Single-particle processing in RELION-3.1

Sjors H.W. Scheres
(ccpem@jiscmail.ac.uk)

October 9, 2019

Abstract

This tutorial provides an introduction to the use of RELION-3.1 for cryo-EM structure determination. This tutorial covers the entire single-particle analysis workflow in RELION-3.1: beam-induced motion-correction, CTF estimation; automated particle picking; particle extraction; 2D class averaging; SGD-based initial model generation; 3D classification; high-resolution 3D refinement; CTF refinement and higher-order aberration correction; the processing of movies from direct-electron detectors; and final map sharpening and local-resolution estimation. Carefully going through this tutorial should take less than a day (if you have a suitable GPU or if you follow our precalculated results). After that, you should be able to run RELION on your own data.

This tutorial uses a test data set on beta-galactosidase that was kindly given to us by Takayuki Kato from the Namba group at Osaka university, Japan. It was collected on a 200kV JEOL cryo-ARM microscope. The data and our precalculated results may be downloaded and unpacked using the commands below. The full data set is also available at EMPIAR-10204.

```
wget ftp://ftp.mrc-lmb.cam.ac.uk/pub/scheres/relion30_tutorial_data.tar
wget ftp://ftp.mrc-lmb.cam.ac.uk/pub/scheres/relion31_tutorial_precalculated_results.tar.gz
tar -xf relion30_tutorial_data.tar
tar -zxf relion31_tutorial_precalculated_results.tar.gz
```

If you have any questions about RELION, first read this entire document, check the FAQ on the RELION Wiki and the archives of the CCPEM mailing list. If that doesn't help, subscribe to the CCPEM email list and use the email address above for asking your question. *Please, please, please, do not send us direct emails, as we can no longer respond to all of those.*

Contents

1 What's new in release 3.1?

1.1 Aberration corrections and optics groups

One of the major new features in RELION-3.1 is a correction for higher-order aberrations in the data, i.e. besides the beamtilt correction already present in RELION-3.0, the current version can also estimate and correct for trefoil and tetrafoil, as well as deviations from the nominal spherical aberration (Cs). The corresponding paper can be found on bioRxiv [21]. The signal to estimate these aberrations is calculated by averaging over particles from multiple micrographs. To allow for multiple subsets of a data set having different Zernike coefficients, RELION-3.1 implements the new concept of *optics groups*. Optics groups are defined in a separate table called `data_optics` at the top of a STAR file, which will also contain a table called `data_movies`, `data_micrographs` or `data_particles`, depending on what type of images it refers to. The second table is similar to the content of STAR files in previous releases, but contains a new column called `rlnOpticsGroup`, which is also present in the `data_optics` table. Common CTF-parameters, like `rlnVoltage` and `_rlnSphericalAberration`, but also the new `rlnOddZernike` and `rlnEvenZernike`, can be stored once for each optics group in the `data_optics` table, without the need to store them for each particle/micrograph in the second table.

The same program that handles higher-order aberrations can also be used to refine differences in (anisotropic) magnification between the reference and (groups of) the particles. Besides correcting for anisotropic magnification in the data, this is also useful when combining data from different scopes. As of release 3.1, the program that does 2D/3D classification and 3D refinement (`relion_refine`) can combine particles with different box sizes and pixel sizes in a single refinement, and the magnification refinement can be used to correct small errors in the (calibrated) pixel sizes. The box and pixel size of the input reference (or the first optics group in 2D classification) will be used for the reconstructions/class averages. You may want to check they are on the desired scale before running classifications or refinements!

Upon reading STAR files that were generated in older releases of RELION, RELION-3.1 will attempt to convert these automatically into the RELION-3.1-style STAR files. Therefore, moving a project from an older release to RELION-3.1 should be easy. However, please note that RELION-3.1-style STAR files cannot

be read by older releases. Therefore, it will be more difficult to go back from a RELION-3.1 project to an older release.

1.2 The External job-type

RELION-3.1 allows execution of third-party software within the RELION pipeline through the new External job-type. See section 14.4 for details on how to use this.

1.3 *Schedules* for on-the-fly processing

The python script `relion_it.py` in RELION-3.0 has been replaced by a new framework of *Schedules*, which implement decision-based scheduling and execution of RELION jobs. This comes with its own GUI interface. See section 14.5 for details on how to use this.

1.4 General tweaks

Several tweaks have been made to enhance user experience:

- The pipeliner no longer looks for output files to see whether a job has finished. Instead, upon successful exit, all programs that are launched from within the RELION pipeline will write out a file called `RELION_EXIT_SUCCESS` in the job directory. This avoids problems with subsequent execution of scheduled jobs with slow disc I/O.
- Likewise, when encountering an error, all programs will write out a file called `RELION_EXIT_FAILURE`. The GUI will recognise these jobs and use a red font in the Finished jobs list. Note that incorrectly labeled jobs can be changed using the 'Mask as finished' or 'Mark as failed' options from the Job actions pull-down menu.
- There is an 'Abort running' option on the Job actions pull-down menu, which will trigger the currently selected job to abort. This works because all jobs that are executed from within the RELION pipeline will be on the lookout for a file called `RELION_JOB_ABORT_NOW` in their output directory. When this file is detected, the job will exit prematurely and write out a `RELION_EXIT_ABORTED` file in the job directory. Thereby, users no longer need to kill undesired processes through the queuing or operating system. The GUI will display aborted jobs with a strike-through red font in the Finished jobs list.
- When a job execution has given an error, in previous releases the user would need to fix the error through the input parameters, and then launch

a new job. They would then typically delete the old job. RELION-3.1 allows to directly overwrite the old job. This is accessible on Linux systems through 'ALT+o' or through the 'Overwrite continue' option from the 'File menu'. Note that the `run.out` and `run.err` files will be deleted upon a job overwrite.

1.5 Tweaks to helical processing

Several new functionalities were implemented for helical processing:

- The `relion_helix_inimodel2d` program can be used to generate initial 3D reference maps for helices, in particular for amyloids, from 2D classes that span an entire cross-over (see section 14.6.1).
- The translational offsets along the direction of the helical axis can now be restricted to a single rise in 2D-classification.
- The 3D refinement and 3D classification now can use a prior on the first Euler angle, (`rlnAngleRotPrior`), which was implemented by Kent Thurber from the Tycko lab at the NIH.

2 Preprocessing

2.1 Getting organised

We recommend to create a single directory per project, i.e. per structure you want to determine. We'll call this the project directory. **It is important to always launch the RELION graphical user-interface (GUI) from the project directory.** Inside the project directory you should make a separate directory to store all your raw micrographs or micrograph movies in MRC or TIFF format. We like to call this directory `Movies/` if all movies are in one directory, or for example `Movies/15jan16/` and `Movies/23jan16/` if they are in different directories (e.g. because they were collected on different dates). If for some reason you do not want to place your movies inside the RELION project directory, then inside the project directory you can also make a symbolic link to the directory where your movies are stored.

Single-image micrographs should have a `.mrc` extension, movies can have a `.mrc`, `.mrcs`, `.tif` or `.tiff` extension. When you unpacked the tutorial test data, the (`Movies/`) directory was created. It should contain 24 movies in compressed TIFF format, a gain-reference file (`gain.mrc`) and a `NOTES` file with information about the experiment.

We will start by launching the RELION GUI. As said before, this GUI always needs to be launched from the project directory. To prevent errors with this, the GUI will ask for confirmation the first time you launch it in a new directory. Therefore, the first time you launch the GUI in a new directory, you should not use the "&" character to launch it in the background. Make sure you are inside the project directory, and launch the GUI by typing:

```
relion
```

and answer "y" to set up a new RELION project here.

The first thing to do is to import the set of recorded micrograph movies into the pipeline. Select "Import" from the job-type browser on the left, and fill in the following parameters on the Movies/mics tab:

- `Import raw movies/micrographs?` `Yes`
- `Raw input files:` `Movies/*.tiff`
- `Are these multi-frame movies?` `Yes`

  (Set this to "No" if these are single-frame micrographs)
- `Optics group name:` `opticsGroup1`

  (This field can be used to divide the data set into multiple optics groups: separately import each optics group with its own name, and then use the "Join star files" jobtype to combine the groups.

- `MTF of the detector:` `mtf_k2_200kV.star`
- `Pixel size (Angstrom):` `0.885`
- `Voltage (kV):` `200`
- `Spherical aberration (mm):` `1.4`
- `Amplitude contrast:` `0.1`
- `Beamtilt in X (mrad):` `0`
- `Beamtilt in Y (mrad):` `0`

On the Others tab, make sure the following is set:

- `Import other node types?` `No`

You may provide a meaningful alias (for example: `movies`) for this job in the white field named `Current job: Give_alias_here`. Clicking the Run! button will launch the job. A directory called `Import/job001/` will be created, together with a symbolic link to this directory that is called `Import/movies`. Inside the newly created directory a STAR file with all the movies is created. Have a look at it using:

```
less Import/job001/movies.star
```

If you had extracted your particles in a different software package, then instead of going through the Preprocessing steps below, you would use the same Import job-type to import particles STAR file, 3D references, 3D masks, etc. Note that this is NOT the recommended way to run RELION, and that the user is responsible for generating correct STAR files.

2.2 Beam-induced motion correction

The Motion correction job-type implements RELION's own (CPU-based) implementation of the UCSF MOTIONCOR2 program for convenient whole-frame movie alignment, as well as a wrapper to the (GPU-based) MOTIONCOR2 program itself [20]. Besides executing the calculations on the CPU/GPU, there are three other differences between the two implementations:

- Bayesian polishing (for per-particle motion-correction; see section 9) can only read local motion tracks from our own implementation;
- The MOTIONCOR2 program performs outlier-pixel detection on-the-fly, and this information is not conveyed to Bayesian polishing, which may result in unexpectedly bad particles after polishing;
- Our own implementation can write out the sum of power spectra over several movie frames, which can be passed directly into CTFFIND4-1 for faster CTF-estimation.

For these three reasons, we now favour running our own implementation.

On the I/O tab set:

- `Input movies STAR file:` `Import/movies/movies.star`

  (Note that the Browse button will only list movie STAR files.)

- `First frame for corrected sum:` `1`
- `Last frame for corrected sum:` `0`

  (This will result in using all movie frames.)

- `Dose per frame (e/A2)` `1.277`
- `Pre-exposure (e/A2)` `0`
- `Do dose-weighting?` `Yes`
- `Save non-dose-weighted as well?` `No`

  (In some cases non-dose-weighted micrographs give better CTF estimates. To save disk space, we're not using this option here as the data are very good anyway.)

- `Save sum of power spectra?` `Yes`
- `Sum of power spectra every e/A2:` `4`

  (This seems to be a good value according to measurements by Greg McMullan and Richard Henderson.)

Fill in the Motion tab as follows:

- `Bfactor:` `150`

  (use larger values for super-resolution movies)

- `Number of patches X,Y` `5 5`
- `Group frames:` `1`
- `Binning factor:` `1`

  (we often use 2 for super-resolution movies)

- `Gain-reference image:` `Movies/gain.mrc`

  (This can be used to provide a gain-reference file for on-the-fly gain-reference correction. This is necessary in this case, as these movies are not yet gain-corrected.)

- `Gain rotation:` `No rotation (0)`
- `Gain flip:` `No flipping (0)`

- `Defect file:`

  (This can be used to mask away broken pixels on the detector. Formats supported in our own implementation and in UCSF MOTIONCOR2 are either a text file in UCSF MOTIONCOR2 format (each line contains four numbers: x, y, width and height of a defect region); or a defect map (an image in MRC or TIFF format, where 0=good and 1=bad pixels). The coordinate system is the same as the input movie before application of binning, rotation and/or flipping. **Note that defect text files produced by serialem are NOT supported!** However, one can convert a serialEM-style defect file into a defect map using IMOD.)

- `Use RELION's own implementation?` `Yes`

  (this reduces the requirement to install the UCSF implementation. If you have the UCSF program installed anyway, you could also use that one. In that case, you also need to fill in the options below.)

Fill in the Running tab as follows:

- `Number of MPI procs:` `1`

  (Assuming you're running this tutorial on a local computer)

- `Number of threads:` `12`

  (As these movies are 24 frames, each thread will do two movie frames)

- `Submit to queue?` `No`

  (Again, assuming you're running this tutorial on a local computer)

Executing this program takes approximately 5 minutes when using 12 threads on a reasonably modern machine. Note that our own implementation of the MOTIONCOR2 algorithm does not use a GPU. It is however multi-threaded. As each thread will work independently on a movie frame, it is optimal to use a number of threads such that the number of movie frames divided by the number threads is an integer number. As these movies have 24 frames, using 12 threads will result in 2 frames being processed by each thread. You can look at the estimated beam-induced shifts, and their statistics over the entire data set, by selecting the `out: logfile.pdf` from the Display: button below the run buttons, or you can look at the summed micrographs by selecting `out: corrected_micrographs.star`. Depending on the size of your screen, you should probably downscale the micrographs (`Scale: 0.3`) and use `Sigma contrast: 3` and few columns (something like `Number of columns: 3`) for convenient visualisation. Note that you cannot select any micrographs from this display. If you want to exclude micrographs at this point (which we will not do, because they are all fine), you could use the Subset selection job-type.

2.3 CTF estimation

Next, we will estimate the CTF parameters for each corrected micrograph. You can use the CTF estimation job-type as a wrapper to Kai Zhang's GCTF to execute on the GPU, or you can also use Alexis Rohou and Niko Grigorieff's CTFFIND4.1 to execute efficiently on the CPU. We now prefer CTFFINDCTFFIND-4.1, as it is the only open-source option, and because it allows reading in the movie-averaged power spectra calculation by RELION's own implementation of the MOTIONCOR-2 algorithm. On the I/O tab, use the Browse button to select the `corrected_micrographs.star` file of the Motion correction job. Then fill in the other settings as follows:

On the I/O:

- `Use micrograph without dose-weighting?` `No`

  (These may have better Thon rings than the dose-weighted ones, but we decided in the previous step not to write these out)

- `Estimate phase shifts?` `No`

  (This is only useful for phase-plate data)

- `Amount of astigmatism (A):` `100`

  (Assuming your scope was reasonably well aligned, this value will be suitable for many data sets.)

On the CTFFIND-4.1 tab, set:

- `Use CTFFIND-4.1?` `Yes`
- `CTFFIND-4.1 executable:` `/wherever/it/is/ctffind.exe`
- `Use power spectra from MotionCorr job?` `Yes`

  (We can use these, as we told RELION's own implementation of the MOTIONCOR-2 algorithm to write these out in the previous section.)

- `Use exhaustive search?` `No`

  (In difficult cases, the slower exhaustive searches may yield better results. For these data, this is not necessary.)

- `Estimate CTF on window size (pix)` `-1`

  (If a positive value is given, a squared window of this size at the center of the micrograph will be used to estimate the CTF. This may be useful to exclude parts of the micrograph that are unsuitable for CTF estimation, e.g. the labels at the edge of photographic film.)

- `FFT box size (pix):` `512`
- `Minimum resolution (A):` `30`

- `Maximum resolution (A):` `5`
- `Minimum defocus cvalue (A):` `5000`
- `Maximum defocus cvalue (A):` `50000`
- `Defocus step size (A):` `500`

On the Gctf tab, make sure the option to use GCTF instead is set to No. On the Running tab, use six MPI processes to process the 24 micrographs in parallel. This took less than 10 seconds on our machine. Once the job finishes there are additional files for each micrograph inside the output `CtfFind/job003/Movies` directory: the `.ctf` file contains an image in `MRC` format with the computed power spectrum and the fitted CTF model; the `.log` file contains the output from CTFFIND or GCTF; (only in case of using CTFFIND, the `.com` file contains the script that was used to launch CTFFIND).

You can visualise all the Thon-ring images using the Display button, selecting `out: micrographs_ctf.star`. The zeros between the Thon rings in the experimental images should coincide with the ones in the model. Note that you can sort the display in order of defocus, maximum resolution, figure-of-merit, etc. The `logfile.pdf` file contains plots of useful parameters, such as defocus, astigmatism, estimated resolution, etc for all micrographs, and histograms of these values over the entire data set. Analysing these plots may be useful to spot problems in your data acquisition.

If you see CTF models that are not a satisfactory fit to the experimental Thon rings, you can delete the `.log` files for those micrographs, select the `CtfFind/job003` entry from the Finished jobs list, alter the parameters in the parameter-panel, and then re-run the job by clicking the Continue! button. Only those micrographs for which a `.log` file does not exist will be re-processed. You can do this until all CTF models are satisfactory. If this is not possible, or if you decide to discard micrographs because they have unsatisfactory Thon rins, you can use the Subset selection job-type to do this.

2.4 Manual particle picking

The next job-type Manual picking may be used to manually select particle coordinates in the (averaged) micrographs. We like to manually select at least several micrographs in order to get familiar with our data. Often, the manually selected particles to calculate reference-free 2D class averages, which will then be used as templates for automated particle picking of the entire data set. However, as of release 3.0, RELION also contains a reference-free auto-picking procedure based on a Laplacian-of-Gaussian (LoG) filter. In most cases tested thus far, this procedure provides reasonable starting coordinates, so that the Manual picking step may be skipped. The pre-shipped `Schedules` for on-the-fly processing make use of this functionality to perform fully automated on-the-fly

processing. In this tutorial, we will just launch a Manual picking job for illustrative purposes, and then proceed with LoG-based Auto-picking to generate the first set of particles.

Picking particles manually is a personal experience! If you don't like to pick particles in RELION, we also support coordinate file formats for Jude Short's XIMDISP [17] (with any extension); for XMIPP-2.4 [16] (with any extension); and for Steven Ludtke's E2BOXER.PY [18] (with a `.box` extension). If you use any of these, make sure to save the coordinate files as a text file in the same directory as from where you imported the micrographs (or movies), and with the same micrograph rootname, but a different (suffix+) extension as the micrograph, e.g. `Movies/006.box` or `Movies/006_pick.star` for micrograph `Movies/006.mrc`. You should then use the Import job-type and set `Node type:` to `2D/3D particle coordinates`. Make sure that the `Input Files:` field contains a linux wildcard, followed by the coordinate-file suffix, e.g. for the examples above you *have to* give `Movies/*.box` or `Movies/*_pick.star`, respectively.

On the I/O tab of the Manual picking job-type, use the Browse button to select the `micrographs_ctf.star` file that was created in `CtfFind/job003`, ignore the Colors tab, and fill in the Display tab as follows:

- `Particle diameter (A):` `200`

  (This merely controls the diameter of the circle that is displayed on the micrograph.)

- `Scale for micrographs:` `0.25`

  (But this depends on your screen size)

- `Sigma contrast:` `3`

  (Micrographs are often best display with "sigma-contrast", i.e. black will be 3 standard deviation below the mean and white will be 3 standard deviations above the mean. The grey-scale is always linear from black to white. See the DisplayImages entry on the RELION wiki for more details)

- `White value:` `0`

  (Use this to manually set which value will be white. For this to work, `Sigma contrast` should be set to 0)

- `Black value:` `0`

  (Use this to manually set which value will be black. For this to work, `Sigma contrast` should be set to 0)

- `Lowpass filter (A):` `20`

  (Playing with this may help you to see particles better in very noisy micrographs)

- `Highpass filter (A):` `0`

  (This is sometimes useful to remove dark-¿light gradients over the entire micrograph)

- `Pixel size:` `0.885`

  (This is needed to calculate the particle diameter, and the low- and high-pass filters)

- `Scale for CTF image:` `1`

  (This merely controls how large the Thon-ring images will be when you click the `CTF` button for any micrograph)

Run the job by clicking the Run! button and click on a few particles if you want to. However, as we will use the LoG-based autopicking in the next section, **you do not need to pick any if you don't want to**. If you were going to use manually picked particles for an initial 2D classification job, then you would need approximately 500-1,000 particles in order to calculate reasonable class averages. Left-mouse click for picking, middle-mouse click for deleting a picked particle, right-mouse click for a pop-up menu in which *you will need to save the coordinates!*. Note that you can always come back to pick more from where you left it (provided you saved the STAR files with the coordinates throught the pop-up menu), by selecting `ManualPick/job004` from the Finished jobs and clicking the Continue! button.

2.5 LoG-based auto-picking

We will now use a template-free auto-picking procedure based on a Laplacian-of-Gaussian (LoG) filter to select an initial set of particles. These particles will then be used in a 2D classification job to generate templates for a second Auto-picking job. Because we do not need many particles in the first round, we will only perform LoG-based auto-picking on the first 3 micrographs. Note that in general, one would probably perform LoG-based picking on all available micrographs to get as good templates as possible. However, here we only use a few micrographs to speed up the calculations in this tutorial.

First, in order to select a few micrographs, go to the Subset selection job, and on the I/O tab leave everything empty, except:

- `OR select from picked coords:` `ManualPick/job004/coords_suffix_manualpick.star`

  (which was generated when we saved a few manually picked coordinates. We are not going to use the coordinates here, we are only using that job to make a subset selecton of the micrographs.)

We used an alias `5mics` for this job. When you press Run!, the same pop-up window of the Manual picking job will appear again, i.e. the one with all

the pick and CTF buttons. Use the 'File' menu to 'Invert selection'; click on the check box in front of the first five micrographs to select those; and then use the 'File' menu again to 'Save selection'. This will result in a file called `ManualPick/job004/micrographs_selected.star`, which we will use for the Auto-picking job below.

Then, proceed to the Auto-picking job, and on the I/O tab set:

- `Input micrographs for autopick:` `Select/job005/micrographs_selected.star`
- `Pixel size in micrographs (A)` `-1`

  (The pixel size will be set automatically from the information in the input STAR file.)
- `2D references:`

  (Leave this empty for template-free LoG-based auto-picking.)
- `OR: provide a 3D reference?` `No`
- `OR: use Laplacian-of-Gaussian?` `Yes`

On the Laplacian tab, set:

- `Min. diameter for loG filter (A)` `150`
- `Max. diameter for loG filter (A)` `180`

  (This should correspond to the smallest and largest size of your particless projections in Ångstroms.)
- `Are the particles white?` `No`

  (They are black.)
- `Maximum resolution to consider` `20`

  (Just leave the default value here.)
- `Adjust default threshold` `0`

  (Positive values, i.e. high thresholds, will pick fewer particles, negative values will pick fewer particles. Useful values are probably in the range [-1,1], but in many cases the default value of zero will do a decent job. The threshold is moved this many standard deviations away from the average.)
- `Upper threshold` `5`

  (Use this to discard picks with LoG values that are this many standard deviations above the average, e.g. to avoid high contrast contamination like ice and ethane droplets. Good values depend on the contrast of micrographs and may need to be interactively explored; for low contrast micrographs, values of 1.5 may be reasonable, but this value is too low for the high-contrast micrographs in this tutorial.)

Ignore the References tab, and on the autopicking tab, the first four options will be ignored. Set the rest as follows:

- `Write FOM maps?` `No`

  (This will be used in the template-based picking below.)

- `Read FOM maps?` `No`

  (This will be used in the template-based picking below.)

- `Shrink factor:` `0`

  (By setting shrink to 0, the autopicking program will downscale the micrographs to the resolution of the lowpass filter on the references. This will go much faster and require less memory, which is convenient for doing this tutorial quickly. Values between 0 and 1 will be the resulting fraction of the micrograph size. Note that this will lead to somewhat less accurate picking than using shrink=1, i.e. no downscaling. A more detailed description of this new parameter is given in the next subsection.)

- `Use GPU acceleration?` `No`

  (LoG-based picking has not been GPU-accelerated as the calculations are very quick anyway.)

Ignore the Helix tab, and run using a single MPI processor on the Running tab. Perhaps an alias like `LoG` would be meaningful? Using a single processor, these calculations take about 15 seconds on our computer.

You can check the results by clicking the `coords_suffix_autopick` option from the Display: button. One could manually add/delete particles in the pop-up window that appears at this stage. In addition, one could choose to pick more or fewer particle by running a new job while adjusting the default threshold on the Laplacian tab, and/or the parameters for the stddev and avg of the noise on the autopicking tab. However, at this stage we are merely after a more-or-less OK initial set of particles for the generation of templates for a second auto-picking job, so in many cases this is probably not necessary.

2.6 Particle extraction

Once you have a coordinate file for every micrograph that you want to pick particles from, you can extract the corresponding particles and gather all required metadata through the Particle extraction job-type. On the corresponding I/O tab, set:

- `micrograph STAR file:` `CtfFind/job003/micrographs_ctf.star`

  (Use the `Browse` button to select this file. You could also chose the selected micrographs file from the ManualPick directory. It doesn't matter as there are only coordinate files for the three selected micrographs anyway.

Warning that coordinates files are missing for the rest of the micrographs will appear in red in the bottom window of the GUI.)

- `Coordinate-file suffix:` `AutoPick/job006/coords_suffix_autopick.star`

  (Use the `Browse` button to select this file)

- `OR re-extract refined particles?` `No`

  (This option allows you to use a `_data.star` file from a 2D cassification, 3D classification or 3D auto-refine job for re-extraction of only those particles in the STAR file. This may for example be useful if you had previously down-scaled your particles upon extraction, and after initial classifications you now want to perform refinements with the original-scaled particles. As of RELION-3.0, this functionality has been extended with an option to 're-center refined coordinates' on a user-specified X,Y,Z-coordinate in the 3D reference used for a 3D classification or 3D auto-refine job. This will adjust the X and Y origin coordinates of all particles, such that a reconstruction of the newly extracted particles will be centered on that X,Y,Z position. This is useful for focused refinements.)

- `Manually set pixel size?` `No`

  (This is only necessary when the input micrograph STAR file does NOT contain CTF information.)

On the extract tab you set the parameters for the actual particle extraction:

- `Particle box size (pix):` `256`

  (This should always be an even number!)

- `Invert contrast?` `Yes`

  (This makes white instead of black particles.)

- `Normalize particles?` `Yes`

  (We always normalize.)

- `Diameter background circle (pix):` `200`

  (Particles will be normalized to a mean value of zero and a standard-deviation of one for all pixels in the background area.The background area is defined as all pixels outside a circle with this given diameter in pixels (before rescaling). When specifying a negative value, a default value of 75% of the Particle box size will be used.)

- `Stddev for white dust removal:` `-1`

- `Stddev for black dust removal:` `-1`

(We only remove very white or black outlier pixels if we actually see them in the data. In such cases we would use stddev values of 5 or so. In this data set there are no outlier pixels, so we don't correct for them, and leave the default values at -1 (i.e. don't do anything).

- `Rescale particles?` `Yes`

  (Down-scaling particles will speed up computations. Therefore, we often down-scale particles in the initial stages of processing, in order to speed up the initial classifications of suitable particles. Once our reconstructions get close to the Nyquist frequency, we then re-extract the particles without down-scaling.)

- `Re-scaled sized (pixels)?` `64`

As we will later on also use the same job-type to extract all template-based auto-picked particles, it may be a good idea to give this job an alias like `LoG`. Ignore the Helix tab, and run using a single MPI processor.

Your particles will be extracted into MRC stacks (which always have an `.mrcs` extension in RELION) in a new directory called `Extract/job007/Movies/`. It's always a good idea to quickly check that all has gone OK by visualising your extracted particles selecting `out: particles.star` from the Display: button. Right-mouse clicking in the display window may be used for example to select all particles (`Invert selection`) and calculating the average of all unaligned particles (`Show average of selection`).

2.7 Making templates for auto-picking

To calculate templates for the subsequent auto-picking of all micrographs, we will use the 2D classification job-type. On the I/O tab, select the `Extract/job007/particles.star` file (using the Browse button), and on the CTF tab set:

- `Do CTF-correction?` `Yes`

  (We will perform full phase+amplitude correction inside the Bayesian framework)

- `Ignore CTFs until first peak?` `No`

  (This option is occasionally useful, when amplitude correction gives spuriously strong low-resolution components, and all particles get classified together in very few, fuzzy classes.)

On the Optimisation tab, set:

- `Number of classes:` `50`

  (For cryo-EM data we like to use on average at least approximately 100 particles per class. For negative stain one may use fewer, e.g. 20-50 particles per class. However, with this small number of particles, we have

observed a better separation into different classes by relaxing these numbers. Possibly, always having a minimum of 50 classes is not a bad idea.)

- `Regularisation parameter T:` `2`

  (For the exact definition of T, please refer to [13]. For cryo-EM 2D classification we typically use values of T=2-3, and for 3D classification values of 3-4. For negative stain sometimes slightly lower values are better. In general, if your class averages appear very noisy, then lower T; if your class averages remain too-low resolution, then increase T. The main thing is to be aware of overfitting high-resolution noise.)

- `Number of iterations:` `25`

  (We hardly ever change this)

- `Use fast subsets for large data sets?` `No`

  (If set to Yes, the first 5 iterations will be done with random subsets of only K*100 particles, with K being the number of classes; the next 5 with K*300 particles, the next 5 with 30% of the data set; and the final ones with all data. This was inspired by a cisTEM implementation by Tim Grant, Niko Grigorieff et al. This option may be useful to make classification of very large data sets. With hundreds of thousands of particles it is much faster. For a small data set like this one, it is not needed.)

- `Mask diameter (A):` `200`

  (This mask will be applied to all 2D class averages. It will also be used to remove solvent noise and neighbouring particles in the corner of the particle images. On one hand, you want to keep the diameter small, as too much noisy solvent and neighbouring particles may interfere with alignment. On the other hand, you want to make sure the diameter is larger than the longest dimension of your particles, as you do not want to clip off any signal from the class averages.)

- `Mask individual particles with zeros?` `Yes`

- `Limit resolution E-step to (A):` `-1`

  (If a positive value is given, then no frequencies beyond this value will be included in the alignment. This can also be useful to prevent overfitting. Here we don't really need it, but it could have been set to 10-15A anyway. Difficult classifications, i.e. with very noisy data, often benefit from limiting the resolution.)

On the Sampling tab we hardly ever change the defaults. Six degrees angular sampling is enough for most projects, although some large icosahedral viruses may benefit from finer angular samplings. In that case, one could first run 25 iterations with a sampling of 6 degrees, and then continue that same run (using the Continue! button) for an additional five iteration (by setting

`Number of iterations: 30` on the Optimisation tab) with a sampling of say 2 degrees. For this data set, this is NOT necessary at all. It is useful to note that the same Continue! button may also be used to resume a job that has somehow failed before, in which case one would not change any of the parameters. For continuation of 2D classification, 3D initial model, 3D classification, or 3D auto-refine jobs one always needs to specify the `_optimiser.star` file from the iteration from which one continues on the I/O tab.

Ignore the Helix tab, and on the Compute tab, set:

- `Use parallel disc I/O?` `Yes`

  (This way, all MPI slaves will read their own particles from disc. Use this option if you have a fast (parallel?) file system. Note that non-parallel file systems may not be able to handle parallel access from multiple MPI nodes. In such cases one could set this option to No. In that case, only the master MPI node will read in the particles and send them through the network to the MPI slaves.)

- `Number of pooled particles:` `30`

  (Particles are processed in individual batches by MPI slaves. During each batch, a stack of particle images is only opened and closed once to improve disk access times. All particle images of a single batch are read into memory together. The size of these batches is at least one particle per thread used. The nr_pooled_particles parameter controls how many particles are read together for each thread. If it is set to 30 and one uses 8 threads, batches of 30x8=240 particles will be read together. This may improve performance on systems where disk access, and particularly metadata handling of disk access, is a problem. Typically, when using GPUs we use values of 10-30; when using only CPUs we use much smaller values, like 3. This option has a modest cost of increased RAM usage.)

- `Pre-read all particles into RAM?` `Yes`

  (If set to Yes, all particle images will be read into computer memory, which will greatly speed up calculations on systems with slow disk access. However, one should of course be careful with the amount of RAM available. Because particles are read in double-precision, it will take (N * box_size * box_size * 4 / (1024 * 1024 * 1024)) Giga-bytes to read N particles into RAM. If parallel disc I/O is set to Yes, then all MPI slaves will read in all particles. If parallel disc I/O is set to No, then only the master reads all particles into RAM and sends those particles through the network to the MPI slaves during the refinement iterations.)

- `Copy particles to scratch directory?`

  (This is useful if you don't have enough RAM to pre-read all particles, but you do have a fast (SSD?) scratch disk on your computer. In that case,

specify the name of the scratch disk where you can make a temporary directory, e.g. /ssd)

- `Combine iterations through disc?` `No`

  (This way all MPI nodes combine their data at the end of each iteration through the network. If the network is your main bottle-neck or somehow causing problems, you can set this option to No. In that case, all MPI nodes will write/read their data to disc.)

- `Use GPU acceleration?` `Yes`

  (If you have a suitable GPU, this job will go much faster.)

- `Which GPUs to use:` `0:1:2:3`

  (This will depend on the available GPUs on your system! If you leave this empty, the program will try to figure out which GPUs to use, but you can explicitly tell it which GPU IDs , e.g. 0 or 1, to use. If you use multiple MPI-processors, you can run each MPI process on a specified GPU. GPU IDs for different MPI processes are separated by colons, e.g. 0:1:0:1 will run MPI process 0 and 2 on GPU 0, and MPI process 1 and 3 will run on GPU 1.)

On the Running tab, specify the 'Number of MPI processors' and the 'Number of threads' to use. The total number of requested CPUs, or cores, will be the product of the two values. Note that 2D classification, 3D classification, 3D initial model and 3D auto-refine use one MPI process as a master, which does not do any calculations itself, but sends jobs to the other MPI processors. Therefore, if one specifies 4 GPUs above, running with five MPI processes would be a good idea. Threads offer the advantage of more efficient RAM usage, whereas MPI parallelization scales better than threads. Often, for 3D classification and 3D auto-refine jobs you will probably want to use many threads in order to share the available RAM on each (multi-core) computing node. 2D classification is less memory-intensive, so you may not need so many threads. However, the points where communication between MPI processors (the bottle-neck in scalability there) becomes limiting in comparison with running more threads, is different on many different clusters, so you may need to play with these parameters to get optimal performance for your setup. We pre-read all particles into RAM, used parallel disc I/O, 4 GPUs and 5 MPI process with 6 threads each, and our job finished in approximately three minutes.

Because we will run more 2D classification jobs, it may again be a good idea to use a meaningful alias, for example `LoG`. You can look at the resulting class averages using the Display: button to select `out: run_it025_model.star` from. On the pop-up window, you may want to choose to look at the class averages in a specific order, e.g. based on `rlnClassDistribution` (in reverse order, i.e. from high-to-low instead of the default low-to-high) or on `rlnAccuracyRotations`.

2.8 Selecting templates for auto-picking

Selection of suitable class average images is done in the Subset selection job-type. On the I/O tab, remove the picked coords entry from before, and select the `Class2D/LoG/run_it025_model.star` file using the Browse button on the line with `Select classes from model.star:`.

On the Class options tab, give:

- `Re-center the class averages?` `Yes`

  (This option allows automated centering of the 2D class averages. The images are centered based on their center-of-mass, and the calculations for this require that the particles are WHITE (not black). Re-centering is often necessary, as class averages may become non-centered in the 2D classification run. In particular when using class average images for auto-picking it is important that the are centered, as otherwise all your particle coordinates will become systematically off-centered.)

- `Regroup the particles?` `No`

  (This option is useful when there are very few (selected) particles on individual micrographs, in which case the estimation of noise power spectra and scale factors become unstable. By default, the latter are calculated independently per micrograph. This option allows to grouping particles from multiple micrographs together in these calcutaions. RELION will warn you (in classification or auto-refine runs) when your groups become too small.)

Ignore the other tabs, and use an alias like `templates4autopick`. You may again want to order the class averages based on their `rlnClassDistribution`. Select a few class averages that represent different views of your particle. Don't repeat very similar views, and don't include bad class averages. We selected four templates from our run. Selection is done by left-clicking on the class averages. You can save your selection of class averages from the right-click pop-up menu using the `Save selected classes` option.

2.9 Auto-picking

We will now use the selected 2D class averages as templates in a reference-based run of the Auto-picking job-type. However, before we will run the auto-picking on all micrographs, we will need to optimise four of its main parameters on the autopicking tab: the `Picking threshold`, the `Minimum inter-particle distance`, the `Maximum stddev noise`, and the `Minimum avg noise`. This will be done on only a few micrographs in order to save time. We will use the same five micrographs we selected for the LoG-based auto-picking before.

Then, on the I/O tab of the Auto-picking job-type, set:

- `Input micrographs for autopick:` `Select/5mics/micrographs.star`
- `Pixel size in micrographs (A)` `-1`

  (The pixel size will be set automatically from the information in the input STAR file.)
- `2D references:` `Select/templates4autopick/class_averages.star`
- `OR: provide a 3D reference?` `No`
- `OR: use Laplacian-of-Gaussian?` `No`

Ignore the Laplacian, and on the References tab, set:

- `Lowpass filter references (A):` `20`

  (It is very important to use a low-pass filter that is significantly LOWER than the final resolution you aim to obtain from the data, to keep "Einstein-from-noise" artifacts at bay)
- `Highpass filter (A):` `-1`

  (If you give a positive value, e.g. 200, then the micrograph will be high-pass filtered prior to autopicking. This can help in case of strong grey-scale gradients across the micrograph.)
- `Pixel size in references (A):` `3.54`

  (If a negative value is given, the references are assumed to be on the same scale as the input micrographs. If this is not the case, e.g. because you rescaled particles that were used to create the references upon their extraction, then provide a positive value with the correct pixel size in the references here. **As we downscaled the particles by a factor of 4 (i.e. from 256 to 64) in the Particle extraction job, the pixel size in the references is now** $4 * 0.885 = 3.54\mathring{A}$**.**)
- `Mask diameter (A)` `-1`

  (When a negative value is given, the diameter of the mask will be determined automatically from the input reference images to be the same as the one used in the 2D classification job.)
- `Angular sampling (deg):` `5`

  (This value seems to work fine in almost all cases.)
- `References have inverted contrast?` `Yes`

  (Because we have black particles in the micrographs, and the references we will use are white.)
- `Are References CTF corrected?` `Yes`

  (Because we performed 2D class averaging with the CTF correction.)

- `Ignore CTFs until first peak:` `No`

  (Only use this option if you also did so in the 2D classification job that you used to create the references.)

On the autopicking tab, set:

- `Picking threshold:` `0.8`

  (This is the threshold in the FOM maps for the peak-search algorithms. Particles with FOMs below this value will not be picked.)

- `Minimum inter-particle distance (A):` `200`

  (This is the maximum allowed distance between two neighbouring particles. An iterative clustering algorithm will remove particles that are nearer than this distance to each other. Useful values for this parameter are often in the range of 50-60% of the particle diameter.)

- `Maximum stddev noise:` `-1`

  (This is useful to prevent picking in carbon areas, or areas with big contamination features. Peaks in areas where the background standard deviation in the normalized micrographs is higher than this value will be ignored. Useful values are probably in the range 1.0 to 1.2. Set to -1 to switch off the feature to eliminate peaks due to high background standard deviations.)

- `Minimum avg noise:` `-999`

  (This is useful to prevent picking in carbon areas, or areas with big contamination features. Peaks in areas where the background standard deviation in the normalized micrographs is higher than this value will be ignored. Useful values are probably in the range -0.5 to 0. Set to -999 to switch off the feature to eliminate peaks due to low average background densities.)

- `Write FOM maps?` `Yes`

  (See the explanation below.)

- `Read FOM maps?` `No`

  (See the explanation below.)

- `Shrink factor:` `0`

  (By setting shrink to 0, the autopicking program will downscale the micrographs to the resolution of the lowpass filter on the references. This will go much faster and require less memory, which is convenient for doing this tutorial quickly. Values between 0 and 1 will be the resulting fraction of the micrograph size. Note that this will lead to somewhat less accurate picking than using shrink=1, i.e. no downscaling. A more detailed description of this new parameter is given in the next subsection.)

- `Use GPU acceleration?` `Yes`

  (Only if you have a suitable GPU!)

- `Which GPUs to use:` `0`

  (If you leave this empty, the program will try to figure out which GPUs to use, but you can explicitly tell it which GPU IDs , e.g. 0 or 1, to use. If you use multiple MPI-processors (not for this case!), you can run each MPI process on a specified GPU. GPU IDs for different MPI processes are separated by colons, e.g. 0:1:0:1 will run MPI process 0 and 2 on GPU 0, and MPI process 1 and 3 will run on GPU 1.)

Ignore the Helix tab, and run using a single MPI processor on the Running tab. Perhaps an alias like `optimise_params` would be meaningful? When using GPU-acceleration, the job completes in half a minute.

The expensive part of this calculation is to calculate a probability-based figure-of-merit (related to the cross-correlation coefficient between each rotated reference and all positions in the micrographs. This calculation is followed by a much faster peak-detection algorithm that uses the threshold and minimum distance parameters mentioned above. Because these parameters need to be optimised, the program will write out so-called FOM maps as specified on the References tab. These are two large (micrograph-sized) files per reference. To avoid running into hard disc I/O problems, the autopicking program can only be run sequentially (hence the single MPI processor above) when writing out FOM maps.

Once the FOM maps have been written to disc they can be used to optimise the picking parameters much faster. First, examine the auto-picked particles with the current settings using the `coords_suffix_autopick` option from the Display: button of the job you just ran. Note that the display window will take its parameters (like size and sigma-contrast) from the last Manual picking job you executed. You can actually change those parameters in the Manual picking job-type, and save the settings use the option `Save job settings` from the top left `Jobs` menu. Do this, after you've set on the following on the Colors tab:

- `Blue<>red color particles?` `Yes`
- `MetaDataLabel for color:` `rlnAutopickFigureOfMerit`
- `STAR file with color label:`

  Leave this empty.

- `Blue value:` `1`
- `Red value:` `0`

Executing the job on the Running tab will produce a similar GUI with pick and CTF buttons as before. Open both micrographs from the display window,

and decide whether you would like to pick more or less particles (i.e. decrease or increase the threshold) and whether they could be closer together or not (fot setting the minimum inter-particle distance). Note that each particle is colored from red (very low FOM) to blue (very high FOM). You can leave the display windows for both micrographs open, while you proceed with the next step.

Select the `AutoPick/optimise_params` job from the Finished jobs, and change the parameters on the autopicking tab. Also, change:

- `Write FOM maps?` `No`
- `Read FOM maps?` `Yes`

When executing by clicking the Continue! button, this will re-read the previously written FOM maps from disc instead of re-doing all FOM calculations. The subsequent calculation of the new coordinates will then be done in a few seconds. Afterwards, you can right-click in the micrograph display windows and select `Reload coordinates` from the pop-up menu to read in the new set of coordinates. This way you can quickly optimise the two parameters.

Have a play around with all three parameters to see how they change the picking results. Once you know the parameters you want to use for auto-picking of all micrographs, you click the Auto-picking option in the job-type browser on the top left of the GUI to select a job with a Run! button. On the I/O tab, you replace the input micrographs STAR file with the 2 selected micrographs with the one from the original CTF estimation job (`CtfFind/job003/micrographs_ctf.star`). Leave everything as it was on the References tab, and on the autopicking tab set:

- `Picking threshold:` `0.0`
- `Minimum inter-particle distance (A):` `100`

  (Good values are often around 50-70% of the particle diameter.)
- `Maximum stddev noise:` `-1`
- `Minimum avg noise:` `-999`
- `Write FOM maps?` `No`
- `Read FOM maps?` `No`

This time, the job may be run in parallel (as no FOM maps will be written out). On the Running tab, specify the number of cores to run auto-picking on. The maximum useful number of MPI processors is the number of micrographs in the input STAR file. Using only a single MPI process and a single GPU, our calculation still finisged in about one minute. We used as alias `template`.

Note that there is an important difference in how the Continue! button works depending on whether you read/write FOM maps or not. When you either write

or read the FOM maps and you click Continue!, the program will re-pick all input micrographs (typically only a few). However, when you do not read, nor write FOM maps, i.e. in the second job where you'll autopick all micrographs, upon clicking the Continue! button, only those micrographs that were not autopicked yet will be done. This is useful in the iterative running of scheduled jobs, e.g. for on-the-fly data processing during your microscopy session. Also see section 14.5 for more details. If you want to-repick all micrographs with a new set of parameters (instead of doing only unfinished micrographs), then click the Auto-picking entry on the jobtype-browser on the left to get a Run! button instead, which will then make a new output directory.

You can again check the results by clicking the `coords_suffix_autopick` option from the Display: button. Some people like to manually go over all micrographs to remove false positives. For example, carbon edges or high-contrast artifacts on the micrographs are often mistaken for particles. You can do this for each micrograph using the pop-up window from the Display: button. Remove particles using the middle-mouse button; you can hold it down to remove many false positives in larger areas. Remember to save the new coordinates using the right-mouse click pop-up menu!

Once you're happy with the overall results, in order to save disc space you may want to delete the FOM-maps that were written out in the first step. You can use the `Gentle clean` option from the Job actions button to do that conveniently.

Once you are happy with your entire set of coordinates, you will need to re-run a Particle extraction job, keeping everything as before, and change the `Input coordinates` for the newly generated, autopick ones. This will generate your initial single-particle data set that will be used for further refinements below. Perhaps an alias like `template` would be meaning ful?

2.9.1 The shrink parameter

To enable faster processing, RELION implements a filtering option of the micrographs through the command-line argument `--shrink`. The simplest way to use it is to simply use `--shrink 0`. But it can be used with much more control. If this is desired, it works in the following way:

The default value for `--shrink` is 1.0, which has no effect and does not filter the micrographs at all. This is identical behaviour to versions prior to the RELION-2.0 autopicker.

- `--shrink` $val = 0$ results in a micrograph lowpassed to `--lowpass`, the same as that of the reference templates. This is recommended use for single-particle analysis (but not for helical picking).
- `--shrink` $0 < val \leq 1$ results in a size $= val \cdot micrograph\_size$, i.e. val is

a scaling factor applied to the micrographs original size.

- `--shrink` $val > 1$ results in a size $= val - (val \mod 2)$, i.e. the smallest even integer value lower than val. This is then a way to give the micrographs a specific (even) integer size.

If the new size is smaller than `--lowpass`, this results in a *non-fatal* warning, since this limits resolution beyond the requested (or default) resolution. Because the RELION autopicker does many FFTs, the size of micrographs is now also automatically adjusted to avoid major pitfalls of FFT computation. A note of this and how it can be overridden is displayed in the initial output of each autopicking the user performs.

3 Reference-free 2D class averaging

We almost always use reference-free 2D class averaging to throw away bad particles. Although we often try to only include good particles for the particle extraction step in the previous section (for example by manually supervising the auto-picking results, and by sorting the extracted particles), most of the times there are still particles in the data set that do not belong there. Because they do not average well together, they often go to relatively small classes that yield ugly 2D class averages. Throwing those away then becomes an efficient way of cleaning up your data.

3.1 Running the job

Most options will remain the same as explained when we were generating templates for the auto-picking in the previous section, but on the I/O tab of the 2D classification job-type, set:

- `Input images STAR file:` `Extract/template/particles.star`

and on the Optimisation tab, we used:

- `Number of classes:` `100`

  (because we now have more particles.)

You could use an alias like `template`. Using 4 GPUs, and 5 MPI processes, each with 6 threads, this job took 20 minutes on our computer. Perhaps a good time for a cup of coffee?

After the job has finished, we can launch a Subset selection job, with the `_model.star` file from this run as input. An alias like `class2d_template` may be meaningful. Now select all nice-looking classes by clicking on them (and/or using the right-mouse pop-up menu option `Select all classes above`). At

this point, if you would have used a low threshold in the auto-picking procedure, you should be very wary of "Einstein-from-noise" classes, which look like low-resolution ghosts of the templates used to pick them, on which high-resolution noise may have accumulated. Avoid those in the selection. After all good classes have been selected use the right-mouse pop-up menu option to save the selection.

Note that this procedure of selecting good classes may be repeated several times.

3.2 Analysing the results in more detail

If you are in a hurry to get through this tutorial, you can skip this sub-section. It contains more detailed information for the interested reader.

For every iteration of 2D or 3D classification RELION performs, it writes out a set of files. For the last iteration of our 2D class averaging calculation these are:

- `Class2D/template/run_it025_classes.mrcs` is the MRC stack with the resulting class averages. These are the images that will be displayed in the RELION GUI when you select the `_model.star` file from the Display: button on the main GUI. Note that RELION performs full CTF correction (if selected on the GUI), so your class averages are probably white on a black background. If the data is good, often they are very much like projections of a low-pass filtered atomic model. The quality of your 2D class averages are a very good indication of how good your 3D map will become. We like to see internal structure within projections of protein domains, and the solvent area around you particles should ideally be flat. Radially extending streaks in the solvent region are a typical sign of overfitting. If this happens, you could try to limit the resolution in the E-step of the 2D classification algorithm.

- `Class2D/template/run_it025_model.star` contains the model parameters that are refined besides the actual class averages (i.e. the distribution of the images over the classes, the spherical average of the signal-to-noise ratios in the reconstructed structures, the noise spectra of all groups, etc. Have a look at this file using the `less` command. In particular, check the distribution of particles over each class in the table `data_model_classes`. If you compare this with the class averages themselves, you will see that particles with few classes are low-resolution, while classes with many particles are high-resolution. This is an important feature of the Bayesian approach, as averaging over fewer particles will naturally lead to lower signal-to-noise ratios in the average. The estimated spectral signal-to-noise ratios for each class are stored in the `data_model_class_N` tables, where N is the number of each class. Likewise, the estimated noise spectra for each group are stored in the tables called `data_model_group_N`. The table `data_model_groups` stores a refined intensity scale-factor for each group: groups with values higher than one have a stronger signal than the average, relatively low-signal groups have values lower than one. These values are often

correlated with the defocus, but also depend on accumulated contamination and ice thickness.

- `Class2D/template/run_it025_data.star` contains all metadata related to the individual particles. Besides the information in the input `particles.star` file, there is now additional information about the optimal orientations, the optimal class assignment, the contribution to the log-likelihood, etc. Note that this file can be used again as input for a new refinement, as the STAR file format remains the same.
- `Class2D/template/run_it025_optimiser.star` contains some general information about the refinement process that is necessary for restarting an unfinished run. For example, if you think the process did not converge yet after 25 iterations (you could compare the class averages from iterations 24 and 25 to assess that), you could select this job in the Finished jobs panel, and on the I/O tab select this file for `Continue from here`, and then set `Number of iterations: 40` on the Optimisation tab. The job will then restart at iteration 26 and run until iteration 40. You might also choose to use a finer angular or translational sampling rate on the Sampling tab. Another useful feature of the optimiser.star files is that it's first line contains a comment with the exact command line argument that was given to this run.
- `Class2D/template/run_it025_sampling.star` contains information about the employed sampling rates. This file is also necessary for restarting.

3.3 Making groups

If you are in a hurry to get through this tutorial, you can skip this sub-section. It contains more detailed information for the interested reader.

RELION groups particles together to do two things: estimate their average noise power spectrum and estimate a single-number intensity scale factor that describes differences in overall signal-to-noise ratios between different parts of the data, e.g. due to ice thickness, defocus or contamination.

The default behaviour is to treat all particles from each micrograph as a separate group. This behaviour is fine if you have many particles per micrograph, but when you are using a high magnification, your sample is very diluted, or your final selection contains only a few particles per micrograph, then the estimation of the intensity scale factor (and the noise spectra) may become unstable. We generally recommend to have at least 10-20 particles in each group, but do note that initial numbers of particles per group may become much smaller after 2D and 3D classification.

In cases with few particles per micrograph we recommend to group particles from multiple micrographs together. For this purpose, the GUI implements a convenient functionality in the Subset selection job-type: when selecting a `_model.star` file on the I/O tab, one can use `Regroup particles? Yes` and

`Approximate nr of groups: 5` on the Class options tab to re-group all particles into 5 groups. (The actual number may vary somewhat from the input value, hence the "Approximate" on the input field.) This way, complicated grouping procedures in previous releases of RELION may be avoided. As the micrographs in this tutorial do contain sufficient particles, we will not use this procedure now.

Please note that the groups in RELION are very different from defocus groups that are sometimes used in other programs. RELION will always use per-particle (anisotropic) CTF correction, irrespective of the groups used.

4 *De novo* 3D model generation

RELION uses a Stochastic Gradient Descent (SGD) algorithm, as was first introduced in the cryoSPARC program [9], to generate a *de novo* 3D initial model from the 2D particles. Provided you have a reasonable distribution of viewing directions, and your data were good enough to yield detailed class averages in 2D classification, this algorithm is likely to yield a suitable, low-resolution model that can subsequently be used for 3D classification or 3D auto-refine.

4.1 Running the job

Select the `Select/class2d_template/particles.star` file on the I/O tab of the 3D initial model jobtype. Everything is aready in order on the CTF. Fill in the optimisation tab as follows (leave the defaults for the angular and offset sampling):

- `Number of classes` `1`

  (Sometimes, using more than one class may help in providing a 'sink' for sub-optimal particles that may still exist in the data set. The additional argument `--sgd_skip_anneal` may then also be useful. In this case, we will just use a single class in order to speed up things).

- `Mask diameter (A)` `200`

  (The same as before).

- `Flatten and enforce non-negative solvent` `Yes`

- `Symmetry` `C1`

  (If you don't know what the symmetry is, it is probably best to start with a C1 reconstruction. Also, some higher-symmetry objects may be easier to solve by SGD in C1 than in their correct space group. This data set is great data, and would also work in the correct point group D2. However, to illustrate how to proceed from C1 to D2, we will run the SGD in C1.)

Typically, in first instance one would not change anything on the SGD tab, as the default are suitable for many cases. However, in order to speed things up for this tutorial, we will only perform half the default number of iterations. Therefore change:

- `Number of initial iterations` `25`
- `Number of in-between iterations` `100`
- `Number of final iterations` `25`

On the Compute tab, optimise things for your system. You may well be able to pre-read the few thousand particles into RAM again. GPU acceleration will also yield speedups, though multiple maximisation steps during each iteration will slow things down compared to standard 2D or 3D refinements or classifications. We used an alias of `symC1` for this job. Using 4 GPU cards, 5 MPI processes and 6 threads per MPI process, this run took approximately 15 minutes on our system. If you didn't get that coffee before, perhaps now is a good time too...

4.2 Analysing the results

Look at the output volume (`InitialModel/job015/run_it150_class001.mrc`) with a 3D viewer like UCSF CHIMERA. If you recognise additional point group symmetry at this point, then you will need to align the symmetry axes with the main X,Y,Z axes of the coordinate system, according to RELION's conventions. Use the following command line instruction to do this:

```
relion_align_symmetry --i InitialModel/job015/run_it150_class001.mrc \
  --o InitialModel/job015/run_it150_class001_alignD2.mrc --sym D2
```

And after confirming in UCSF CHIMERA or `relion_display` that the symmetry axes in the map are now indeed aligned with the X, Y and Z-axes, we can now impose D2 symmetry using:

```
relion_image_handler --i InitialModel/job015/run_it150_class001_alignD2.mrc \
  --o InitialModel/job015/run_it150_class001_symD2.mrc --sym D2
```

The output map of the latter command should be similar to the input map. You could check this by:

```
relion_display --i InitialModel/job015/run_it150_class001_alignD2.mrc &
relion_display --i InitialModel/job015/run_it150_class001_symD2.mrc &
```

5 Unsupervised 3D classification

All data sets are heterogeneous! The question is how much you are willing to tolerate. RELION's 3D multi-reference refinement procedure provides a powerful

unsupervised 3D classification approach.

5.1 Running the job

Unsupervised 3D classifcation may be run from the 3D classification job-type. On the I/O tab set:

- `Input images STAR file:` `Select/class2d_template/particles.star`
- `Reference map:` `InitialModel/symC1/run_it150_class001_symD2.mrc`

  (Note that this map does not appear in the Browse button as it is not part of the pipeline. You can either type it's name into the entry field, or first import the map using the Import jobtype. Also note that, because we wil be running in symmetry C1, we could have also chosen to use the non-symmetric `InitialModel/job015/run_it150_class001.mrc`. However, already being in the right symmetry setting is more convenient later on.)

- `Reference mask (optional):`

  (Leave this empty. This is the place where we for example provided large/small-subunit masks for our focussed ribosome refinements. If left empty, a spherical mask with the particle diameter given on the Optimisation tab will be used. This introduces the least bias into the classification.)

On the Reference tab set:

- `Ref. map is on absolute greyscale:` `Yes`

  (Given that this map was reconstructed from this data set, it is already on the correct greyscale. Any map that is not reconstructed from the same data in RELION should probably be considered as not being on the correct greyscale.)

- `Initial low-pass filter (A):` `50`

  (One should NOT use high-resolution starting models as they may introduce bias into the refinement process. As also explained in [12], one should filter the initial map as much as one can. For ribosome we often use 70Å, for smaller particles we typically use values of 40-60Å.)

- `Symmetry:` `C1`

  (Although we know that this sample has D2 symmetry, **it is often a good idea to perform an initial classification without any symmetry**, so bad particles, which are not symmetric, can get separated from proper ones, and the symmetry can be verified in the reconstructed maps.)

On the CTF tab set:

- `Do CTF correction?` `Yes`

- `Has reference been CTF-corrected?` `Yes`

  (As this model was made using CTF-correction in the SGD.)

- `Ignore CTFs until first peak?` `No`

  (Only use this option if you also did so in the 2D classification job that you used to create the references.)

On the Optimisation tab set:

- `Number of classes:` `4`

  (Using more classes will divide the data set into more subsets, potentially describing more variability. The computational costs scales linearly with the number of classes, both in terms of CPU time and required computer memory.)

- `Regularisation parameter T:` `4`

  For the exact definition of T, please refer to [13]. For cryo-EM 2D classification we typically use values of T=1-2, and for 3D classification values of 2-4. For negative stain sometimes slightly lower values are better. In general, if your class averages appear noisy, then lower T; if your class averages remain too-low resolution, then increase T. The main thing is to be aware of overfitting high-resolution noise.

- `Number of iterations:` `25`

  (We typically do not change this.)

- `Use fast subsets (for large data sets)?:` `No`

  (This option will significantly speed up calculations for data sets of hundreds of thousands pf particles. However, sometimes performance is affected too. For small data sets like this one, we do not recommend using this option.)

- `Mask diameter (A):` `200`

  (Just use the same value as we did before in the 2D classification job-type.)

- `Mask individual particles with zeros?` `Yes`

- `Limit resolution E-step to (A):` `-1`

  (If a positive value is given, then no frequencies beyond this value will be included in the alignment. This can also be useful to prevent overfitting. Here we don't really need it, but it could have been set to 10-15A anyway.)

On the Sampling tab one usually does not need to change anything (only for large and highly symmetric particles, like icosahedral viruses, does one typically use a 3.7 degree angular sampling at this point). Ignore the Helix tab, and fill in the Compute tab like you did for the previous 2D-classification. Again, on the

Running tab, one may specify the `Number of MPI processors` and `threads` to use. As explained for the 2D classification job-type, 3D classification takes more memory than 2D classification, so often more threads are used. However, in this case the images are rather small and RAM-shortage may not be such a big issue. Perhaps you could use an alias like `first_exhaustive`, to indicate this is our first 3D classification and it uses exhaustive angular searches? On our computer with 4 GPUs, 5 MPIs and 6 threads, this calculation took approximately 10 minutes.

When analysing the resulting class reconstructions, it is extremely useful to also look at them in slices, not only as a thresholded map in for example UCSF CHIMERA. In the slices view you will get a much better impression of unresolved heterogeneity, which will show up as fuzzy or streaked regions in the slices. Slices also give a good impression of the flatness of the solvent region. Use the Display: button and select any of the reconstructions from the last iteration to open a slices-view in RELION.

When looking at your rendered maps in 3D, e.g. using UCSF CHIMERA, it is often a good idea to fit them all into the best one, as maps may rotate slightly during refinement. In CHIMERA, we use the `Tools -> Volume Data -> Fit in Map` tool for that. For looking at multiple maps alongside each other, we also like the `Tools -> Structure Comparison -> Tile Structures` tool, combined with the `independent` center-of-rotation method on the `Viewing` window.

As was the case for the 2D classification, one can again use the Subset selection to select a subset of the particles assigned to one or more classes. On the I/O tab select the `_model.star` file from the last iteration. The resulting display window will show central slices through the 4 refined models. Select the best classes, and save the corresponding particles using the right-mouse pop-up menu. Use an alias like `class3d_first_exhaustive`.

5.2 Analysing the results in more detail

Again, if you are in a hurry to get through this tutorial, you can skip this subsection. It contains more detailed information for the interested reader.

The output files are basically the same as for the 2D classification run (we're actually using the same code for 2D and 3D refinements). The only difference is that the map for each class is saved as a separate MRC map, e.g. `run_it025_class00?.mrc`, as opposed to the single MRC stack with 2D class averages that was output before.

As before, smaller classes will be low-pass filtered more strongly than large classes, and the spectral signal-to-noise ratios are stored in the `data_model_class_N` tables (with $N = 1, \ldots, K$) of the `_model.star` files. Perhaps now is a good

time to introduce two handy scripts that are useful to extract any type of data from STAR files. Try typing:

```
relion_star_printtable Class3D/first_exhaustive/run_it025_model.star
  data_model_class_1 rlnResolution rlnSsnrMap
```

It will print the two columns with the resolution (`rlnResolution`) and the spectral signal-to-noise ratio (`rlnSsnrMap`) from table `data_model_class_1` to the screen. You could redirect this to a file for subsequent plotting in your favourite program. Alternatively, if `gnuplot` is installed on your system, you may type:

```
relion_star_plottable Class3D/first_exhaustive/run_it025_model.star
  data_model_class_1 rlnResolution rlnSsnrMap
```

To check whether your run had converged, (as mentioned above) you could also monitor:

```
grep _rlnChangesOptimalClasses Class3D/first_exhaustive/run_it???_optimiser.star
```

As you may appreciate by now: the STAR files are a very convenient way of handling many different types of input and output data. Linux shell commands like `grep` and `awk`, possibly combined into scripts like `relion_star_printtable`, provide you with a flexible and powerful way to analyze your results.

6 High-resolution 3D refinement

Once a subset of sufficient homogeneity has been selected, one may use the 3D auto-refine procedure in RELION to refine this subset to high resolution in a fully automated manner. This procedure employs the so-called *gold-standard* way to calculate Fourier Shell Correlation (FSC) from independently refined half-reconstructions in order to estimate resolution, so that self-enhancing overfitting may be avoided [15]. Combined with a procedure to estimate the accuracy of the angular assignments [14], it automatically determines when a refinement has converged. Thereby, this procedure requires very little user input, i.e. it remains objective, and has been observed to yield excellent maps for many data sets. Another advantage is that one typically only needs to run it once, as there are hardly any parameters to optimize.

However, before we start our high-resolution refinement, we should first re-extract our current set of selected particles with less down-scaling, so that we can potentially go to higher resolution. To do this, go to the Particle extraction jobtype on the GUI, and on the I/O tab give:

- `micrograph STAR file:` `CtfFind/job003/micrographs_ctf.star`

  (This should still be there.)

- `Coordinate-file suffix:`

  (Leave this empty now.)

- `OR re-extract refined particles?` `Yes`

- `Refined particles STAR file:` `Select/class3d_first_exhaustive/particles.star`

  (Now, we will use only the refined subset of selected particles.)

- `Reset the refiend offsets to zero?` `No`

  (This would discard the translational offsets from the previous classification runs.)

- `OR: re-center refined coordinates?` `Yes`

  (This will re-center all the particles according to the aligned offsets from the 3D classification job above.)

- `Recenter on - X, Y, Z (pix)` `0 0 0`

  (We want to keep the centre of the molecule in the middle of the box.)

- `Manually set pixel size?` `No`

  (This is only necessary when the input micrograph STAR file does NOT contain CTF information.)

And on the extract tab, we keep everything as it was, except:

- `Particle box size (pix)` `360`

  (we will use a larger box, so that de-localised CTF signals can be better modeled. This is important for the CTF refinement later on.)

- `Rescale particles?` `Yes`

  (to prevent working with very large images, let's down-sample to a pixel size of 360*0.885/256=1.244 Å. This will limit our maximum achievable resolution to 2.5 Å, which is probably enough for such a small data set.)

- `Re-scaled size (pixels):` `256`

We used the alias `best3dclass_bigbox` for this job.

In addition, we will need to rescale the best map obtained thus far to the 256-pixel box size. This is done from the command-line:

```
relion_image_handler --i Class3D/first_exhaustive/run_it025_class001.mrc \
 --angpix 3.54 --rescale_angpix 1.244 --new_box 256 \
 --o Class3D/first_exhaustive/run_it025_class001_box256.mrc
```

6.1 Running the auto-refine job

On the I/O tab of the 3D auto-refine job-type set:

- `Input images STAR file:` `Extract/best3dclass_bigbox/particles.star`
- `Reference map:` `Class3D/first_exhaustive/run_it025_class001_box256.mrc`

  (Note this one is again not directly available through the Browse button.)
- `Reference mask (optional):`

  (leave this empty for now)

On the Reference tab, set:

- `Ref. map is on absolute greyscale?` `No`

  (because of the different normalisation of down-scaled images, the rescaled map is no longer on the correct absolute grey scale. Setting this option to 'No' is therefore important, and will correct the greyscale in the first iteration of the refinement.)
- `Initial low-pass filter (A)` `50`

  (We typically start auto-refinements from low-pass filtered maps to prevent bias towards high-frequency components in the map, and to maintain the "gold-standard" of completely independent refinements at resolutions higher than the initial one.)
- `Symmetry` `D2`

  (We now aim for high-resolution refinement, so imposing symmetry will effectively quadruple the number of particles.)

Parameters on the CTF, Optimisation and Auto-sampling tabs remain the same as they were in the 3D classification job. Note that the orientational sampling rates on the Sampling tab will only be used in the first few iterations, from there on the algorithm will automatically increase the angular sampling rates until convergence. Therefore, for all refinements with less than octahedral or icosahedral symmetry, we typically use the default angular sampling of 7.5 degrees, and local searches from a sampling of 1.8 degrees. Only for higher symmetry refinements, we use 3.7 degrees sampling and perform local searches from 0.9 degrees.

As the MPI nodes are divided between one master (who does nothing else than bossing the others around) and two sets of slaves who do all the work on the two half-sets, it is most efficient to use an odd number of MPI processors, and the minimum number of MPI processes for 3D auto-refine jobs is 3. Memory requirements may increase significantly at the final iteration, as all frequencies until Nyquist will be taken into account, so for larger sized boxes than the ones in this test data set you may want to run with as many threads as you

have cores on your cluster nodes. Perhaps an alias like `first3dref` would be meaningful?

6.2 Analysing the results

Also the output files are largely the same as for the 3D classification job. However, at every iteration the program writes out two `run_it0??_half?_model.star` and two `run_it0??_half?_class001.mrc` files: one for each independently refined half of the data. Only upon convergence a single `run_model.star` and `run_class001.mrc` file will be written out (without `_it0??` in their names). Because in the last iteration the two independent half-reconstructions are joined together, the resolution will typically improve significantly in the last iteration. Because the program will use all data out to Nyquist frequency, this iteration also requires more memory and CPU.

Note that the automated increase in angular sampling is an important aspect of the auto-refine procedure. It is based on signal-to-noise considerations that are explained in [14], to estimate the accuracy of the angular and translational assignments. The program will not use finer angular and translational sampling rates than it deems necessary (because it would not improve the results). The estimated accuracies and employed sampling rates, together with current resolution estimates are all stored in the `_optimiser.star` and `_model.star` files, but may also be extracted from the stdout file. For example, try:

```
grep Auto Refine3D/first3dref/run.out
```

7 Mask creation & Postprocessing

After performing a 3D auto-refinement, the map needs to be sharpened. Also, the gold-standard FSC curves inside the auto-refine procedures only use unmasked maps (unless you've used the option `Use solvent-flattened FSCs`). This means that the actual resolution is under-estimated during the actual refinement, because noise in the solvent region will lower the FSC curve. RELION's procedure for B-factor sharpening and calculating masked FSC curves [3] is called "post-processing". First however, we'll need to make a mask to define where the protein ends and the solvent region starts. This is done using the Mask Creation job-type.

7.1 Making a mask

On the I/O tab, select the output map from the finished 3D auto-refine job: `Refine3D/first3dref/run_class001.mrc`. On the Mask tab set:

- `Lowpass filter map (A):` `15`

  (A 15Å low-pass filter seems to be a good choice for smooth solvent masks for many proteins.)

- `Pixel size (A):` `-1`

  (This value will be taken automatically from the header of the input map.)

- `Initial binarisation threshold:` `0.005`

  (This should be a threshold at which rendering of the low-pass filtered map in for example CHIMERA shows absolutely no noisy spots outside the protein area. Move the threshold up and down to find a suitable spot. Remember that one can use the command-line program called `relion_image_handler` with the options `--lowpass 15 --angpix 1.244` to get a low-pass filtered version of an input map. Often good values for the initial threshold are around 0.01-0.04.)

- `Extend binary map this many pixels:` `0`

  (The threshold above is used to generate a black-and-white mask. The white volume in this map will be grown this many pixels in all directions. Use this to make your initial binary mask less tight.)

- `Add a soft-edge of this many pixels:` `6`

  (This will put a cosine-shaped soft edge on your masks. This is important, as the correction procedure that measures the effect of the mask on the FSC curve may be quite sensitive to too sharp masks. As the mask generation is relatively quick, we often play with the mask parameters to get the best resolution estimate.)

Ignore the Helix tab and use an alias like `first3dref`. Note that you can run the mask_create program with multiple threads to accelerate this step. You can look at slices through the resulting mask using the Display: button, or you can load the mask into UCSF CHIMERA. The latter may be a good idea, together with the map from the auto-refine procedure, to confirm that the masks encapsulates the entire structure, but does not leave a lot of solvent inside the mask. You can continue the same job with new settings for the mask generation until you have found a mask you like.

7.2 Postprocessing

Now select the Post-processing job-type, and on the I/O tab, set:

- `One of the 2 unfiltered half-maps:` `Refine3D/first3dref/run_half1_class001_unfil.mrc`
- `Solvent mask:` `MaskCreate/first3dref/mask.mrc`

- `Calibrated pixel size (A):` `1.244`

  (Sometimes you find out when you start building a model that what you thought was the correct pixel size, in fact was off by several percent. Inside RELION, everything up until this point was still consistent. so you do not need to re-refine your map and/or re-classify your data. All you need to do is provide the correct pixel size here for your correct map and final resolution estimation.)

On the Sharpen tab, set:

- `Estimate B-factor automatically:` `Yes`

  (This procedure is based on the classic Rosenthal and Henderson paper [11], and will need the final resolution to extend significantly beyond 10 Å. If your map does not reach that resolution, you may want to use your own "ad-hoc" B-factor instead.)

- `Lowest resolution for auto-B fit (A):` `10`

  (This is usually not changed.)

- `Use your own B-factor?` `No`

- `Perform MTF correction?` `No`

  (As we provided an MTF file when we imported the movies, MTF correction has already been performed inside the refinement.)

On the Filter tab, set:

- `Skip FSC-weighting?` `No`

  (This option is sometimes useful to analyse regions of the map in which the resolution extends *beyond* the overall resolution of the map. This is not the case now.)

Run the job (no need for a cluster, as this job will run very quickly) and use an alias like `first3dref`. Using the Display button, you can display slizes through the postprocessed map and a PDF with the FSC curves and the Guinier plots for this structure. You can also open the `PostProcess/first3dref/postprocess.mrc` map in CHIMERA, where you will see that it is much easier to see where all the alpha-helices are then in the converged map of the 3D auto-refine procedure. The resolution estimate is based on the phase-randomization procedure as published previously [3]. Make sure that the FSC of the phase-randomized maps (the red curve) is more-or-less zero at the estimated resolution of the postprocessed map. If it is not, then your mask is too sharp or has too many details. In that case use a stronger low-pass filter and/or a wider and more softer mask in the Mask creation step above, and repeat the postprocessing.

8 CTF and aberration refinement

Next, we'll use the CTF refinement job-type to estimate the asymmetrical and symmetrical aberrations in the dataset; whether there is any anisotropic magnification; and we'll re-estimate per-particle defocus values for the entire data set. Running this job-type can lead to further improvements in resolution at a relatively minor computational cost, but it all depends on how flat your ice was (for per-particle defocus estimates), and how well you had aligned your scope (for the aberrations). It runs from a previous 3D auto-refine job as well as a corresponding Post-processing job. Let's start with the higher-order aberrations, to see whether this data suffered from beamtilt or trefoil (which are asymmetric aberrations), or from tetrafoil or an error in spherical aberration (which are symmetric aberrations).

8.1 Higher-order aberrations

On the I/O tab of CTF refinement job-type on the GUI the set:

- `Particles (from Refine3D)` `Refine3D/first3dref/run_data.star`
- `Postprocess STAR file:` `PostProcess/first3dref/postprocess.star`

On the Fit tab set:

- `Estimate (anisotropic magnification` `No`

  (We will do this later, see below.)

- `Perform CTF parameter fitting?` `No`

  (We will do this later, see below.)

- `Estimate beamtilt?` `Yes`

  (Despite the observation that most microscopists perform coma-free alignment schemes prior to data acquisition, there are still many data sets with a significant amount of beamtilt. That's why in this example we're first looking for beamtilt, and do the anisotropic magnification and the per-particle defocus later. In general, one would try to first estimate the source of the largest errors.)

- `Also estimate trefoil?` `Yes`

  (This will allow more higher-order Zernike polynomials in the fitting of the asymmetric aberrations.)

- `estimate 4th order aberrations?` `Yes`

  (This is done mostly for illustrative purposes here. One would not expect a big improvement at the current resolution of 3 Å.)

- `Minimum resolution for fits (A):` `30`

  (Just leave the default.)

This program is only implemented on the CPU. Using 1 MPI and 12 threads, on our computer, this job finished in approximately one minute. We used the alias `aberrations`.

You can analyse the accumulated averages for the asymmetrical and symmetrical aberrations, as well as their models, by selecting the `logfile.pdf` file from the Display: button on the GUI. You'll see that this data actually suffered from some beamtilt: one side of the asymmetrical aberration images is blue, whereas the other side is red. You can find the values (approximately -0.2 mrad of beamtilt in the Y-direction) in the optics table of the output STAR file:

```
less CtfRefine/aberrations/particles_ctf_refine.star
```

There was also a small error in the spherical aberration, as the symmetrical aberration image shows a significant, circularly symmetric difference (the image is blue at higher spatial frequencies, i.e. away from the center of the image). Importantly, for both the asymmetric and the symmetric aberaations, the model seems to capture the aberrations well.

If the data had suffered from trefoil, then the asymmetric aberration plot would have shown 3-fold symmetric blue/red deviations. If the data had suffered from tetrafoil, then the symmetric aberration plot would have shown 4-fold symmetric blue/red deviations. Examples of those are shown in the supplement of our 2019 publication on Tau filaments from the brain of individuals with chronic traumatic encephalopathy (CTE).

8.2 Anisotropic magnification

Next, let's see whether these data suffer from anisotropic magnification. On the I/O tab of CTF refinement job-type on the GUI, use the output from the previous CTF refinement job as input to this one:

- `Particles (from Refine3D)` `CtfRefine/aberrations/particles_ctf_refine.star`
- `Postprocess STAR file:` `PostProcess/first3dref/postprocess.star`

And this time, on the Fit tab set:

- `Estimate (anisotropic magnification` `Yes`

  (This will deactivate most of the other options, as simultaneous magnification and aberration refinement is unstable.)

- `Minimum resolution for fits (A):` `30`

  (Just leave the default.)

Using 1 MPI and 12 threads, on our computer, this job finished in approximately one minute. We used the alias `magnification`.

Again, the relevant images to analyse are in the `logfile.pdf`. There seem to be some blue-red trends, but the actual anisotropy is very small, as assessed from the `_rlnMagMat??` elements of the (2x2) transformation matrix in the optics table of the output STAR file:

```
less CtfRefine/magnification/particles_ctf_refine.star
```

8.3 Per-particle defocus values

Lastly, let's re-estimate the defocus values for each particle. Again, use the output from the previous job as input for this one (although we could have just as well kept using the output from the aberration correction, as the magnification anisotropy was very small):

- `Particles (from Refine3D)` `CtfRefine/magnification/particles_ctf_refine.star`
- `Postprocess STAR file:` `PostProcess/first3dref/postprocess.star`

And this time, on the Fit tab set:

- `Estimate (anisotropic magnification` `No`
- `Perform CTF parameter fitting?` `Yes`
- `Fit defocus?` `Per-particle`

  (Provided the resolution of the reference extends well beyond 4 Å, per-particle defocus estimation seems to be relatively stable. It will account for non-horizontal ice layers, and particles at the top or bottom of the ice layer.)
- `Fit astigmatism?` `Per-micrograph`

  (Provided the resolution of the reference extends well beyond 4 Å, and there are enough particles on each micrograph, estimating astigmatism on a per-micrograph basis seems to be relatively stable. Doing this on a pre-particle basis would require particles with very strong signal.)
- `Fit B-factor?` `No`
- `Fit phase-shift?` `No`

  (This is useful for phase-plate data.)
- `Estimate beamtilt?` `No`
- `estimate 4th order aberrations?` `No`

- `Minimum resolution for fits (A):` `30`

  (Just leave the default.)

Using 1 MPI and 12 threads, on our computer, this job finished in six minutes. We used the alias `defocus`.

Per-particle defocus values are plotted by colour for each micrograph in the `logfile.pdf`. Can you spot micrographs with a tilted ice layer?

It is probably a good idea to re-run 3D auto-refine and Post-processing at this stage, so we can confirm that the new particle STAR file actually gives better results. We used the alias `ctfrefined` for both runs, and the resolution improved (a bit): from 3.03 Å to 2.97 Å.

9 Bayesian polishing

RELION also implements a Bayesian approach to per-particle, reference-based beam-induced motion correction. This approachs aims to optimise a regularised likelihood, which allows us to associate with each hypothetical set of particle trajectories a prior likelihood that favors spatially coherent and temporally smooth motion without imposing any hard constraints. The smoothness prior term requires three parameters that describe the statistics of the observed motion. To estimate the prior that yields the best motion tracks for this particular data set, we can first run the program in 'training mode'. Once the estimates have been obtained, one can then run the program again to fit tracks for the motion of all particles in the data set and to produce adequately weighted averages of the aligned movie frames.

9.1 Running in training mode

Using 16 threads in parallel, this job took 1 hour and 15 minutes on our computer. If you do not want to wait for this, you can just proceed to section 9.2 and use the sigma-values from our precalculated results, which are already given in that section.

If you do want to run this job yourself, on the I/O tab of the Bayesian polishing job-type set:

- `Micrographs (from MotionCorr):` `MotionCorr/relioncor2/corrected_micrographs.star`

  (It is important that this Motion correction job has been run in RELION-3.0, or above. Motion correction jobs run RELION-2.1 or below will NOT work, as required metadata about the motion correction is not written out.

- `Particles (from Refine3D or CtfRefine):` `Refine3D/ctfrefined/particles_ctf_refine.star`

  (These particles will be polished)

- `Postprocess STAR file` `PostProcess/ctfrefined/postprocess.star`

  (the mask and FSC curve from this job will be used in the polishing prceodure.)

- `First movie frame:` `1`

- `Last movie frame:` `-1`

  (Some people throw away the first or last frames from their movies. Note that this is **not recommended** when performing Bayesian polishing in RELION. The B-factor weighting of the movie frames will automatically optimise the signal-to-noise ratio in the shiny particles, so it is best to include all movie frames.)

On the Train tab set:

- `Train optimal parameters?` `Yes`

- `Fraction of Fourier pixels for testing:` `0.5`

  (Just leave the default here)

- `Use this many particles:` `4000`

  (That's almost all we have anyway. Note that the more particles, the more RAM this program will take. If you run out of memory, try training with fewer particles. Using much fewer than 4000 particles is not recommended.)

On the Polish tab make sure you set:

- `Perform particle polishing?` `No`

Note that the training step of this program has not been MPI-parallelised. Therefore, make sure you use only a single MPI process. We ran the program with 16 threads to speed it up. Still, the calculation took more than 1 hour. We used an alias of `train`.

9.2 Running in polishing mode

Once the training step is finished, the program will write out a text file called `Polish/train/opt_params.txt`. To use these parameters to polish your particles, click on the job-type menu on the left to select a new Bayesian polishing job. Keep the parameters on the I/O tab the same as before, and on the Train tab, make sure you switch the training off. Then, on the Polish tab set:

- `Perform particle polishing?` `Yes`

- `Optimised parameter file:` `Polish/train/opt_params.txt`
- `OR use your own parameters?` `No`
- `Minimum resolution for B-factor fit (A):` `20`
- `Maximum resolution for B-factor fit (A):` `-1`

  (just leave the defaults for these last two parameters)

Alternatively, if you decided to skip the training set, then you can fill in the Polish tab with the sigma-parameters that we obtained in our run:

- `Perform particle polishing?` `Yes`
- `Optimised parameter file:`

  (leave this empty to use the optimal parameters we got as per below.)
- `OR use your own parameters?` `Yes`
- `Sigma for velocity (A/dose)` `0.42`
- `Sigma for divergence (A)` `1600`
- `Sigma for acceleration (A/dose)` `2.61`
- `Minimum resolution for B-factor fit (A):` `20`
- `Maximum resolution for B-factor fit (A):` `-1`

  (just leave the defaults for these last two parameters)

This part of the program is MPI-parallelised. Using 3 MPI processes, each with 16 threads, our run finished in two minutes. We used an alias of `polish`.

9.3 Analysing the results

The Bayesian polishing job outputs a STAR file with the polished particles called `shiny.star` and a PDF logfile. The latter contains plots of the scale and B-factors used for the radiation-damage weighting, plus plots of the refined particle tracks for all included particles on all micrographs. Looking at the plots for this data set, it appeared that the stage was a bit drifty: almost all particles move from the top right to the bottom left during the movies.

After polishing, the signal-to-noise ratio in the particles has improved, and one should submit a new 3D auto-refine job and a corrsponding Post-processing job. We chose to run the 3D auto-refine job with the shiny particles using the following option on the I/O tab:

- `Reference mask (optional):` `MaskCreate/first3dref/mask.mrc`

(this is the mask we made for the first Post-processing job. Using this option, the solvent will be set to zero for all pixels outside the mask. This reduces noise in the reference, and thus lead to better orientation assignments and thus reconstructions.)

and this option on the Optmisation tab:

- `Use solvent-flattened FSCs?` `Yes`

  (Using this option, the refinement will use a solvent-correction on the gold-standard FSC curve at every iteration, very much like the one used in the Post-processing job-type. This option is particularly useful when the protein occupies a relatively small volume inside the particle box, e.g. with very elongated molecules, or when one focusses refinement on a small part using a mask. The default way of calculating FSCs in the 3D auto-refinement is without masking the (gold-standard) half-maps, which systematically under-estimates the resolution during refinement. This is remediated by calculating phase-randomised solvent-corrected FSC curves at every iteration, and this generally leads to a noticeable improvement in resolution.)

As you can see in the pre-calculated results, we obtained a final resolution just beyond 2.8 Å. Not bad for 3GB of data, right?

9.4 When and how to run CTF refinement and Bayesian polishing

Both Bayesian polishing and CTF refinement, which comprises per-particle defocus, magnification and higher-order aberration estimation, may improve the resolution of the reconstruction. This raises a question of which one to apply first. In this example, we first refined the aberrations, the magnification, and then the per-particle defocus values. We then followed up with polishing, but we could have also performed the polishing before any of the CTF refinements. Both approaches benefit from higher resolution models, so an iterative procedure may be beneficial. For example, one could repeat the CTF refinement after the Bayesian polishing. In general, it is probably best to tackle the biggest problem first, and some trial and error may be necessary.

Moreover, we have seen for some cases that the training prodcedure of Bayesian polishing yields inconsistent results: i.e. multiple runs yield very different sigma values. However, we have also observed that often the actual sigma values used for the polishing do not matter much for the resolution of the map after re-refining the shiny particles. Therefore, and also because the training is computationally expensive, it may be just as well to run the polishing directly with the default parameters ($\sigma_{\text{vel}} = 0.2; \sigma_{\text{div}} = 5000; \sigma_{\text{acc}} = 2$), i.e. without training for your specific data set.

10 Local-resolution estimation

The estimated resolution from the post-processing program is a global estimate. However, a single number cannot describe the variations in resolution that are often observed in reconstructions of macromolecular complexes. Alp Kucukelbir and Hemant Tagare wrote a nifty program to estimate the variation in resolution throughout the map [8]. RELION implements a wrapper to this program through the Local resolution job-type. Alternatively, one can choose to run a post-processing-like procedure with a soft spherical mask that is moved around the entire map. In the example below, we use the latter.

10.1 Running the job

On the I/O tab set:

- `One of the two unfiltered half-maps:` `Refine3D/shiny/run_half1_class001_unfil.mrc`
- `User-provided solvent mask:` `MaskCreate/first3dref/mask.mrc`
- `Calibrated pixel size:` `1.244`

  (Sometimes you find out when you start building a model that what you thought was the correct pixel size, in fact was off by several percent. Inside RELION, everything up until this point was still consistent. so you do not need to re-refine your map and/or re-classify your data. All you need to do is provide the correct pixel size here for your correct map and final resolution estimation.)

On the ResMap tab set `Use ResMap?` to `No`; on the Relion tab set:

- `Use Relion?` `Yes`
- `User-provided B-factor:` `-30`

  (This value will be used to also calculate a locally-filtered and sharpened map. Probably you want to use a value close to the one determined automatically during the Post-processing job.)

- `MTF of the detector (STAR file):`

  (The same as for the Post-processing job.)

10.2 Analysing the results

We used `shiny` as an alias. Running with 8 MPI processes, this job took approximately 7 minutes. The output is a file called `LocalRes/polished/relion_locres.mrc` that may be used in UCSF CHIMERA to color the `Postprocess/polished/postprocess.mrc`

map according to local resolution. This is done using `Tools -> Volume data -> Surface color`, and then select `by volume data value` and browse to the `resmap` file.

Unique to the RELION option is the additional output of a locally-filtered (and sharpened map), which may be useful to describe the overall variations in map quality in a single map. This map is saved as `LocalRes/polished/relion_locres_filtered.mrc` and can be visualised directly in UCSF CHIMERA (and optionally also coloured by local resolution as before).

11 Checking the handedness

Careful inspection of the map may indicate that the handedness is incorrect, e.g. because the α-helices turn the wrong way around. Remember that it is impossible to determine absolute handedness from a data set without tilting the microscopy stage. The SGD algorithm in the 3D initial model jobtype therefore has a 50% chance of being in the opposite hand. In our precalculated results, this was not the case. One may flip the handedness of the postprocessed map as follows:

```
relion_image_handler --i PostProcess/polished/postprocess.mrc  \
  --o PostProcess/polished/postprocess_invert.mrc --invert_hand
```

The same command could also be run on any of the other maps. If one realises earlier on in the image processing procedure that the hand is wrong, one could of course also switch to the other hand earlier on. For RELION itself it doesn't matter, as both hands cannot be distinguished, but it may be more convenient to flip the hand as soon as you notice it.

Once in the correct hand, you might want to load the map into UCSF CHIMERA and superimpose it with an atomic model for β-galactosidase. You could try fetching one straight from the PDB using PDB-ID 5a1a.

12 Wrapping up

12.1 Making a flowchart

Do you wonder how you got to your final reconstruction? Select the last job you performed from the Finished jobs list and try the `Make flowchart` option from the Job actions button. You'll need LaTeX and the `TikZ` package on your system in order for this to work. On the first page will be an overview flowchart without the exact job names, which may be useful for publication purposes (perhaps after editing it in your favourite vector-based design program). After the overview flowchart, the first detailed flowchart shows you the path how you

got to this end. Note that flowcharts longer than 10 steps will be cut into pieces. There may be branches in your work flow. Therefore, following the flowchart of your last job, there will also be flowcharts for each branch. You can click on the links to get to the corresponding position in the PDF file.

12.2 Cleaning up your directories

In order to save disk space, RELION has an option to clean up job directories. There are two modes of cleaning: 'gentle' cleaning will only delete intermediate files from the job directory being cleaned; 'harsh' cleaning also deletes files that may be necessary to launch a new job that needs input from the job being cleaned. For example, harsh cleaning will remove averaged micrographs from a MotionCorr job, or extracted particles stacks from a Particle extraction job, while gentle cleaning will remove all files from itermediate iterations of 2D classification, 3D classification or 3D auto-refine jobs. You can clean individual jobs from the Job actions button; or you can clean all jobs from the 'Jobs' pull-down menu at the top of the GUI. We used the 'Gently clean all jobs' option from that menu before making a tarball of the project directory that we distributed as our precalculated results. You might want to gently clean your project directory before you put your data in long-term storage.

12.3 Asking questions and citing us

That's it! Hopefully you enjoyed this tutorial and found it useful. If you have any questions about RELION, please first check the FAQ on the RELION Wiki and the CCPEM mailing list. If that doesn't help, use the CCPEM list for asking your question. *Please, please, please, do not send a direct email to Sjors, as he can no longer respond to all of those.*

If RELION turns out to be useful in your research, please do cite our papers and tell your colleagues about it.

12.4 Further reading

The theory behind the refinement procedure in RELION is described in detail in:

- S.H.W. Scheres (2012) "RELION: Implementation of a Bayesian approach to cryo-EM structure determination" *J. Struc. Biol.*, 180, 519-530.
- S.H.W. Scheres (2012) "A Bayesian view on cryo-EM structure determination" *J. Mol. Biol.*, 415, 406-418.

A comprehensive overview of how to use RELION for all types of classifications is described in:

- S.H.W. Scheres (2016) "Processing of structurally heterogeneous cryo-EM data in RELION" *Meth. Enzym.*, 579, 125-157.

This tutorial does not cover multi-body refinement, which is useful to describe continuous motions in relatively large complexes. You can find a manuscript with specific instructions on how to perform multi-body refinement on the RELION Wiki.

13 Appendix A: notes on installation

13.1 Install MPI

Note that you'll need a computing cluster (or a multi-core desktop machine with NVIDIA GPUs) with an MPI (message passing interface) installation. To compile RELION, you'll need a mpi-devel package. The exact flavour (openMPI, MPICH, LAM-MPI, etc) or version will probably not matter much. If you don't have an mpi-devel installation already on your system, we recommend installing openMPI.

13.2 Install CUDA

If you have a relatively modern GPU from NVIDIA (with compute capability 3.5+), then you can accelerate your autopicking, classification and refinement jobs considerably. In order to compile RELION with GPU-acceleration support, you'll need to install CUDA. We used CUDA-8.0 to prepare this tutorial. Download it from NVIDIA's website.

13.3 Install RELION

RELION is open-source software. Download it for free from the RELION wiki, and follow the installation instructions. If you're not familiar with your job submission system (e.g. Sun Grid Engine, PBS/TORQUE, etc), then ask your system administrator for help in setting up the qsub.csh script as explained in the installation instructions. Note that you will probably want to run so-called hybridly-parallel jobs, i.e. calculations that use both MPI for distributed-memory parallelization AND pthreads for shared-memory parallelization. Your job submission queueing system may require some tweaking to allow this. Again, ask your sysadmin for assistance.

13.4 Install motion-correction software

RELION-3.0 provides a wrapper to the UCSF program MOTIONCOR2, which is used for whole-frame micrograph movie-alignment [20]. Download the program from David Agard's page and follow his installation instructions. Alternatively, you may also use RELION's own (CPU-only) implementation of MOTIONCOR2, so don't worry if you have trouble installing the UCSF implementation. Note that, as of version 3.0, the wrapper to UNBLUR [5] from Niko grigorieff's group has been discontinued from the GUI.

13.5 Install CTF-estimation software

CTF estimation is not part of RELION. Instead, RELION provides a wrapper to Alexis Rohou and Niko Grigorieff's CTFFIND4 [10]. Please download this from Niko's CTFFIND website and follow his installation instructions. Alternatively, if you have NVIDIA graphics cards (GPUs) in your machine, you may also use Kai Zhang's GCTF [19], which may be downloaded from Kai's website at LMB.

13.6 Install RESMAP

Local-resolution estimation may be performed inside RELION's own postprocessing program, or through a wrapper to Alp Kucukelbir's RESMAP [8]. Please download it from Alp's RESMAP website and follow his installation instructions.

14 Appendix B: using RELION

14.1 The GUI

14.1.1 A pipeline approach

The GUI serves a central role in it's pipelined approach, details of which have been published in the 2016 Proceedings of the CCP-EM Spring Symposium [4]. We recommend to create a single directory per project, i.e. per structure you want to determine. We call this the project directory. It is important to always launch the relion graphical user-interface (GUI), by typing the command `relion`, from the project directory.

The GUI keeps track of all jobs and how output from one job is used as input for another, thereby forming a workflow or pipeline. Each type of job has its own output directory, e.g. `Class2D/`, and inside these job-type directories, new jobs get consecutive numbers, e.g. `Class2D/job010`. Inside these individual job directories, output names are fixed, e.g. `Class2D/job010/run`. To provide a mechanism to have more meaningful names for jobs, a system of job "aliases" is used, which are implemented as symbolic links to the individual job directories on the filesystem. All info about the pipeline is stored in a file called `default_pipeline.star`, but in normal circumstances the user does not need to look into this file. In case this file gets corrupted, one can copy back a backup of this file from the last executed job directory.

14.1.2 The upper half: jobtype-browser and parameter-panel

On the left of the upper half of the GUI is the jobtype-browser: a vertical list of jobtypes, e.g. 2D classification. On the right is a panel with multiple tabs, where parameters to the different types of jobs may be input. On the top left of the GUI are three different menu's, providing a range of functionalities. The Schedule and Run! buttons can be used to schedule jobs for future execution, or to execute them now. The former is particularly useful in preparing fully automated "pipelines" that can be run iteratively, for example in real-time as data is being collected. See section **??** for more details. By clicking in the jobtype-browser on the left-hand side of the GUI, a new job (with a Run! button) will be loaded in the parameter-panel on the right.

14.1.3 The lower half: job-lists and stdout/stderr windows

The lower half of the GUI contains lists of jobs that are still running (Running jobs), have already finished (Finished jobs), or are scheduled for later execution (Scheduled jobs). By clicking jobs in these lists, the parameters of that job will

be loaded in the parameter-panel, and the Run! button will change color and turn into continue now!. Upon clicking the latter, no new output job-directory will be made, but the job will be continued according to the parameters given in the parameter-panel. 2D classification, 3D classifications and 3D auto-refine jobs will need a `_optimiser.star` file to continue from, and will have filenames with the iteration from which they were continued, e.g. `run_ct23`. Other types of jobs may continue from the point until they were executed before, e.g. Motion correction, CTF estimation, Auto-picking and Particle Extraction will continue by running only on those micrographs that weren't done before. The Input to this job and Output from this job lists link jobs together and can be used to browse backwards or forwards in the project history.

At the bottom of the lower half of the GUI, the standard output (stdout) and standard error (stderr) of the selected (finished or running) job will show in black and red text, respectively. The stderr should ideally be empty, any text here is usually worth inspection. These text displays get updated every time you click on a job in the job-lists. Double-clicking on the stdout or stderr displays will open a pop-up window with the entire text for more convenient scrolling.

14.1.4 The Display button

The Display: button below the run and schedule buttons serves to visualise the most important input and output files for each job. When a job from the job-lists in the lower half of the GUI is selected, clicking this button will pop-up a menu with all the input and output of this job that can be displayed (for example, particles, micrographs, coordinates, PDF files, etc). A more general functionality to display any (e.g. intermediate) file can be accessed through the `Display` option of the `File` menu on the top left of the GUI.

14.1.5 The Job actions button

The Job actions button opens up a little menu with options for the selected (running, finished or scheduled) job. Here, you can access a file called `note.txt` (that is saved in every individual job directory and which may be used to store user comments); you can change the alias of the job; you can mark a job as finished (in case it somehow got stuck); you can make a flowchart of the history of that job (provided LaTeX and the `TikZ` package are installed on your system, also see section 12); or you can delete or clean a job to save disk space (see below).

14.1.6 Clean-up to save disk space

Deletion of jobs moves the entire job directory from the project directory into a directory called `Trash/`. You can empty the Trash folder from `File` menu on

the top left of the GUI to really free up the space. Until you do so, you can still "undelete" jobs using the corresponding option from the `Jobs` menu on the top left.

To save disk space, you can also "clean" jobs, which will move intermediate files to the Trash folder, e.g. the files written out for all intermediate iterations of refine jobs. There are two cleaning options: `gentle clean` will leave all files intact that could be used as input into another job, while `harsh clean` may also remove those. Evidently, "harsh" cleaning can free up more space, in particular directories with particle stacks or micrographs may become large, e.g. from Motion correction, Particle extraction, Movie refinement and Particle polishing job types. One can also clean all directories in the project with a single click using the corresponding options from the `Jobs` menu on the top left of the GUI. You can protect specific directories from "harsh" cleaning by placing a file called `NO_HARSH_CLEAN` inside them, e.g. you may want to protect your final set of polished particles from deletion by executing:

```
touch Polish/job098/NO_HARSH_CLEAN
```

14.2 Optimise computations for your setup

14.2.1 GPU-acceleration

Dari Kimanius and Bjoern Forsberg from the group of Erik Lindahl (Stockholm) have ported the most computationally expensive parts of RELION for the use of GPUs. Because they used the CUDA-libraries from NVIDIA to do this, GPU-acceleration in RELION only works with NVIDIA cards. These need to be of compute capability 3.5 or higher. Both single and double precision cards will work, so one is not restricted to the expensive double-precision cards, but can use the cheaper gaming cards as well. Details of their implementation can be found in their eLife paper[7].

Two different relion programs have been GPU-accelerated: `relion_autopick` (for Auto-picking) and `relion_refine` (for 2D classification, 3D classification and 3D auto-refine jobs). Both the sequential and the MPI-versions of these programs have been accelerated.

14.2.2 Disk access

With the much improved speed of image processing provided by the GPU-acceleration, access to the hard disk increasingly becomes a bottle neck. Several options are available on the RELION GUI to optimise disk access for your data set and computer setup. For 2D classification, 3D initial model, 3D classification and 3D auto-refine one can choose to `use parallel disc I/O`. When set to `Yes`, all MPI processes will read particles simultaneously from the hard disk.

Otherwise, only the master will read images and send them through the network to the slaves. Parallel file systems like gluster of fhgfs are good at parallel disc I/O. NFS may break with many slaves reading in parallel.

One can also set the `number of pooled particles`. Particles are processed in individual batches by MPI slaves. During each batch, a stack of particle images is only opened and closed once to improve disk access times. All particle images of a single batch are read into memory together. The size of these batches is at least one particle per thread used. This parameter controls how many particles are read together in a batch by each thread. If it is set to 3 and one uses 8 threads, batches of 3x8=24 particles will be read together. This may improve performance on systems where disk access, and particularly metadata handling of disk access, is a problem. It has a modest cost of increased RAM usage.

If one has a relatively small data set (and/or a computer with a lot of RAM), then one can `pre-read all particles into RAM` at the beginning of a calculation. This will greatly speed up calculations on systems with relatively slow disk access. However, one should of course be careful with the amount of RAM available. Because particles are read in float-precision, it will take $\frac{N \times boxsize \times boxsize \times 4}{1024 \times 1024 \times 1024}$ Giga-bytes to read N particles into RAM. For 100,000 particles with a 200-pixel boxsize that becomes 15Gb, or 60 Gb for the same number of particles in a 400-pixel boxsize.

If the data set is too large to pre-read into RAM, but each computing node has a local, fast disk (e.g. a solid-state drive) mounted with the same name, then one can let each MPI slave copy all particles onto the local disk prior to starting the calculations. This is done using the `Copy particles to scratch directory`. If multiple slaves will be executed on the same node, only the first slave will copy the particles. If the local disk is too small to hold the entire data set, those particles that no loner fit on the scratch disk will be read from their original position. A sub-directory called `relion_volatile` will be created inside the specified directory name. For example, if one specifies `/ssd`, then a directory called `/ssd/relion_volatile` will be created. If the `/ssd/relion_volatile` directory already exists, it will be wiped before copying the particles. Then, the program will copy all input particles into a single large stack inside this directory. If the job finishes correctly, the `/ssd/relion_volatile` directory will be deleted again. If the job crashes before finishing, you may want to remove it yourself. The program will create the `/ssd/relion_volatile` directory with writing permissions for everyone. Thereby, one can choose to use `/ssd`, i.e. without a username, as a scratch directory. That way, provided always only a single job is executed by a single user on each computing node, the local disks do not run the risk of filling up with junk when jobs crash and users forget to clean the scratch disk themselves.

Finally, there is an option to `combine iterations through disc`. If set to `Yes`, at the end of every iteration all MPI slaves will write out a large file with their accumulated results. The MPI master will read in all these files, combine them all, and write out a new file with the combined results. All MPI salves

will then read in the combined results. This reduces heavy load on the network, but increases load on the disc I/O. This will affect the time it takes between the progress-bar in the expectation step reaching its end (the mouse gets to the cheese) and the start of the ensuing maximisation step. It will depend on your system setup which is most efficient. This option was originally implemented to circumvent bugs on the network cards on our old cluster at LMB. Nowadays, we prefer not to use this option, as it tends to be very slow when refinements reached high resolutions.

14.3 Interaction with other programs

Although, in principle, RELION can use particles that have been extracted by a different program, this is NOT the recommended procedure. Many programs change the particles themselves, e.g. through phase flipping, band-pass or Wiener filtering, masking etc. All these are sub-optimal for subsequent use in RELION. Moreover, gathering all the required metadata into a correctly formatted RELION-type STAR file may be prone to errors. Because re-extracting your particles in RELION is straightforward and very fast, the procedure outlined below is often a much easier (and better) route into RELION.

Also, several implementations of wrappers around RELION have now been reported (e.g. in EMAN2, SCIPION and APPION). Although we try to be helpful when others write these wrappers, we have absolutely no control over them and do not know whether their final product uses RELION in the best way. Therefore, in case of any doubt regarding results obtained with these wrappers, we would recommend following the procedures outlined in this tutorial. The recommended way of executing external programs from within the RELION pipeline itself is outlined in the next section.

14.4 The External job-type

14.4.1 User interaction through the GUI

The External job-type on the RELION-3.1 GUI provides a way to execute any third-party program from within the RELION pipeline. The interaction with the user is as follows:

On the Input tab set:

- `External executable:` `myscript.py`

  (This is the filename of an executable script, which will call the external program.)

- `Input movies:`
- `Input micrographs:`

- `Input particles:`
- `Input coordinates:`
- `Input 3D reference:`
- `Input 3D mask:`

The user provides at least one of the input entries to tell RELION from which other jobs the input nodes come from, and what type of input this is. This will therefore allow to maintain an intact directional graph inside the pipeliner. On the Params tab, the user can then provide up to ten (optional) free parameters that will be passed onto the executable script. Finally, the Running tab allows multi-threaded execution, queue submission, and any other arguments to be passed through the `Additional arguments` entry.

The GUI will then create and execute the following command, either locally or through a queueing system:

```
myscript.py --o External/jobXXX/ --in_YYY ZZZ --LABELN VALUEN --j J
```

where

- `XXX` is the current jobnumber in the pipeline.
- `YYY` is the type of the input node: `movies`, `mics`, `parts`, `coords`, `3dref`, or `mask`,
- `ZZZ` is the name of the corresponding input node. Note that more than one input node may be given, each with its own `--in_YYY` argument.
- `LABELN` is the label of a free parameter, as defined on the Params tab of the GUI. Note that up to ten different labels may be used.
- `VALUEN` is the corresponding value of the free parameter. This is optional: not every label needs a value.
- `J` is the number of threads defined on the running tab.

14.4.2 Functionality of the executable script

It is the responsibility of the executable script (`myscript.py`) to handle the command line parsing of the generated command. In addition, there are a few rules the script needs to adhere to:

- All output needs to be written out in the output directory, as specified by the `--o External/jobXXX/` option. In addition, for jobs that emulate RELION job-types like Motion correction or Auto-picking, the output should be organised in the same directory structure as the corresponding RELION job-type would make.

- When completed, the script should create an empty file called `RELION_JOB_EXIT_SUCCESS` in the output directory. This will tell the pipeliner that the task has finished successfully. Aborted or failed runs may optionally be communicated by creating files called `RELION_JOB_EXIT_ABORTED` and `RELION_JOB_EXIT_FAILURE`.
- The job should output file called `RELION_OUTPUT_NODES.star`, which should have a table called `data_output_nodes`. This table should have two columns with the names and types of all the output nodes of the job, using the `_rlnPipeLineNodeName` and `_rlnPipeLineNodeType` metadata labels. See the `data_pipeline_nodes` table in the `default_pipeline.star` file of any RELION project directory for examples. Output nodes defined here will lead to the creation of edges between jobs in the directional graph of the pipeliner; and output nodes will be available for convenient displaying by the user using the Display: button on the GUI.
- The following may not be necessary or relevant, but the GUI has a Job actions button, which allows users to abort running jobs. This button will create a file called `RELION_JOB_ABORT_NOW` in the output directory. If this functionality is to be used, the script should abort the job when this file is present, create the `RELION_JOB_EXIT_ABORTED` file, remove the `RELION_JOB_ABORT_NOW` file, and then exit.

14.4.3 Example: a particle-picker

If your external program is a particle picker, e.g. `topaz` [1] (and see here arxiv.pdf for the preprint), then you would give on the Input tab:

- `External executable:` `run_topaz.py`
- `Input micrographs:` `CtfFind/job005/micrographs_ctf.star`

  (This file would be visible through the Browse button next to the input entry, which would only show star files of micrographs that exist in the current project.)

On the Params tab, one would provide any necessary arguments to be picked up by the script, for example:

- `Param1 label, value:` `threshold 0.1`
- `Param2 label, value:` `denoise_first`

Upon pressing the Run! button, this would execute the following command:

```
run_topaz.py --o External/job006/ --in_mics CtfFind/job005/micrographs_ctf.star \\
  --threshold 0.1 --denoise_first --j 1
```

The executable `run_topaz.py` is then responsible for correctly passing the command line arguments to `topaz`, and to make sure the rules in the previous section are adhered to. For picking jobs, the directory structure of the input movies

(or micrographs) should be maintained inside the output directory, and each micrograph would have a STAR file with the picked coordinates that has the same rootname as the original micrograph, but with a `_PICKNAME.star` suffix. The `PICKNAME` is a free string. One could use the name of the particle-picking program, for example `topaz`. Therefore, if a movie was originally imported as `Movies/mic001.tif`, its corresponding STAR file with the picked coordinate would be placed in `External/job006/Movies/mic001_topaz.star`. In addition, in the output directory, the script should create a text file called `coords_suffix_PICKNAME.star` (i.e. `coords_suffix_topaz.star`). This file should contain at least one line of text, which is the name of the input micrographs STAR file given on the Input tab, i.e. `CtfFind/job005/micrographs_ctf.star`. The output node (the `coords_suffix_topaz.star` file) should also be listed in the `RELION_OUTPUT_NODES.star` file. This file would therefore look like this:

```
data_output_nodes
loop_
_rlnPipeLineNodeName #1
_rlnPipeLineNodeType #2
External/job006/coords_suffix_topaz.star            2
```

14.5 On-the-fly processing: *Schedules*

Schedules are a new feature introduced to RELION-3.1. *Schedules* aim to provide a generalised replacement for the `relion_it.py` script for on-the-fly processing introduced to RELION-3.0. Although all the functionality to write python scripts is maintained in RELION-3.1, we no longer maintain and distribute the script itself for every RELION release. The reason for this is that for each new release some input/output options on the GUI will have changed, requiring a rewrite of (parts of) the python script. Instead, the *Schedules* framework in RELION-3.1 aims to formalise the decision process that was encoded in the python code of the `relion_it.py` script. The *Schedules* framework is built around the following key concepts: a directed graph that represents the logic of a series of subsequent RELION job-types is encoded in *Nodes* and *Edges*. *Nodes* can be either a RELION job or a so-called *Operator*; *Edges* form the connections between *Nodes*. In addition, *Schedules* have their own *Variables*.

All information for each *Schedule* is stored in its own subdirectory of the `Schedules/` directory in a RELION project, e.g. `Schedules/preprocess`. Within each *Schedule*'s directory, the `schedule.star` file contains information about all the *Variables*, *Edges*, *Operators* and *Jobs*. In addition, within the *Schedule*'s directory, a `schedule_pipeline.star` file contains information which jobs provide input for other jobs, and each job has a unique subdirectory (named according to its *JobNameOriginal*, see below) that contains a `job.star` file with the parameters for that job.

14.5.1 Variables

Three different types of *Variables* exist: *floatVariables* are numbers; *boolean-Variables* are either True or False; and *stringVariables* are text. Each Variable has a *VariableName*; a so-called *VariableResetValue*, at which the value is initialised; and a *VariableValue*, which may change during execution of the *Schedule* through the actions of Operators, as outlined below.

One special *stringVariable* is called `email`. When this is set, upon completion or upon encountering an error, the *Schedule* will send an email (through the Linux `mail` command) to the value of the `email` *stringVariable*.

14.5.2 Jobs

Jobs are the first type of *Node*. They can be of any of the jobtypes defined in the RELION pipeliner, i.e. Import, Motion correction, etc, including the new External. Any *Variable* defined in the *Schedule* can be set as a parameter in a Job, by using two dollar signs on the GUI or in the `job.star` file. For example, one could define a *floatVariable* `voltage` and use `$$voltage` on the correspondig input line of an Import job. Upon execution of the job inside the *Schedule*, the `$$voltage` will be replaced with the current value of the `voltage` *floatVariable*.

Jobs within a *Schedule* each have a *JobName* and a *JobNameOriginal*. The latter is defined upon creation of the job (see next section); the former depends on the execution status of the *Schedule*, and will be set to the executed RELION job's name, e.g. `CtfFind/job004`. In addition, each job has a *JobMode* and a *jobHasStarted* status. There are three types of *JobMode*:

- **new**: regardless of *jobHasStarted*, a new job will be created, with its own new *JobName*, every time the *Scheduler* passes through this Node.
- **continue**: if *jobHasStarted* is False, a new job, with its own new *JobName*, will be created. If *jobHasStarted* is True, the job will be executed as a continue job inside the existing *JobName* directory.
- **overwrite**: if *jobHasStarted* is False, a new job, with its own new *JobName*, will be created. If *jobHasStarted* is True, a new job execution will overwrite what was already present inside the existing *JobName* directory.

When a *Schedule* executes a Job, it always sets *jobHasStarted* to True. When a *Schedule* is reset, the *jobHasStarted* status for all jobs is set to False.

14.5.3 Operators

Operators are the second type of *Node*. Each operator within a *Schedule* has a unique name and a type. Operators can also have an output Variable: *output*,

on which they act, and up to two input Variables: *input1* and *input2*. Most, but not all operators change the value of their *outputVariable*.

The following types of operators exist to act on a *floatVariable*:

- **float=set**: *output* = *floatVariable input1*
- **float=plus**: *output* = *floatVariable input1* + *floatVariable input2*
- **float=minus**: *output* = *floatVariable input1* − *floatVariable input2*
- **float=mult**: *output* = *floatVariable input1* ∗ *floatVariable input2*
- **float=divide**: *output* = *floatVariable input1* / *floatVariable input2*
- **float=round**: *output* = ROUND(*floatVariable input1*)
- **float=count_images**: sets *output* to the number of images in the STAR file called *stringVariable input1*. *stringVariable input2* can be `particles`, `micrographs` or `movies`.
- **float=count_words**: sets *output* to the number of words in *stringVariable input1*, where individual words need to be separated with a **,** (comma) sign.
- **float=read_star**: reads *output* from a double or integer that is stored inside a STAR file. *stringVariable input1* defines which variable to read as: *starfilename,tablename,metadatalabel*. If *tablename* is a table instead of a list, then *floatVariable input2* defines the line number, with the default of zero being the first line.
- **float=star_table_max**: sets *output* to the maximum value of a column in a starfile table, where *stringVariable input1* specifies the column as *starfilename,tablename,metadatalabel*.
- **float=star_table_min**: sets *output* to the minimum value of a column in a starfile table, where *stringVariable input1* specifies the column as *starfilename,tablename,metadatalabel*.
- **float=star_table_avg**: sets *output* to the average value of a column in a starfile table, where *stringVariable input1* specifies the column as *starfilename,tablename,metadatalabel*.
- **float=star_table_sort_idx**: a sorting will be performed on the values of a column in a starfile table, where *stringVariable input1* specifies the column as *starfilename,tablename,metadatalabel*. *StringVariable input2* specifies the index in the ordered array: the lowest number is 1, the second lowest is 2, the highest is -1 and the one-but-highest is -2. Then, *output* is set to the corresponding index in the original table.

The following types of operators exist to act on a *booleanVariable*:

- **bool=and**: *output* = *booleanVariable input1* AND *booleanVariable input2*

- **bool=or**: *output* = *booleanVariable input1* OR *booleanVariable input2*
- **bool=not**: *output* = NOT *booleanVariable input1*
- **bool=gt**: *output* = *floatVariable input1* > *floatVariable input2*
- **bool=lt**: *output* = *floatVariable input1* < *floatVariable input2*
- **bool=ge**: *output* = *floatVariable input1* >= *floatVariable input2*
- **bool=le**: *output* = *floatVariable input1* <= *floatVariable input2*
- **bool=eq**: *output* = *floatVariable input1* == *floatVariable input2*
- **bool=file_exists**: *output* = True if *stringVariable input1* exists on the file system; False otherwise
- **bool=read_star**: reads *output* from a boolean that is stored inside a STAR file. *stringVariable input1* defines which variable to read as: *starfilename,tablename,metadatalabel*. If *tablename* is a table instead of a list, then *floatVariable input2* defines the line number, with the default of zero being the first line.

The following types of operators exist to act on a *stringVariable*:

- **string=join**: *output* = concatenate *stringVariable input1* and *stringVariable input2*
- **string=before_first**: sets *output* to the substring of *stringVariable input1* that occurs before the first instance of substring *stringVariable input2*.
- **string=after_first**: sets *output* to the substring of *stringVariable input1* that occurs after the first instance of substring *stringVariable input2*.
- **string=before_last**: sets *output* to the substring of *stringVariable input1* that occurs before the last instance of substring *stringVariable input2*.
- **string=after_last**: sets *output* to the substring of *stringVariable input1* that occurs after the last instance of substring *stringVariable input2*.
- **string=read_star**: reads *output* from a string that is stored inside a STAR file. *stringVariable input1* defines which variable to read as: *starfilename,tablename,metadatalabel*. If *tablename* is a table instead of a list, then *floatVariable input2* defines the line number, with the default of zero being the first line.
- **string=glob**: *output* = GLOB(*stringVariable input1*), where input1 contains a Linux wildcard and GLOB is the Linux function that returns all the files that exist for that wildcard. Each existing file will be separated by a comma in the *output* string.
- **string=nth_word**: *output* = the Nth substring in *stringVariable input1*, where N=*floatVariable input2*, and substrings are separated by commas.

Counting starts at one, and negative values for *input2* mean counting from the end, e.g. *input2=-2* means the second-last word.

The following types of operators do not act on any variable:

- **touch_file**: performs "touch *stringVariable input1*" on the file system
- **copy_file**: performs "cp *stringVariable input1 stringVariable input2*" on the file system. *Input1* may contain a linux wildcard. If *input2* contains a directory structure that does not exist yet, it will be created.
- **move_file**: performs "mv *stringVariable input1 stringVariable input2*" on the file system. *Input1* may contain a linux wildcard. If *input2* contains a directory structure that does not exist yet, it will be created.
- **delete_file**: performs "rm -f *stringVariable input1*" on the file system. *Input1* may contain a linux wildcard.
- **email**: sends an email, provided a *stringVariable* with the name `email` exists and the Linux command `mail` is functional. The content of the email has the current value of *Input1*, and optionally also *Input2*.
- **wait**: waits *floatVariable input1* seconds since the last time this operator was executed. The first time it is executed, this operator only starts the counter and does not wait. Optionally, if *output* is defined as a *floatVariable*, then the elapsed number of seconds since last time is stored in *output*.
- **exit**: terminates the execution of the *Schedule*, and sends a confirmation email if the `email` *stringVariable* is defined.

14.5.4 Edges

Two types of Edges exist. The first type is a normal *Edge*, which connects an *inputNode* to an *ouputNode*, thereby defining their consecutive execution. The second type is called a *Fork*. A Fork has one *inputNode* and two *outputNodes*. Whether one or the other output Node is executed depends on the current value of the booleanVariable that is associated with the Fork. Thereby, Forks are the main instrument of making decisions in *Schedules*.

14.5.5 Creating a new *Schedule*

The combination of the *Variables*, *Nodes* and *Edges* allows one to create complicated sequences of jobs. It is probably a good idea to draw out a logical flow-chart of your sequence before creating a *Schedule* as outlined below.

The creation of a *Schedule* is most easily done through the GUI, using the following command:

```
relion --schedule preprocess &
```

Note that the `--schedule` argument launches the GUI in a modifed mode, where slider bars and Yes/No pull-down menus are replaced by plain input text fields for more convenient placement of *Variables* with a `$$` prefix.

Variables can be added or deleted using the corresponding Set and Del buttons, respectively. The left-hand input field defines the *VariableName*, the right-hand input field defines its *VariableValue* and *VariableResetValue*. Any variable names that contain a *JobNameOriginal* of any of the *Jobs* inside the same *Schedule*, will be replaced by the current *JobName* upon execution of an operator.

Similarly, *Operators* can be added or deleted using the corresponding Add and Del buttons, respectively. The upper-left pull-down menu contains all possible *OperatorTypes*. The upper-right pull-down menu (next to the `->` sign) will define the *Output* variable, and the menu contains a list of all defined *Variables*. The lower two pull-down menus (with labels `i1:` and `i2:`) define *Input1* and *Input2* variables. Adding *Operators* with types for the input or output variables that are incompatible with the *OperatorType* will result in a pop-up error message.

Jobs can be added by first clicking on the job-type menu on the left-hand side of the top half of the GUI; then filling in the parameters on all tabs. Note that parameters may be updated with the current values of *Variables* from the *Schedule* by using the `$$` prefix, followed by the name of the corresponding *Variable*, as also mentioned above. If one *Job* depends on another *Job* inside the same *Schedule*, it is important to use the Browse button for its input parameters on the I/O tab of the job, and to select the input files from the same `Schedules` subdirectory. This is because, much like with *Variables*, all parameters of *Jobs* that contain a *JobNameOriginal* of any of the *Jobs* inside the same *Schedule*, will be replaced by the current *JobName* upon execution of that *Job*. This way, the *Schedule* will be able to define the correct dependencies between the newly created jobs upon its execution. Once all tabs on the top part of the GUI have been filled in, one needs to provide a *JobNameOriginal* in the input field with the label `Name:`. In addition, the *JobMode* needs to be chosen from the pull-down menu: **new**, **continue** or **overwrite**. Then, the job can be added to the *Schedule* by clicking the Add job button.

Finally, once all the *Variables*, *Operators* and *Jobs* are in place, one may add or delete the *Edges* using the corresponding Add and Del buttons, respectively. All defined *Operators* and *Jobs* will be available from the pull-down menus below these buttons. Normal *Edges* go from the left-hand pull-down to the right-hand pull-down menu, with the `->` sign in between them. *Forks* are defined by also selecting a *booleanVariable* from the pull-down menu with the `if:` label. When the *booleanVariable* is True, it will point to the *Node* defined by the lower-right pull-down menu (with the `:` label). When the *booleanVariable* is False, it will point to the *Node* defined by the upper-right pull-down menu (with the `->` label). The *Schedule* will be initialised (and reset) to the left-hand *Node* of the

first defined *Edge*. If the *Schedule* is not an infinite loop, it is recommended to add the **exit** *Operator* as the last *Node*.

To check the logic of the defined *Schedule* one can use the Set, Prev, Next and Reset buttons at the bottom of the GUI to set the *CurrentNodeName* to any of the defined *Nodes*; to go to the previous *Node*; to go to the next *Node*; or to reset all *Variables* and set *CurrentNodeName* to the left-hand side *Node* of the first *Edge*.

Also, using the 'Scheduling' menu on the top of the GUI, one can make a copy of any *Schedule* using the 'Copy Schedule' option. This may be useful to make a back-up of a schedule during the different stages of its creation. Once a *Schedule* has been created, it may be useful for more than one RELION project. Therefore, you may want to store it in a tar-ball:

```
tar -zcvf preprocess_schedule.tar.gz Schedules/preprocess
```

That tar-ball can then be extracted in any new RELION project directory:

```
tar -zxvf preprocess_schedule.tar.gz
```

14.5.6 Executing a *Schedule*

Once a *Schedule* has been created using the `--schedule` argument to the GUI, it is no longer necessary to provide that argument. One can instead launch the GUI normally (and have slider bars for numbers and Yes/No pull-down menues for booleans):

```
relion &
```

The *Schedule* can then be accessed through the 'Scheduling' menu at the top of the GUI, where all defined *Schedules* are available through the 'Schedules' sub-menu. The same GUI can be toggled back into the normal 'pipeline' mode from the same menu (or by pressing ALT+'p'). If one wants to start a *Schedule* from scratch, one would typically press the Reset button first, and then press the Run! button. This will lock the *Schedule* directory from further writing by the GUI and to reflect this, the lower part of the GUI will be de-activated. Once the *Schedule* finishes, the lock (in effect a hidden directory with the name `.relion_lock_schedule_NAME`) will be removed and the bottom part of the GUI will be re-activated. One can safely toggle between the pipeliner and the scheduler mode during execution of any *Schedule*, and multiple (different) *Schedules* can run simultaneously.

When a *Schedule* for whatever reason dies, the lock will not be automatically removed. If this happens, use the Unlock button to remove the lock manually. Be careful not to remove the lock on a running *Schedule* though, as this itself will cause it to die with an error.

If one would like to stop a running *Schedule* for whatever reason, one can press the Abort button. This will send an abort signal (i.e. it will create files called `RELION_JOB_ABORT_NOW` in the job directory of the currently running job, and in the directory of the *Schedule* itself), which will cause the *Schedule* to stop, and the lock to be removed. If one were to press the Run! button again, the same *Schedule* would continue the same execution as before, from the point where it was aborted. Most likely though, one has aborted because one would like to change something in the *Schedule* execution. For example, one could change parameters of a specific *Job*. To do so, select that *Job* by clicking on it in the list of *Jobs* in the lower part of the GUI. Then, edit the corresponding parameters on the relevant tabs of that *Job* on the top part of the GUI. Then, one may want to set *jobHasStarted* status to False, in order to make these options effective for all data processed in the *Schedule* thus far. For example, after running a *Schedule* for automated pre-processing for a while, one might want to change the threshold for picking particles in a auto-picking job. One would then reset the *jobHasStarted* status of the auto-picking job to False, while one would leave the *jobHasStarted* status of other jobs like Motion correction and CTF estimation to True. Thereby, upon a re-start of the *Schedule*, only new movies would be subjected to Motion correction and CTF estimation inside the same output directories as generated previously, but a new auto-picking directory would be created, in which all movies acquired thus far would be processed. Examples like these were very hard to do with the `relion_it.py` script in RELION-3.0.

14.6 Helical reconstruction

Shaoda He, a PhD-student in the Scheres group, implemented a workflow for the processing of helical assemblies. This involves additional tabs to the parameter-panels of the Auto-picking, Particle extraction, 2D classification, 3D classification, 3D auto-refine, Particle polishing and Mask create job-types. We do not have a separate tutorial for processing helical assemblies. The general principles remain the same as for single-particle analysis, which is covered in this tutorial. Therefore, users intending to use RELION for helical processing are still encouraged to do this tutorial first. For a detailed description of the helical options, the user is referred to the corresponding pages on the RELION Wiki, or to Shaoda's paper[6]. We are aware that a tutorial on helical processing is probably overdue, but due to time constraints we haven't got to doing that yet. Sorry...

14.6.1 Initial model generation for amyloids

The `relion_helix_inimodel2d` program is a new feature in RELION-3.1. It allows generation of an initial 3D reference for helical reconstruction, in particular for amyloids. It is run from the command line, and takes a selection of suitable 2D class averages as input. It will try to align these class averages

with respect to each other to form a continuously changing density that spans an entire cross-over. At the heart of the iterative refinement process lies a "tomographic" 2D reconstruction with all 1D pixel columns from the cross-over. Details of this program, together with a more elaborate documentation of its functionality remain to be published. Possible usage is:

```
relion_helix_inimodel2d --i Select/job056/class_averages.star \
--crossover_distance 800  --angpix 1.15 --maxres 9 --search_shift 3 \
--mask_diameter 250 --j 6 --iter 5 --o IniModel/run1

relion_helix_inimodel2d --i IniModel/run1_it005.star \
--iniref 1@IniModel/run1_it005_reconstructed.mrcs  --angpix 1.15 \
--maxres 6 --search_angle 2 --step_angle 0.5 --mask_diameter 250 \
--j 6 --iter 5  --o IniModel/run2
```

We like XMIPP-2.4 for live-updated display of the following images during the optimisation:

```
xmipp_show -img rec.spi after_reproject.spi before_reproject.spi -poll &
```

14.7 Sub-tomogram averaging

For sub-tomogram averaging, which was implemented with help from Tanmay Bharat, a former postdoc in the Lowe group at MRC-LMB, the same holds as for helical processing. Many general concepts remain the same as for single-particle analysis, and users intending to perform sub-tomogram averaging in RELION are encouraged to go through this tutorial first. For a detailed description of the sub-tomogram averaging procedures, the user is referred to the corresponding pages on the RELION Wiki, or to Tanmay's paper[2]. Please note that we are still actively working on making the sub-tomogram averaging pipeline more convenient to use and better. This work is done in close collaboration with the group of John Briggs, also at MRC-LMB. Meanwhile, please be advised that the sub-tomogram averaging pipeline is considerably less stream-lined than the single-particle one, and users should be prepared to do some scripting outside the RELION pipeline for many cases.

References

[1] Tristan Bepler, Andrew Morin, Micah Rapp, Julia Brasch, Lawrence Shapiro, Alex J. Noble, and Bonnie Berger. Positive-unlabeled convolutional neural networks for particle picking in cryo-electron micrographs. *Nature Methods*, 2019.

[2] Tanmay A. M. Bharat, Christopher J. Russo, Jan Lwe, Lori A. Passmore, and Sjors H. W. Scheres. Advances in Single-Particle Electron Cryomicroscopy Structure Determination applied to Sub-tomogram Averaging. *Structure (London, England: 1993)*, 23(9):1743–1753, September 2015.

[3] Shaoxia Chen, Greg McMullan, Abdul R. Faruqi, Garib N. Murshudov, Judith M. Short, Sjors H. W. Scheres, and Richard Henderson. High-resolution noise substitution to measure overfitting and validate resolution in 3d structure determination by single particle electron cryomicroscopy. *Ultramicroscopy*, 135:24–35, December 2013.

[4] Rafael Fernandez-Leiro and Sjors H. W. Scheres. A pipeline approach to single-particle processing in RELION. *Acta Crystallographica. Section D, Structural Biology*, 73(Pt 6):496–502, June 2017.

[5] Timothy Grant and Nikolaus Grigorieff. Measuring the optimal exposure for single particle cryo-EM using a 2.6 reconstruction of rotavirus VP6. *eLife*, 4:e06980, 2015.

[6] Shaoda He and Sjors H. W. Scheres. Helical reconstruction in RELION. *Journal of Structural Biology*, in press, 2017.

[7] Dari Kimanius, Bjrn O Forsberg, Sjors HW Scheres, and Erik Lindahl. Accelerated cryo-EM structure determination with parallelisation using GPUs in RELION-2. *eLife*, 5, November 2016.

[8] Alp Kucukelbir, Fred J Sigworth, and Hemant D Tagare. Quantifying the local resolution of cryo-EM density maps. *Nature methods*, 11(1):63–65, January 2014.

[9] Ali Punjani, John L. Rubinstein, David J. Fleet, and Marcus A. Brubaker. cryoSPARC: algorithms for rapid unsupervised cryo-EM structure determination. *Nature Methods*, 14(3):290–296, March 2017.

[10] Alexis Rohou and Nikolaus Grigorieff. CTFFIND4: Fast and accurate defocus estimation from electron micrographs. *Journal of Structural Biology*, 192(2):216–221, November 2015.

[11] Peter B Rosenthal and Richard Henderson. Optimal determination of particle orientation, absolute hand, and contrast loss in single-particle electron cryomicroscopy. *Journal of Molecular Biology*, 333(4):721–745, October 2003.

[12] Sjors H W Scheres. Classification of Structural Heterogeneity by Maximum-Likelihood Methods. In *Cryo-EM, Part B: 3-D Reconstruction*, volume 482 of *Methods in Enzymology*, pages 295–320. Academic Press, 2010.

[13] Sjors H W Scheres. A Bayesian view on cryo-EM structure determination. *Journal of Molecular Biology*, 415(2):406–418, January 2012.

[14] Sjors H W Scheres. RELION: Implementation of a Bayesian approach to cryo-EM structure determination. *Journal of Structural Biology*, 180(3):519–530, December 2012.

[15] Sjors H W Scheres and Shaoxia Chen. Prevention of overfitting in cryo-EM structure determination. *Nature methods*, 9(9):853–854, September 2012.

[16] Sjors H W Scheres, R. Nunez-Ramirez, C. O. S Sorzano, J. M Carazo, and R. Marabini. Image processing for electron microscopy single-particle analysis using XMIPP. *Nature Protocols*, 3(6):977–90, 2008.

[17] J M Smith. Ximdisp–A visualization tool to aid structure determination from electron microscope images. *Journal of structural biology*, 125(2-3):223–228, May 1999.

[18] Guang Tang, Liwei Peng, Philip R Baldwin, Deepinder S Mann, Wen Jiang, Ian Rees, and Steven J Ludtke. EMAN2: an extensible image processing suite for electron microscopy. *Journal of Structural Biology*, 157(1):38–46, January 2007.

[19] Kai Zhang. Gctf: Real-time CTF determination and correction. *Journal of Structural Biology*, 193(1):1–12, January 2016.

[20] Shawn Q. Zheng, Eugene Palovcak, Jean-Paul Armache, Kliment A. Verba, Yifan Cheng, and David A. Agard. MotionCor2: anisotropic correction of beam-induced motion for improved cryo-electron microscopy. *Nature Methods*, 14(4):331–332, April 2017.

[21] Jasenko Zivanov, Takanori Nakane, and Sjors Scheres. Estimation of high-order aberrations and anisotropic magnification from cryo-em datasets in relion-3.1. *bioRxiv*, 2019.