the mean of all these vectors and we consider this the perpendicular vector to the membrane.

From this whole set of methods, the ones showing a better performance are the two first. Actually, the default method of the program for computing this vector is the first function.

Rotating the protein: Now that we have the vector it is time to place the protein on the center of coordinates and to rotate it, so the obtained vector gets aligned with the Z-axis. This computation it is performed by the *rotate_prot* function.

In order to achieve that, we use spherical coordinates to obtain an angle to rotate the protein by the htmd method **rotate(vector,angle)**. We use this formula to calculate it:

$\theta = arccos(\frac{z}{\sqrt{x^2+y^2+z^2}})$

Then, we calculate the perpendicular vector between our protein vector and the z axis, and we use the method previously mentioned to rotate the protein, with the theta angle and the perpendicular vector calculated

Placing the membrane: Once the protein is in the center of coordinates and oriented with the Z-axis, we will place two layers of dummy atoms as a membrane. For doing this we will evaluate the hidrophobicity of the residues at different levels of the Z-axis along the 3D space. So, we select all residues whose alpha carbon is found within certain intervals according to the Z-coordinate value, and for each interval the number of hidrophobic residues is computed. Then, the membrane is placed on the regions showing a higher density of hidrophobic residues. This action is carried by the functions *hydrophobicity_calculator* and $dummy\_leaflet$.

Finally, the resulting structure is saved in the directory where the program was executed.

3 Limitations and improvements

- **Defining more acurately the transmembrane domains:** By now, we are only defining transmembrane domains reckoning on their length. This may work for some structures, mainly those which mostly comprise the transmembrane domain of the

protein. Buf for bigger structures, which contain globular and transmembrane domains, this approach is prone to fail. This issue could be tackled by incorporating to our program a function for predicting which domains are transmembrane, based on amino acid properties such as hydrophobicity.

- **Handling more preciselly the means between vectors corresponding to transmembrane domains:** We have adressed the problem of contiguous transmembrane domains with oposite directions by asigning numbers to the domains and computing the oposite vector for pair domains. This approach may fail easily. For example, if we consider a globular structure as transmembrane, this would disturb the correspondence of numbers with transmembrane domains. Thus, the program would obtain the opposite vectors for the unapropiate domains. This could be improoved by analyzing the angles between the different vector and obtaining the opposite vectors depending on the angle between them, instead of doing this only by an index.

- **Beta sheet orientating functions:** The inertia tensor and the aromatic ring approach are prone to be unaccurate. In the case of the inertia tensor procedure, for working properly depends on the structure of the protein to be an exact cylinder, something that scarcelly happens in nature. Then, most of the times, in stead of computing a matrix corresponding to a cylinder, the matrix is irregular and represents the mass distribution of an elypsoid. Therefore, the whole approach fails. Another limitation is that only works for cylinders which have a larger height than diameter, so for beta barrels with huge porus this method becomes less effective. This method could be improoved by considering more atoms, not only the alpha carbons, for defining the mass distribution of the body. Using the whole backbone would be a suitable possibility. This would help to define with more accuracy the mass distribution of the body, which is the main property utilized on this function.

  In the case of the aromatic ring procedure, we haven't been able to find a regular pattern in transmembrane proteins regarding aromatic ring orientation. More research and work will be needed for improving these function.

- **Improving the placement of the membrane:** The employed method for placing the membrane has not very good accuracy. That may happen because we are just taking in account information regarding the hidrophobic properties of individual residues in order to predict which regions are in contact with the membrane. This could be improved by incorporating more complex statistic models based on sequence that discriminate transmembrane from non transmembrane sequences. These models could be a naive bayes classifier or a markov model, which have already been used previously for identifying transmembrane domains.