PySeg tutorial: unoriented membranes

Antonio Martinez-Sanchez[1,*]

23.10.2019

[1]Department of Molecular Structural Biology, Max Planck Institute for Biochemistry, Am Klopferespitz 18, 82152 Martinsried, Germany

[*]E-mail: martinez@biochem.mpg.de

1 Introduction

This tutorial gives an example that shows the most of PySeg functionalities for processing unoriented membranes, here a script is provided to generate synthetically the input data. A membrane is unoriented, in contrast to oriented, if we don't know what are its inner and outer sides (see Fig X). We use to have unoriented membranes when cryo-ET tomograms has been processed with automated methods for membrane segmentation, like TomoSemMemTV. In general, to orient a membrane is not at all a trivial task in cryo-ET because of the presence of holes in membranes. To orient a membrane, that is to segment with different labels the inner volume and the outer volume, requires of a further, currently manually supervised, processing.

This tutorial has any scientific interest beyond the training of PySeg users. It is strongly recommended not to start the analysis of real experimental data with PySeg before completing successfully this tutorial. Besides having PySeg system installed, this tutorial also requires TomoSegMemTV (included in PySeg system package), Relion and TOM Toolbox (just for Matlab I/O during membrane segmentation). For results visualization it is required Chimera, IMOD, and Paraview.

VERY IMPORTANT: some computer skills are required to follow this tutorial as Python, Matlab and Bash scripts has to be executed and/or edited. The default settings guarantees to obtain a similar results to those presented here, we also encourage the user to play with different input data as well.

2 Executing the scripts

To complete this tutorial it is required to execute Python, Matlab and Bash scripts.

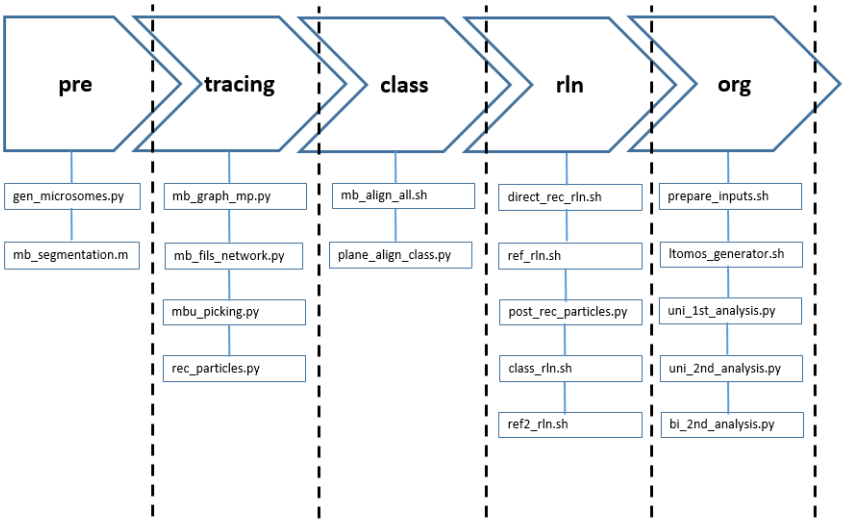

Figure 1: The general workflow.

- Python scripts (*.py) are executed in the terminal as:
  - user@cpu: python script.py
- Matlab scripts (*.m) are executed in the Matlab terminal as:
  - >> script.m
- Bash scripts (*.sh) are divided in two categories, (1) which are directly run in the terminal (it requires execution permissions) and (2) which are submitted to a computers cluster:
  1. user@cpu: ./script.sh
  2. user@cpu: qsub script.sh

3 Synthetic input data

The folder **pre** contains the scripts for input data generation and its segmentation.

3.1 Microsome generation

Python script **gen_microsomes.py** generates a serial of tomograms with a microsome each, here spherical vesicles, with proteins adhered to their membranes. Proteins with different structural models can be used, these proteins

are distributed randomly following different patterns: clustered (the last two models), co-localized (third from last with respect the first one) and the rest are randomly distributed. Depending of the number of output tomograms set running this script may take a long time (e.g. 50 tomograms and default settings with 3 processes may take almost one day).

The input parameters are:

- Input/Output:
  - out_dir: output directory
  - out_stem: stem added to the output generated files
- Multiprocessing parameters:
  - mp_npr: number of parallel processes
- Tomogram parameters:
  - tm_nt: integer with the number of tomograms to generate
  - tm_size: 3-tuple with the tomogram dimesions (in pixels)
  - tm_res: pixel resolution (nm/pixel)
  - tm_snr_rg: 2-tuple with the range of SNR for the output tomograms
  - tm_wedge: wedge angle in degrees (maximum sample tilting)
  - tm_wedge_rot: tilting axis rotation in degrees
  - tm_bin: tomogram binning factor for segmentation and particle picking (particles for subtomogram averaging will be reconstructed at bin 0)
  - tm_rsz: 3-tuple with the sub-volume dimension in pixels for rotating the models.
- Microsome parameters:
  - mc_mbt: microsome membrane thickness in nm
  - mc_mbs: membrane lipid layer decay in nm
  - mc_ip_min_dst: minimum distance between particles
  - mc_1st_crad: radius for clustered random models (the two last ones)
  - mc_c_jump_prob: jumping probability for clustered random models
  - mc_in_models: averaged distance for co-localized random models
  - mc_in_model: tuple with the paths for input models
  - mc_avg_nparts: averaged number of particles per tomogram per each model, tuple with the same length as 'mc_in_model'
  - mc_3sg_nparts: sigma for Gaussian distribution for the number of particles

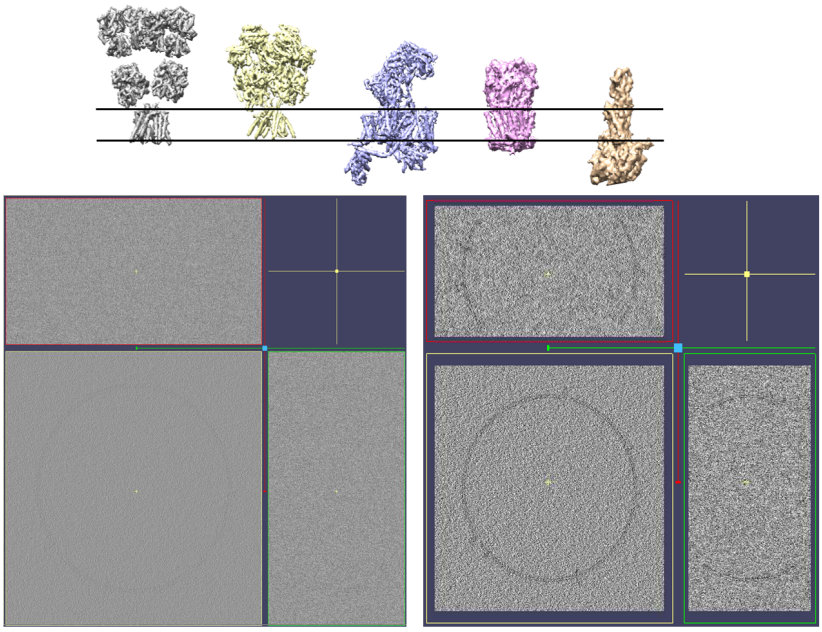

Figure 2: (Top) Structural models used for membrane bound proteins. (Bottom) Tomographic slices of a generated microsome, in bin0 (left) and in bin2 (right). Visualization generated with Chimera and IMOD.

  - mc_zh: sub-volumes model center
  - mc_halo_z: top and bottom halo without with only noise for the output tomograms

- Multiprocessing parameters:
  - mp_npr: number of parallel processes

The output of this scripts are:

- Set of tomograms synthetically generated with a microsome each at bin0 and bin2
- STAR file with the paths for the tomograms
- Some additional files for visualization.

3.2 Membrane segmentation

The Matlab script **mb_segmentation.m** segments the membrane of the binning 2 microsome generated in Section 3.1.

This scripts makes usage of TomoSemeMemTV software (package included in PySeg system) and TOM Toolbox (not included). TOM Toolbox is just used for tomogram I/O in Matlab, then alternative routines could be used with very little modifications to avoid the necessity of TOM Toolbox.

The input are:

- Input/Output directories:
  - in_star: STAR file with the tomograms at original resolution (rlnMicrographName column) and its binned 2 counterpart (rlnImageName column).
  - dout: Output directory
  - out_star: output STAR file where the membrane segmentations (psSegImage column), the binned tomograms and the original tomograms are associated.
- Parameters:
  - s, d: scale factor for TomoSegMemTV, their values should be the same
  - t: membrane thickness factor for TomoSegMemTV
  - v, m, w, e, u: advanced TomoSegMemTV parameters, we recommend use the defaults.
  - tb: binarization threshold after applying TomoSegMemTv
  - mb_ht: membrane half-thickness in pixels
  - halo_z: slices to discard (to set zero) on top and bottom of the tomogram
  - max_mb_dst: maximum distance to membrane for foreground in the output segmentation (0-background, 1-membrane, 2-foreground)

The outputs of this script are:

- A segmentation for each microsome at binning 2 resolution, the membrane is labeled as 1, the foreground (here the membrane surroundings) as 2 and the background (the rest) as 0.
- A STAR file associating each segmentation with the corresponding microsome.

4 Membrane bound densities tracing

The folder **tracing** contains the scripts for tracing the electron dense structures associated to microsome membranes. In addition, it also provides a initial guess for the localization of the membrane bound structures.

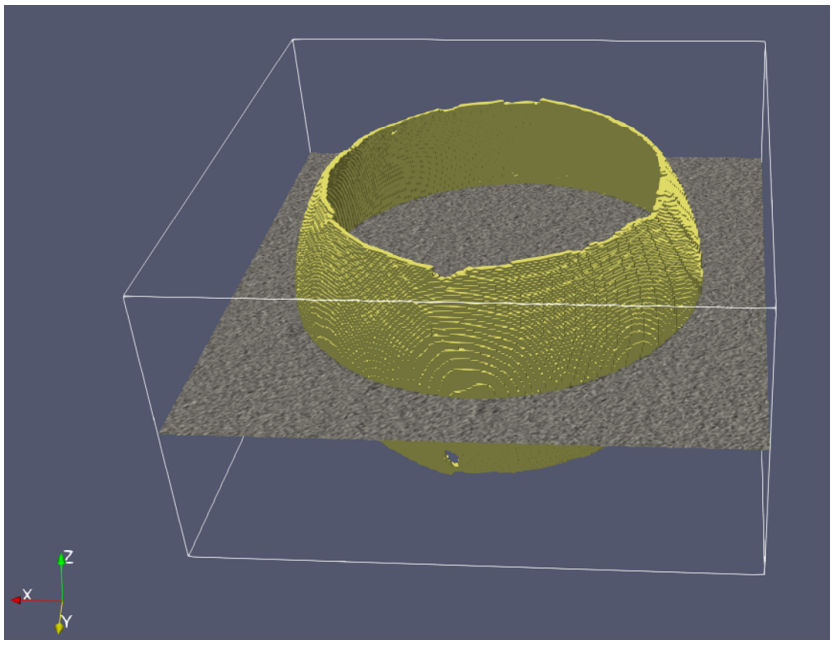

Figure 3: 3D view of the membrane segmentation (yellow) of a microsome. Visualization generated with Paraview.

4.1 Spatially embedded graph

The Python script **mb_graph_mp.py** generates a spatially embedded graph for each microsome. This graph traces all electron density structures in the membrane surrounding and also recovers their connectivity.

- Input files:
  - in_star: The STAR file generated by mb_segmentation.m with the micromoes and their corresponding segmentations.
- Graph computation parameters:
  - res: pixel size in nm
  - s_sig: sigma for preliminary low-pass Gaussian filtering the input density map.
  - v_den: graph vertices density (vertex/nm**3) to be achieved within membranes by simplifications. It the initial density (before simplification) is larger then this value can not be achieve and some WARNING messages will be printed, some possible solutions: decrease v_den value, decrease s_sig, try with data at higher resolution.
  - ve_ratio: maximum ratio between graph vertices and edges.

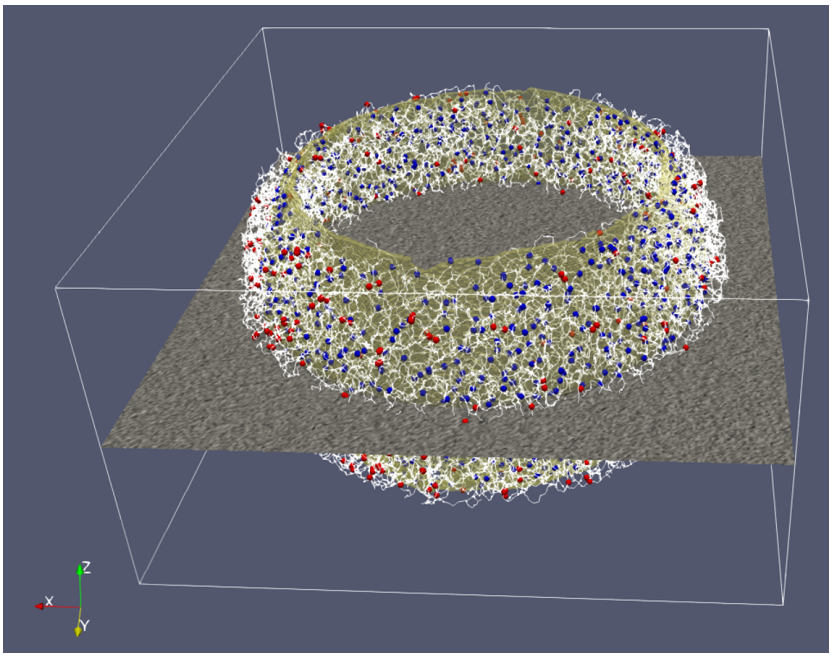

Figure 4: 3D view of the spatially embedded graph. Segmented membrane in transparent yellow, graph vertices embedded in the membrane in red spheres and the rest in blue, graph arcs are displayed in white. Visualization generated with Paraview.

  - max_len: maximum distance to the segmented membrane to process.

- Multiprocessing parameters:
  - npr: number of parallel processes
- Advanced settings: the scripts has some advanced settings which allows a fine tuned graph computation, for the shake of simplicity they are not commented here as they are not required to successfully complete this tutorial.

The outputs of this script are:

- A spatially embedded graph for each microsome processed.
- A STAR file as given by mb_segmentation.m but with an additional column with the corresponding graph.
- Some additional files for visualization.

4.2 Filaments network

The Python script **mb_fils_network.py** find the graph filaments associated with the membrane. PySeg is flexible to many definition for filaments but in this tutorial a filament is the shortest path on the graph between a graph vertex (in red in Figure 4) and the nearest vertex (in blue in Figure 4) within the membrane.

- Input files:
  - in_star: the output STAR file of mb_graph_batch.py with the graphs.
  - in_sources: an XML with the definition for membrane embedded graph vertices.
  - in_targets: an XML with the definition for the vertices not embedded in the membrane.
- Geometrical filament parameters:
  - g_rg_len: range (2-tuple) for the valid filament geodesic length.
  - g_rg_sin: range (2-tuple) for the valid filament sinuosity.
  - g_rg_eud: range (2-tuple) for the valid filament Euclidean distance.
- Advanced settings: the scripts has some advanced settings which allows a fine tuned computation, for the shake of simplicity they are not commented here as they are not required to successfully complete this tutorial.

The outputs of this script are:

- A filament network for each graph processed.
- A STAR file as given by mb_graph_batch.py but with an additional column with the corresponding filament network.
- Some additional files for visualization.

4.3 Particle picking

The Python script **mbu_picking.py** picks particles from an input list of filament network, with picking we mean the center point selection for the further sub-volume reconstruction. In addition, a initial guess for Tilt and Psi angles (Relion format) is provided, this information is obtained from the membrane normal vector at the picking point. PySeg allows to define different slices (sub-volumes defined with respect to a membrane) from where the picking points are selected, in this tutorial we choose the region in the surroundings of the membrane.

- Input files:

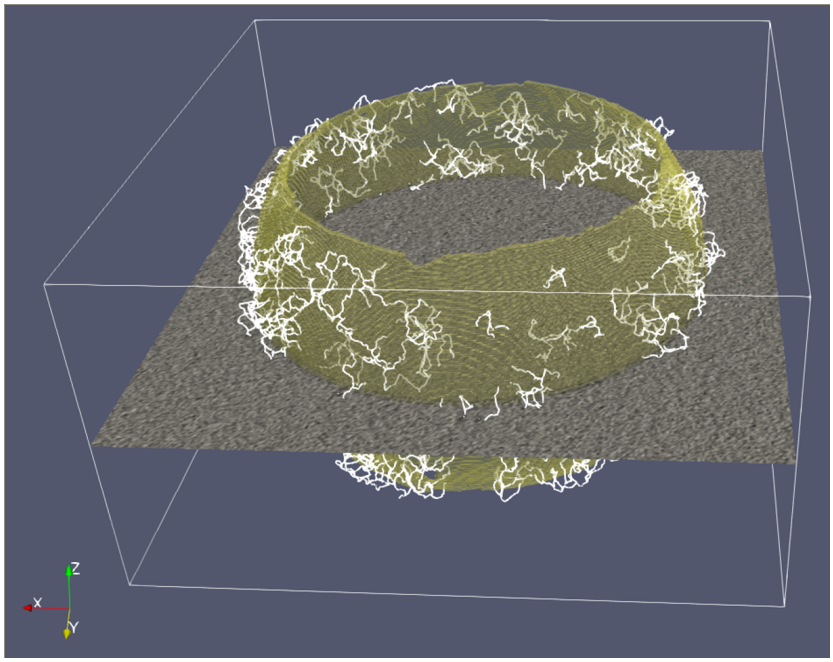

Figure 5: 3D view of the filaments network. Filaments in white and the segmented membrane in transparent yellow. Visualization generated with Paraview.

  - in_star: a STAR file with the filament networks a generated by mb_fils_network.py
  - slices_file: an XML file with the slice definition

- Peaks (or particles) configuration:
  - peak_th: particles are sorted by their density values, this values sets the percentile with the lowest density value to be discarded.
  - peak_ns: radius in nm for scale suppression to reduce particle picking overlapping.

- Advanced settings: the scripts has some advanced settings which allows a fine tuned computation, for the shake of simplicity they are not commented here as they are not required to successfully complete this tutorial. However, it is important to remark that the parameter **norm_proj** is set to True for projecting all picked point to the membrane surface.

The outputs of this script are:

- A STAR file with the picked particles.
- Some additional files for visualization.

4.4 Particle reconstruction

The Python script **rec_particles.py** reconstruct a subvolume for each picked particle an a STAR with the reconstructed particles file Relion compatible.

- Input files:
  - in_star: a STAR file with particles picked by **mbu_picking.py**
  - in_ctf: the subvolume with CTF.
  - in_mask_norm: subvolume mask for gray values normalization (required if '_psSegImage' not in the input STAR)

- Particle pre-processing settings:
  - do_bin: resolution ratio between the subvolumes reconstructed and the picking tomograms.
  - do_ang_prior: copy the prior angular information from the input STAR file.
  - do_ang_rnd: angle for randomization.
  - do_noise: add random noise out the mask
  - do_use_fg: to select foreground or background to select the random values for noise
  - do_norm: do gray value normalization (Relion convention)

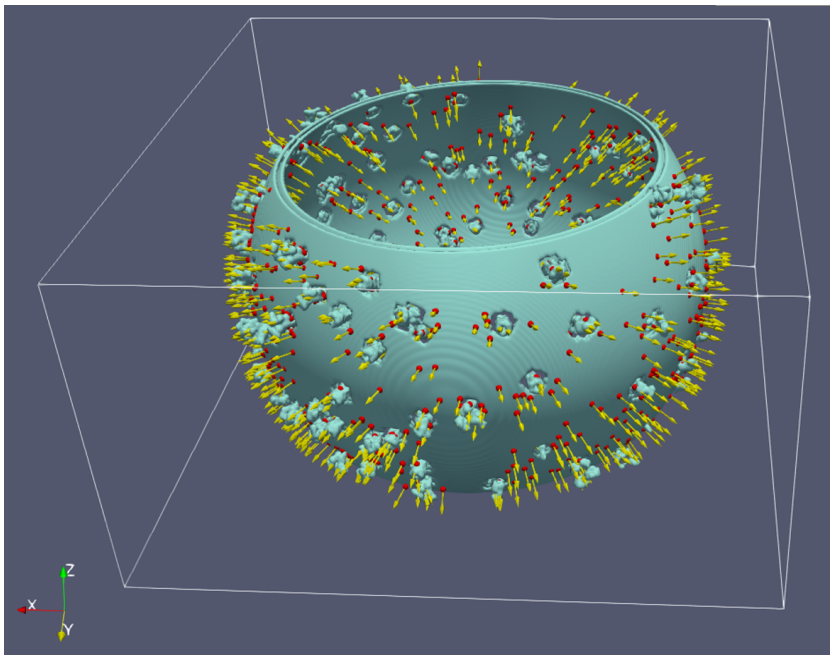

Figure 6: Membrane bound template-free particle picking. The microsome ground truth (membrane + proteins) generated by gen_microsome.py in blue. The picking center in red and their associated normal in yellow. Visualization generated with Paraview.

- Multiprocessing parameters:
  - mp_npr: number of parallel processes

The outputs of this script are:

- A subvolume for each input picked particle.
- A STAR file for indexing the reconstructed subvolumes compatible with Relion.

In addition to rec_particles.py it is also available the script **post_rec_particles.py**. The second allows to post-process any already reconstructed list or particles to update the angular prior and randomization information as well as to randomize the gray values out of an input mask, this script will be used later in this tutorial.

5 Unsupervised classification

The folder **class** contains a scripts for unsupervised and deterministic structural classification by Affinity Propagation (AP) algorithm.

5.1 Membrane alignment

The following Relion script is executed on the terminal to directly reconstruct an initial average that can be used for the further refinement.

- user@cpu: relion_reconstruct --i ../../../../data/tutorials/synth_sumb/rec/particles_rln.star --o ../../../../data/tutorials/synth_sumb/rec/dr_particles.mrc --angpix 2.62 --maxres 45

The bash script **mb_align_all.sh** is an example how to submit to a cluster of computers a Relion call for improving the alignment with respect the membrane of the picked particles, to do so it applies a constrained search for in-plane angle (prior angular information for Tilt and Psi angles) and C10 symmetry, the symmetry axis is normal to the membrane. This scripts is prepared to be executed in the cluster of the MPI of Biochemistry, so the parameters related with the management of the grid engine and multiprocessing may require modifications depending on the your computing architecture. Here a spherical mask with a radius of 60 pixel is used.

5.2 Coarse classification

The Python script **plane_align_class.py** structurally classify the particles previously aligned with respect the membrane, this classification is unsupervised and deterministic. Here we use AP algorithm but the scripts also allows to utilize PCA+Kmeans and PCA+AG (Agglomerative Clustering) instead, contrarily to AP the other require to introduce the final number of classes. Here we only explain the settings for AP.

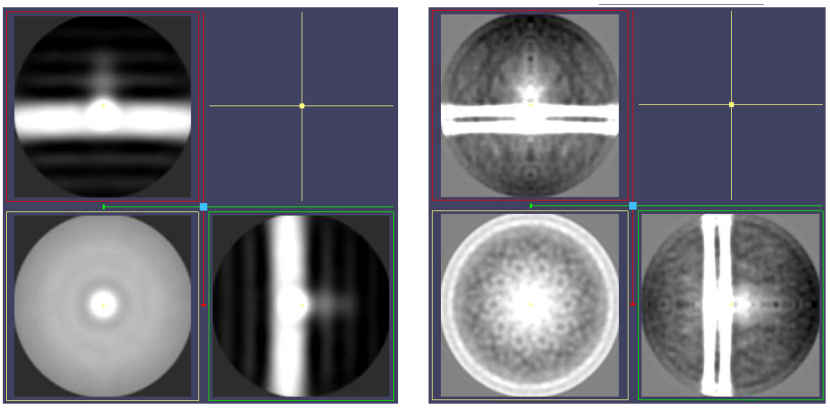

Figure 7: (Left) Direct reconstruction. (Right) Average after the membrane alignment. Visualizations generated with IMOD.

- Input files:
  - in_star: a STAR file with particles to classify
- Particles pre-processing:
  - pp_mask: mask for focusing the analysis. We recommed to use cylindrical mask as particles will be radially averaged, that can be done in TOM toolbox.
  - pp_low_sg: sigma for the Gaussian to low-pass filter the particles.
  - pp_3d: activates r-weighting
- AP settings:
  - ap_pref: usually a negative value, the tendency to create classes increases when this value increases
- Multiprocessing:
  - pp_npr: number of parallel processors.
- Advanced settings: the scripts has some advanced settings which allows a fine tuned classification, for the shake of simplicity they are not commented here as they are not required to successfully complete this tutorial.

The outputs of this script are:

- A STAR file with the particles for all classes, the class is identified by the column rlnClassNumber.
- A STAR file for each class with only the particles within the class.

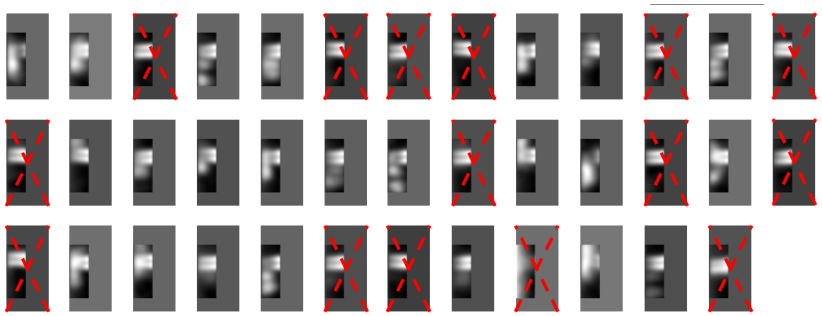

Figure 8: First AP classification round, red dashed X marks the classes discarded for the next round. Here each class is represented by the average of its particles. Protein localization is mirrored in respect of membrane plane when we compare with Figure 7.

- Two folder with a 2D image (radial average) per class:
  - The class exemplary particle.
  - The average of all particles.

Here this script is run two times. Firstly, the purpose is to clean the picked particles by discarding those classes that does not contain a membrane-bound protein (see Figure 8) or a membrane (more common in experimental data). Secondly, we classify a STAR file with the survivor classes merged, now we set the mask to focus the analysis to proteins and excluding the membrane contribution (see Figure 9), the idea is to classify the particles only considering proteins structural differences. In the first round the input is the particles output STAR file generated in , and in the second one the results of merging the not discarded particles STAR files of the previous round.

6 3D Classification and refinement

Starting from the membrane aligned particles and the classes obtained in Section 5.2 we use here Relion to refine the structural analysis, the aim is to obtain a 3D density average reconstruction for each protein as well as to finally determine their localization on the tomograms. In rln folder there are available an exemplary set of Relion scripts for being execute on MPI of Biochemistry cluster, therefore depending on your computer architecture or Relion version they may require some modification. The process (see Figure) is sumarized as:

1. Firstly for each class in Figure 9 we obtain a direct reconstruction by running the script **direct_rec_rln.sh**
2. We do an initial 3D refinement by submitting to the cluster **ref_rln.sh**

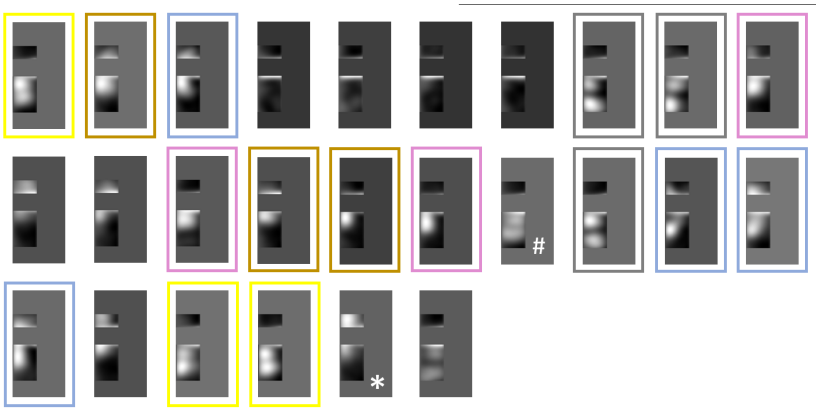

Figure 9: Second AP classification round, color frames indicate which classes are grouped together (see Section 6) based on their structural similarity for further analysis. Here the mask discard the membrane. Class with * seems to represent the same protein as those classes in dark yellow but picked from the other side of the membrane, # shows an example of class potentially not centered properly.

- In this example further results can be improved slightly if the particles STAR files obtained before this step are re-reconstructed using the Python script **post_rec_particles.py**

3. A 3D classification, this process is specific for each protein in **class_rln.sh** you have an initial example.

4. A final 3D refinement for each isolated protein, the script is **ref2_rln.sh**

7 Quantitative analysis for organization

The folder **stat** contains some scripts for the statistical analysis of molecular localization distribution on a segmented Volume Of Interest (VOI), here VOIs are the microsome membranes.

7.1 Surface model generation

Now we generate a surface model for each protein in order to consider their shape for volume exclusion during the statistical analysis. In principle we should take as input the final density averages obtained above (see Figure 12), however here we take those associated density used to generate the synthetic tomogram in order to easily get rid of the membrane (see Figure 2), otherwise membrane should be removed manually. The procedure is outlined as:

1. Open in Paraview the density to process (e.g. 4uqj_fit_s3.mrc)

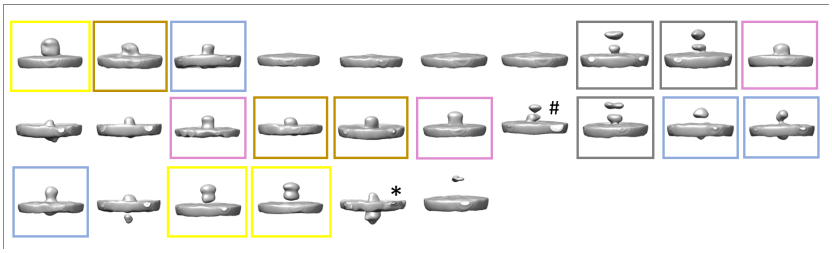

Figure 10: Direct reconstruction of the classes in Figure 9 (step 2 above). Colored frames match Figure 9 but protein localization is mirrored in respect the membrane plane. Class with * seems to represent the same protein as those classes in dark yellow but picked from the other side of the membrane, # shows an example of class potentially not centered properly.

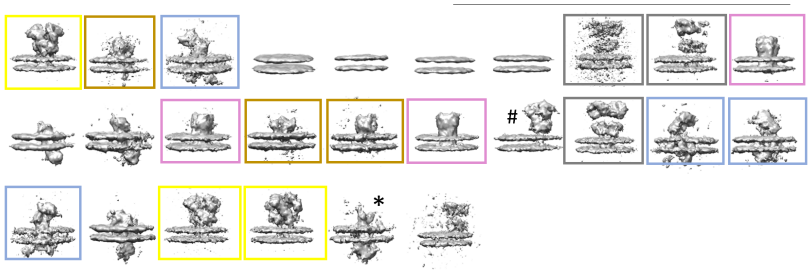

Figure 11: Refinement of the classes in Figure 9 (step 2 above). Colored frames match Figure 10. Class with * indeed represent the same protein as those classes in dark yellow but picked from the other side of the membrane, # shows an example of class not centered properly.

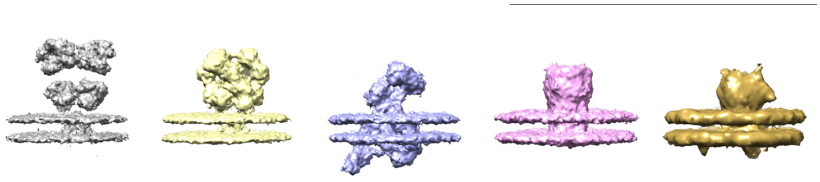

Figure 12: Final structures obtained after grouping AP classes in Figure 10, afterwards they are 3D classified and refined with Relion (steps 3 and 4 above). Colors match the initial structures used for generating the synthetic data (see Figure 2). Protein in dark yellow is mirrored in respect of membrane plane compared to those in Figure 2.

2. Scaling the subvolume using Filter->Transform. Here the input pixel size 2.62 A/px and the picking and segmentation is done at binned twice resolution so the scaling factor is (1/(4*2.62)) = 0.09542. This factor is introduced in the filter in Properties->Scale for X, Y and Z dimensions.

3. Center the structure using Filter->Transform. Here we have a subvolume of X, Y, and Z dimensions 32.50 (coordinate center in (16.25,16.25,16.25)) so in the filter Properties->Translate we set -16.25 for X,Y,Z. At the end of this step the center of the sub-volume must be the coordinate (0,0,0).

4. Closed surface generation using Filter->Contour. Here as surface is generated by applying iso-surface on the input density (scalar field) subvolume, here use a single isosurface with value 0.1.

5. (Optional) Surface decimation using Filter->Decimate. It allows to reduce number of triangles to represent the surface, the lower the number of triangles the lower computations are required for moving these surfaces in the next scripts, nevertheless the higher the number the smoother the surface. Here we use a Target Reduction of 0.6.

This procedure has to be repeated for each protein where initial input files are {4uqj,5pe5,5gjv,5kxi,5vai}_fit_s3.mrc.

7.2 Pre-processing

The Python script **ltomos_generator.py** load the input list of particles and segmentation and generates some intermediate necessary for any further script for statistical analysis.

- Input files:
  - in_star: A STAR file where each row correspond with a list of particles to process, columns are; the STAR file with particles (here the final products obtained in Figure 12), the surface model, the surface model used in simulations, the pixel size used for segmentation and picking, and the pixel size used for subtomogram averaging.
  - in_seg: A STAR file for pairing rlnMicrographName (orginal tomograms) with their corresponding segmenations. In some cases segmentation tomograms correspond only with fractions of the original tomogram, psSegOff{X,Y,Z} indicates the left-front-top coordinate of the segmented tomogram within the original, psSeg{Rot,Tilt,Psi} allows the possibility of rotate the segmentation. Neither psSegOff{X,Y,Z} nor psSeg{Rot,Tilt,Psi} are used in this tutorial.
- Segmentation pre-processing:
  - sg_lbl: label for the VOI in the segmented tomogram
  - sg_pj: if True then particles centers are projected on the VOI surface in case it is not embedded, it is required for processing membranes.

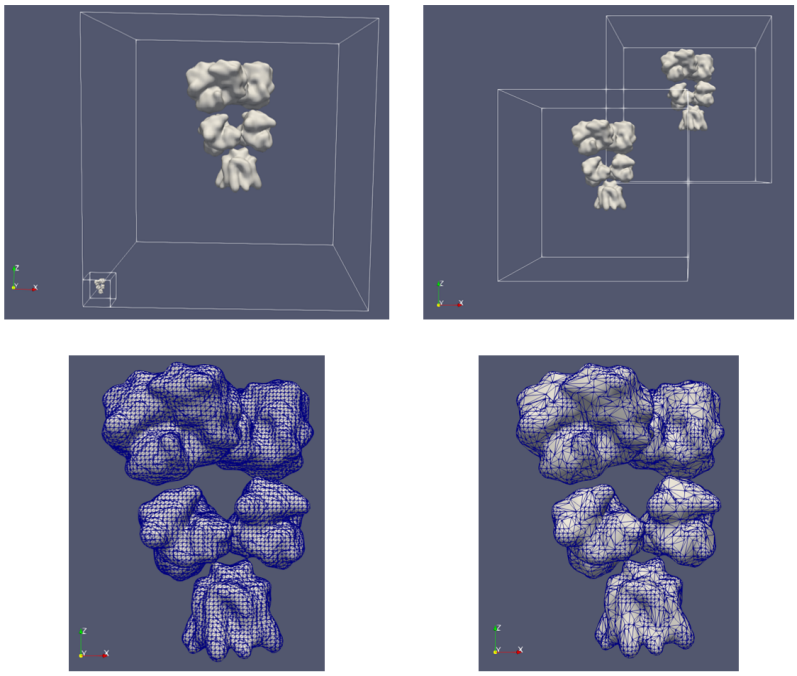

Figure 13: Surface model generation with Paraview. (Top-left) Scaling step. (Top-right) Centering. (Bottom-left) Surface generation. (Bottom-right) Triangles decimation.

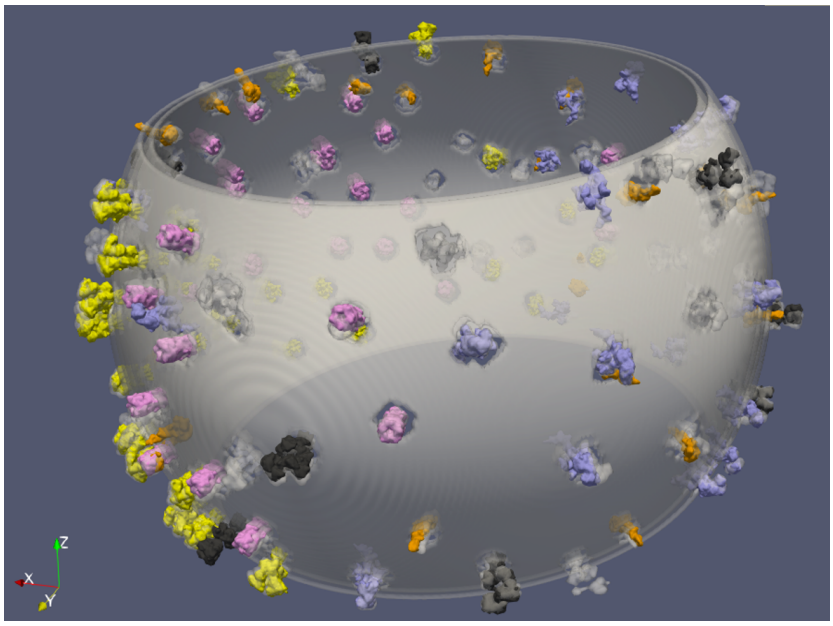

Figure 14: Surface models (color fit the models showed in Figure 12) placed at their original positions on the microsome at the proper scale. In transparent gray the original synthetic data without distortions (ground truth). In general we can observe a good agreement for the final results obtained and the original ground truth.

- Post-processing:
  - pt_ssup: scale suppression, if two particles closer than this distance then one is removed, it allows to get rid of double picked particles.

The outputs of this script are:

- A STAR file for indexing the files required by the scripts for statistical analysis (lists of tomograms, one per protein).
- Some files useful for visualization in Paraview.

7.3 Univariate 1st order analysis

The Python script **uni_1st_analysis.py** carries out a 1st order univariate analysis on the list of tomograms generated in Subsection 7.2. Functions computed are Function G, F and G. For each protein they are computed; separately for each tomogram (microsome), aggregated for the whole dataset.

- Input files:

  - in_star: A STAR file with the input list of tomograms to process.
  - in_wspace: by default None. Optionally it allows to load an already computed workspace, then only the plots are regenerated. This is useful when we want just want to edit the plots and it is not needed to re-compute the statistics.

- Analysis settings:
  - ana_res: resolution (nm/px)
  - ana_nbins: number of bins for histograms.
  - ana_rmax: maximum distance to consider.
  - ana_f_npoints: number of random points for Function F.

- Simulation settings:
  - p_nsims: number of simulations per tomogram and list of tomograms, the random model used is the Completely Spatially Randomness with Volume exclusion (CSRV).
  - p_per: percentile for the intervals of confidence [p_per, 100-p_per]%.

- Multiprocessing settings:
  - mp_npr: number of parallel processes for simulations.

- Advanced settings: the scripts has some advanced settings which allows a fine tuned analysis, for the shake of simplicity they are not commented here as they are not required to successfully complete this tutorial.

The output of this scripts are:

- For each protein (list of tomograms) an aggregated analysis for the functions demanded.
- For each tomogram (microsome) an analysis separated by proteins of the functions demanded.

VERY IMPORTANT: in this tutorial we do not execute this scripts as this analysis is now well suited for membrane bound proteins, for the moment it does not include the possibility of using geodesic distances among proteins.

7.4 Univariate 2nd order analysis

The Python script **uni_2nd_analysis.py** carries out a 2nd order univariate analysis on the lists of tomograms generated in Subsection 7.2. Functions computed are Function L, O and RDF (Radial Distribution Function). For each protein they are computed separately for each tomogram (microsome) and aggregated for the whole dataset. Results obtained here, see Figure 15, confirm that proteins with IDs 1 and 3 are significantly clustered but the rest are

randomly distributed on the microsome membrane, we know that in advance because the particles of these proteins were intentionally distributed in clusters when the initial synthetic data were generated.

- Input files:
  - in_star: A STAR file with the input list of tomograms to process.
  - in_wspace: by default None. Optionally it allows to load an already computed workspace, then only the plots are regenerated. This is useful when we want just want to edit the plots and it is not needed to re-compute the statistics.
- Analysis settings:
  - ana_res: resolution (nm/px)
  - ana_rg: array with the distance to sample
  - ana_shell_thick: if None then it is not considered so the Function L is computed, if a value the it sets a the shell thickness to compute Function O instead.
  - and_rdf: if True the RDF instead of Function O is computed, not considered if ana_shell_thick=None.
  - ana_fmm: if True then Fast Marching Method is used to compute geodesic distances, otherwise Euclidean distances are computed.
- Simulation settings:
  - p_nsims: number of simulations per tomogram and list of tomograms, the random model used is the Completely Spatially Randomness with Volume exclusion (CSRV).
  - p_per: percentile for the intervals of confidence [p_per, 100-p_per]%.
- Multiprocessing settings:
  - mp_npr: number of parallel processes for simulations and functions computation.
- Advanced settings: the scripts has some advanced settings which allows a fine tuned analysis, for the shake of simplicity they are not commented here as they are not required to successfully complete this tutorial.

The output of this scripts are:

- For each protein (list of tomograms) an aggregated analysis for the functions demanded.
- For each tomogram (microsome) an analysis separated by proteins of the functions demanded.

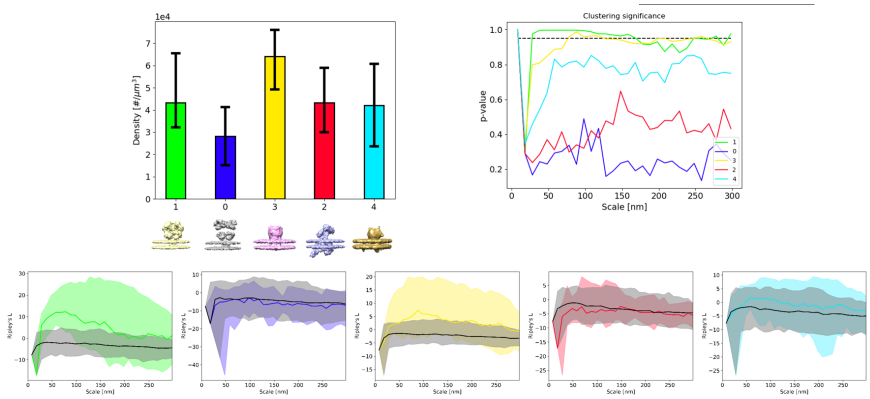

Figure 15: Some statistics obtained from univariate second order analysis. (Top-left) Averaged density for all tomograms and proteins, their structural model is displayed below of each ID (1, 0, 3, 2, 4), colors do not match with those in Figure 12 because they are assigned arbitrarily by the processing script. (Top-right) Clustering significance, only proteins with IDs 1, 3 show a high p-value beyond the initial transitory phase, the rest are distributed randomly for all scales, color lines match with density bars. (Bottom-row) The Ripley's L function for each protein considering all microsomes, color lines match with density bars.

7.5 Bivariate 2nd order analysis

The Python script **bi_2nd_analysis.py** carries out a 2nd order bivariate analysis on the lists of tomograms generated in Subsection 7.2 with respect a reference list of tomograms. Functions computed are Function L, O and RDF (Radial Distribution Function). For each protein, apart from the reference, they are computed separately for each tomogram (microsome) and aggregated for the whole dataset. Results obtained here, see Figure 15, confirm that protein with IDs 0 (reference) is slightly more correlated with the protein with ID 4 than the rest, we know that in advance because protein particles with ID where located randomly at a random distance to an instance of protein ID, this random distance follow a Gaussian distribution with $\sigma = 30$ nm.

- Input files:
  - in_star: A STAR file with the input list of tomograms to process.
  - in_wspace: by default None. Optionally it allows to load an already computed workspace, then only the plots are regenerated. This is useful when we want just want to edit the plots and it is not needed to re-compute the statistics.
- Analysis settings:

  - ana_res: resolution (nm/px)
  - ana_rg: array with the distance to sample
  - ana_shell_thick: if None then it is not considered so the Function L is computed, if a value the it sets a the shell thickness to compute Function O instead.
  - and_rdf: if True the RDF instead of Function O is computed, not considered if ana_shell_thick=None.
  - ana_fmm: if True then Fast Marching Method is used to compute geodesic distances, otherwise Euclidean distances are computed.

- Simulation settings:
  - p_nsims: number of simulations per tomogram and list of tomograms, the random model used is the Completely Spatially Randomness with Volume exclusion (CSRV).
  - p_per: percentile for the intervals of confidence [p_per, 100-p_per]%.

- Multiprocessing settings:
  - mp_npr: number of parallel processes for simulations and functions computation.

- Advanced settings: the scripts has some advanced settings which allows a fine tuned analysis, for the shake of simplicity they are not commented here as they are not required to successfully complete this tutorial.

The output of this scripts are:

- For each protein (list of tomograms) an aggregated analysis for the functions demanded.
- For each tomogram (microsome) an analysis separated by proteins of the functions demanded.

8 References

The original reference associated with this software:

- Martinez-Sanchez et al, "Template-free detection and classification of heterogeneous membrane-bound complexes in cryo-electron tomograms" accepted in *Nature Methods* (pre-print in bioRxiv)

A more detailed information about the organization analysis can be found in:

- Martinez-Sanchez et al, "On to the statistical spatial analysis for cryo-electron tomography" in preparation

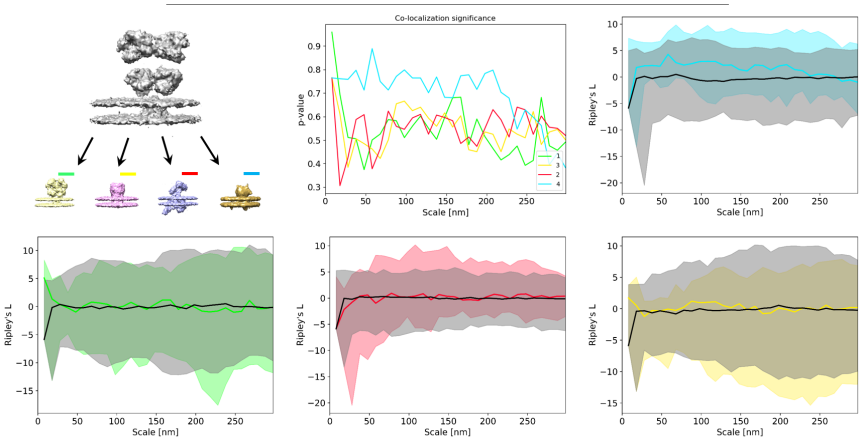

Figure 16: Some statistics obtained from bivariate second order analysis. (Top-left) The reference protein used (ID 0). (Top-center) Co-localization significance, protein with ID 4 shows a higher significance value than the rest. (Top-left) Ripley's L function for the colocalization of protein ID 4 with respect to protein ID 0, in average the colocalization is consistently higher than the random case but its does not exceed the interval of confidence for any scale. (Bottom-row) The Ripley's L function for the colocalization of the rest of protein with respect protein ID 0. Color lines match with density bars in Figure 15 and the lines in the top-left panel.