EMAN2.12 Reconstruction Tutorial

Using the Project Manager

This tutorial was updated in October 2015. It should be used with EMAN2.12 or a nightly snapshot release from after that date.

➡ Arrow items like this are general tips, not directly part of the tutorial sequence.

➡ The main source for EMAN2 documentation is the Wiki at: http://blake.bcm.edu . There is also a Google Group for software support and discussions: http://groups.google.com/group/eman2 . You only have to join the Wiki/Group to post your own comments.

➡ **GUI Tips:** EMAN2 will work best with a 3-button scroll mouse, though there are alternatives using keyboard modifiers for one button mice or trackpads on Macs.

- In most EMAN2 windows (2-D images, 3-D volumes, plots, etc.), the middle mouse button will open a control panel for the widget, which is different for each widget type
- The right mouse button is used for panning in 2-D or 3-D image windows, and can be used to zoom (by shift+dragging), and to reset the zoom (clicking) in plot windows.
- The scroll-wheel will generally act as a zoom. control-panel for more precise control
- If you have a one button mouse, holding down the alt/option key combined with a mouse click will serve the same role as a middle-click.
- In the control panels, and other places in the EMAN2 interface you may encounter 'Value Sliders'. A slider is attached to a text-box with a number in it. Dragging the slider controls the number, and entering a number will change the slider. In addition, the text-box can be used to control the range of the slider and get more precise control. By typing **<value** or **>value** in the text box you can change the limits of the slider.

➡ Text you see in *italics* will generally refer to labels in the GUI, such as buttons to press. Text you see in **bold**, are commands to be typed in. Items like: <param> are parameters you should fill in (without the < >). Items like: [param] are optional parameters (again, fill without the [ ]).

➡ **Mac users** - If you are running El Capitan (10.11), the new operating system has caused some new bugs to appear, including windows that don't always close when they should, and some other unusual behaviors. This is definitely a problem for EMAN2.12. Hopefully we will have these issues corrected in a daily snapshot version in the near future.

➡ **Mac Users** - If you have a Retina display (most will nowadays), then you have a lot of control over the resolution of your display. By default, your display will give you an effective resolution of 1440x900 pixels. This isn't enough space to do image processing well. For the workshop, we strongly suggest opening System Preferences-> Display and adjusting it to "More Space" which will be 1920x1200 effective resolution.

➡ **Windows users** - Windows **will** make your life more difficult with EMAN2. If you have no choice but Windows, please remember that programs must be run from the windows command-line, not by clicking on icons. Installation of an 'enhanced command-line' tool for windows, such as Console 2.x, will make your life somewhat easier.

➡ **Windows 7+** likes to aggressively kill "unresponsive" programs. This means if you have asked EMAN2.1 to do something from the GUI, and it hasn't finished yet, simply clicking on an open window while you wait may cause the whole program to be killed. We have no solution for this at present other than "don't do that".

Introduction
EMAN2 can be used at many different levels ranging from high-level task-based workflow, to command-line utilities, to writing code in Python or C++. In this tutorial, we will be focusing primarily on the task-focused high-level Project Manager interface. This interface will help you work step-by-step through established techniques such as single particle analysis and subtomogram averaging.

Technically, EMAN2/SPARX supports all 3 of the major platforms: Linux, Mac and Windows. That said, development is almost entirely done on Linux and Mac, and few of the developers even have a Windows PC available for routine testing. Why ? Many image processing projects are large, and require Linux clusters to achieve sufficient compute power to complete reconstructions in reasonable periods of time. While, it can be possible to produce subnanometer resolution reconstructions on a laptop in a few hours, some projects still require 10,000+ CPU-hr to complete, particularly in cases of heterogeneity and low symmetry.

We will be using a Beta Galactosidase data set for this tutorial. This data set is a subset of one of the data sets being used in the 2015 Map Challenge sponsored by the EMDatabank. The original paper published with this data achieved a resolution of 3.2 Å. While the full data set can be processed on a workstation in a day or two, it is a bit much for a laptop in a one day tutorial. So, we will be using a subset of the data, which has been down-sampled for faster processing. Even the reduced data set used here is capable of producing about 4 Å resolution, though the final refinement may still be pushing the capabilities of some laptops.

We strongly recommend going through this tutorial using the provided data set. Once you understand how everything is supposed to work, then you can use your own data or download additional public data sets from sites like http://www.ebi.ac.uk/pdbe/emdb/empiar/. The map challenge (http://challenges.emdatabank.org/?q=2015_map_challenge) is another good source for interesting high resolution test data sets.

There are several important things this tutorial does not cover:

1) Movie mode processing
    - 1 movie can easily be 5GB. Just not practical for laptops.
    - e2ddd_movie.py and e2ddd_particles.py
2) Evaluating micrographs
    - This is a VERY important topic, but again, it can take a couple of hours to do properly.
3) Particle picking
    - We have done this tutorial in the past, but it takes a lot of time
4) Subtomogram averaging
    - A big topic, which really needs its own 4 day workshop
5) Detailed Command-line usage
    - I will mention this as I can during the presentations
6) EMAN from Python
    - A fun topic, but not for the whole audience. We have some decent recordings covering this online.

Check the Wiki for other available tutorials, some of which include videos

➡ **If you are an experienced user (or an impatient one), there is a much shorter 'quick' version of the tutorial without all of the detailed explanations, at the end of this document. The section numbering is the same in both versions, so you can move back and forth as necessary. ie- you can go through the short version, and if you don't understand something, jump back to the detailed version for more information. The detailed section also includes many useful comments and explanations for solving problems with your own data, not just the tutorial set.**

Detailed Tutorial

Before getting started, it's a good idea to get a feel for the relative speed of your computer (to set expectations). Run **e2speedtest.py**. This will give you a score telling you how fast a single processor is on your computer. If your machine has 4 cores, you multiply this number by 4 to get a relative performance value. Note, however, that some processors have a 'turbo' mode, and if you are using only 1 processor (which is what the test does), it will run faster than 1 core normally will. This can exaggerate the speedtest score by as much as 20-30%. My 2015 MacBook Pro scores ~1.0 on this test.

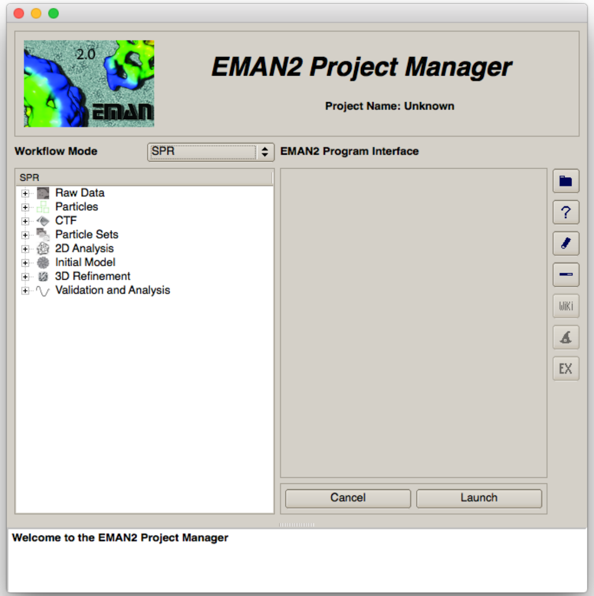
1-2. Prepare your project folder

Unzip the sample data archive in a convenient place. This will give you a folder called *bgal*, with a *0README.txt* file in it and a *particles* folder. Do not run any EMAN2 programs yet. First, from the command line: **cd /path/to/bgal** That is, you want to be in the 'bgal' folder, such that typing **ls** (**dir** on Windows) will show you the two files mentioned above.

The folder you are now in is called the 'project folder'. You will run virtually all commands from this location. All files associated with this project live in this folder or subfolders inside this folder. The system is designed so all of your data and results are self contained, if you use the GUI in the normal way. The goal is that at the end of the project you will be able to trace any step you performed to produce any of your results without ambiguity. Any EMAN2 commands you run should be run from this folder, not from subfolders or other locations.

3. Project Manager

Type **e2projectmanager.py**. A window should appear that looks roughly like this:



Welcome to the EMAN2 Project Manager

- ➡ On the Mac windows may open, but not appear in front of other windows. Click on the rocket-ship icon in your dock to bring them to the front if you can't find them.
- ➡ Single particle reconstruction tends to use a lot of disk space. Disk storage is so inexpensive now, we err on the side of keeping intermediate files, on the theory that they may be useful later. You can always delete things you don't need, but it's very hard to get something back that you didn't keep in the first place. Modern

microscopes can produce upwards of 2 TB of data per day, so we consider the tens of GB of files that may be generated during a reconstruction to be quite reasonable, comparatively speaking.

➡ When you select a specific item in the projectmanager it is simply giving you a graphical interface to commands you could issue equally well from the command-line. If you click on the *command* tab when entering parameters for a program it will show you the exact command that will be launched. You can add additional options to the end of this command before pressing *Launch*. Just be warned that if you do this and switch back to the properties tab and additions you made which are not shown in the property list will be lost.

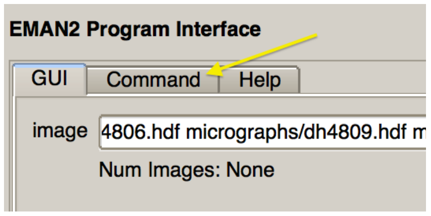


image 4806.hdf micrographs/dh4809.hdf m
Num Images: None

4. Set project properties

The project manager interface is an expandable tree. Each level of the tree has a form with associated information or parameters, even the items which just appear to be labels for the levels below them.

- Begin by setting the overall properties of the project, by using the *Project → Edit Project* menu item.
- For the B-gal demo data set, enter the following parameters: mass = 400, Cs = 2.7, voltage = 300, apix=1.275

➡ A note on image file formats: There are many file formats used in the CryoEM community, including familiar formats like TIFF and PNG, cryo-EM specific formats like IMAGIC and SPIDER, and proprietary formats like DM4 and SER. EMAN2 supports virtually all file formats used in the community (http://blake.bcm.edu/emanwiki/Eman2DataStorage). People often ask "how do I convert file X to work with EMAN2". In most cases, the answer is "you don't have to!" Any EMAN2 program should transparently read any supported file format. Similarly, when specifying output files, simply use the standard file extension and EMAN2 will write to almost any format as well. That said, EMAN2.1 uses an interdisciplinary format called HDF5 as its native format.

➡ MRC format has some special requirements, as the format really only supports one image per file. To store multiple images in one file, it uses a trick known as "stacking" where a bunch of 2-D images are stacked on top of one another to produce a 3-D volume. This trick is NOT used in other file formats, even if they are sometimes called "image stacks". EMAN can deal with normal MRC files as well as MRC stack files. If the files have a ".mrc" extension they will be treated as single image 2-D or 3-D files. If you want to use stacking, you must use the ".mrcs" extension both reading and writing.

5-10. CTF Correction

We are now beginning the process of correcting for the Contrast Transfer Function (CTF) of the microscope. It is important to understand that CTF correction cannot be performed completely as a preprocessing step. Programs that claim to do this are likely performing only the subset of CTF correction known as phase-flipping, and ignoring amplitude correction.

- CTF correction is broken into a sequence of steps. While the fitting is completely automated, it is a good idea to review at least some of the results manually. This is an important step in quality control of your data, as it gives you another opportunity to eliminate images with low quality particles which could actually damage your reconstruction. If the project simply has too many frames to manually review them all, you may consider reviewing the closest and furthest from focus images. These are the most likely areas where errors will occur.

- Here is a basic overview of the normal procedure. Detailed instructions follow:
  - Run automatic fitting with 2x oversampling
  - Run manual fitting on at least a few images. Do a quick check to make sure determined defocuses are ok. If there are a lot of frames start with a few of the closest to and a few of the furthest from focus.
  - Run structure factor determination on the images you just hand-checked (should cover a range of defocuses.
  - Run automatic fitting again (having a structure factor may slightly improve results)
  - Run manual fitting. Evaluate your data critically. Adjust quality of any images you suspect may not be as good.
  - Generate output with refinebysnr. This:
    - Fine tunes defocus based on high resolution SSNR optimization (optional)
    - Performs phase-flipping correction, and optional post-processing
    - Stores CTF parameters including SSNR in particle headers.

Now a bunch of tips before we start the actual process:

➡ If you are working with your own data, and happen to be working in negative stain, we strongly encourage you to perform CTF correction, even if you do not believe it will gain you any resolution. There is more to CTF correction than simply achieving higher resolutions.

➡ This is a good point at which to explain the file naming convention for raw data within the project. The next several steps are working only on the boxed out particles, so at this point in processing it doesn't matter whether you have micrographs, or movies or any other files. The only thing you need at this point is a folder called particles. Inside this folder there should be one image file for each micrograph:

- Each file must be named *micrographname.hdf* or *micrographname_ptcls.hdf*
- The *micrographname*s above must match the filenames present in the micrographs folder if present.
- While EMAN does support pretty much any file format in general, within the project, you must use HDF format, because other formats do not support arbitrary header information, which EMAN makes use of during processing. When importing particles from some other software, they are always converted to HDF.
- During the following steps, we will be creating modified versions of the particles (phase flipping CTF correction, downsampling, filtering, etc.) The naming convention used for these versions is a __ (double underscore) separator. eg - *abc123_ptcls.hdf* might become *abc123__ctf_flip.hdf.* You are free to make any such derived files as long as they include exactly the same particles as the parent _ptcls files.

➡ SSNR = Spectral Signal to Noise Ratio. This is a measure of data quality, and its estimation from the particles themselves is one of EMAN2's most unique features. A Signal to Noise Ratio of 1.0 means there is an equal amount of signal and noise. SSNR is a measurement of noise level as a function of resolution. Naturally, SSNR is higher at lower resolutions, and lower at higher resolutions. SSNR is additive if proper weighting is applied when averaging particles together. Meaning, if the SSNR were 0.05 at 10Å resolution, it should take roughly 20 particles in that orientation to achieve a minimal average SSNR of

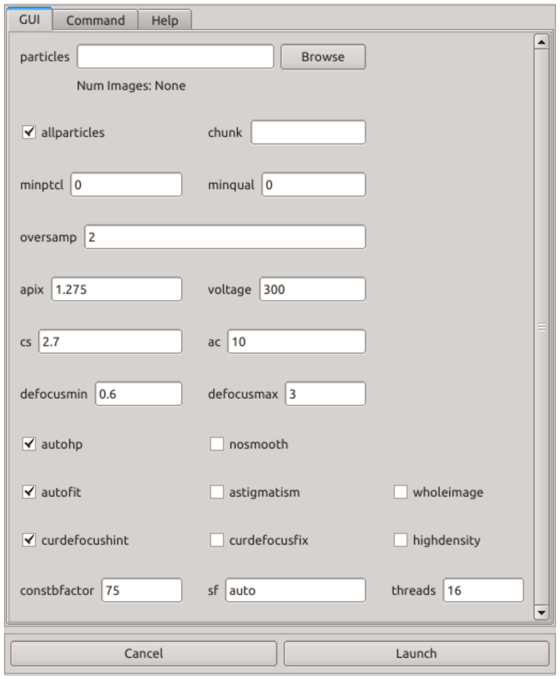
1. In theory you could use this method to achieve arbitrary resolutions, but in practice SSNR values below ~0.02 (50 times more noise than signal) have empirically proven difficult to recover, regardless of the number of particles.

➡ Not much effort is spent on automatic B-factor, particularly without a structure factor, but it doesn’t really matter. Unlike EMAN1, the B-factors are really only used in structure factor determination, and generally only have a minor impact even there. Usually the simplest approach is to manually determine the B factor for a few images manually by assessing falloff at high resolution, then pick a decent fixed value, and fill in to the "constbfactor" box. If you have some other reason to desire accurate B-factor values, you can leave constbfactor at -1 (autofit) (you will get a somewhat better value when you refit after determining a structure factor).

➡ When working with your own data, you may be tempted to take “CTF corrected particles” from some other software and use them in EMAN2. This is a very bad idea. When programs say they have “CTF corrected particles”, it generally means they have been phase-flipped, or phase flipped and filtered with some sort of per-particle Wiener filter. This won’t give EMAN2 the information it needs for accurate amplitude correction, classification or alignment. It will still do as good a job as it can, but this will force the refinement to fall back to very basic algorithms which will not give optimal results. We strongly recommend importing <u>uncorrected</u> particles and performing CTF correction within EMAN2. It is often possible to import defocus values along with the particles if you want to insure both programs use the same values (e2import.py or e2emx.py).

5. Automated fitting (see 9-14 above before starting here)

- Select *CTF → Automated Fitting*
    - check the *allparticles* box. This is an alternative to manually selecting all of the individual images in your particles folder. As a reminder, if *allparticles* is selected, anything in the *particles* box will be ignored completely.
    - *minptcl* allows you to skip any frames with fewer than the specified number of particles. Leave this at 0 for the sample B-gal.
    - *minqual* allows you to skip any frames where the user has set the quality below a specified value. Again, you can leave this at 0 for now.
    - Set *oversamp* to 2. This will provide better fitting in most cases. If you are working with your own data and the box size is large (256+) you can consider leaving this at 1.
    - Select *autohp*. This will modify the SNR curve to eliminate the first sharp peak that appears in virtually all single particle data due to ice gradients and other issues. This peak isn’t completely removed, but is just heavily downfiltered, and in most cases it will noticeably improve alignments. However, in some projects, the structure factor is such that this option will filter out too much low resolution information. If you see strange low-resolution artifacts, you may try disabling this.



particles
Num Images: None

☑ allparticles		chunk			
minptcl	0	minqual	0		
oversamp	2				
apix	1.275	voltage	300		
cs	2.7	ac	10		
defocusmin	0.6	defocusmax	3		
☑ autohp		☐ nosmooth			
☑ autofit		☐ astigmatism		☐ wholeimage	
☑ curdefocushint		☐ curdefocusfix		☐ highdensity	
constbfactor	75	sf	auto	threads	16

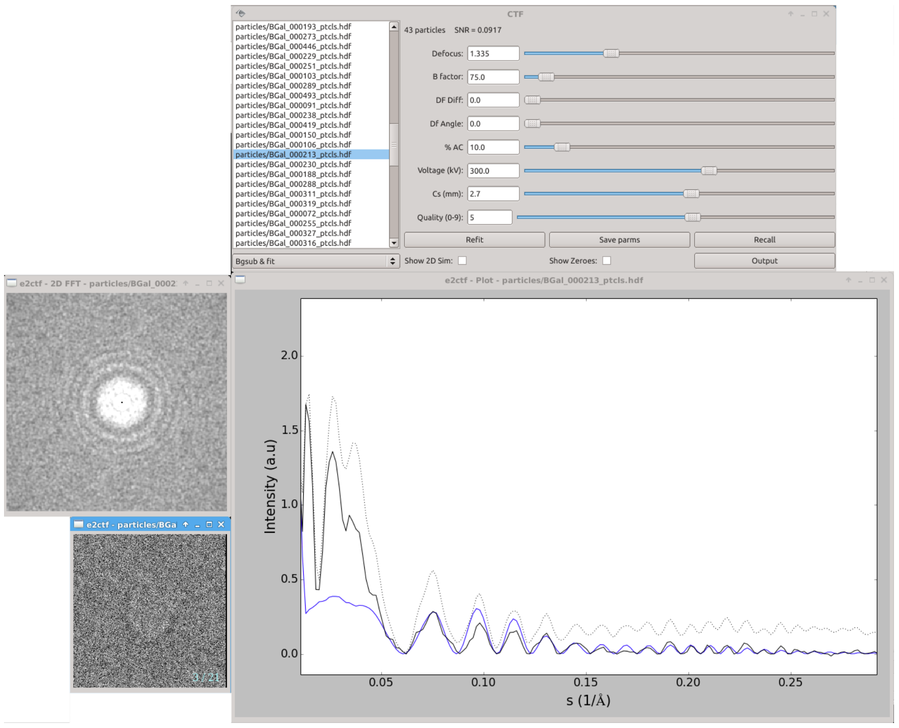
  - Select *curdefocushint*. This will make use of any existing defocus information during fitting. Only uncheck this if you have bad defocus values from earlier fitting attempts and want to start from scratch.
  - Do not select astigmatism for now. If trying to correct astigmatism, you should always do a refinement without it first.
  - You should see: *cs* = 2.7, *voltage* = 300, *apix* = 1.275 filled in for you. If not, you missed something in step 4. Enter the correct values here AND go back to step 4 to get them set in the project correctly.
  - *constbfactor* = 75
  - Threads - This should be set to the number of processors (cores) on your computer (usually 2 or 4 on a laptop), whenever you see this box. On a cluster, this would be the number of cores on one node. If you have a machine newer than ~2012, you may consider using ~25% more threads than you have processors, for a small additional performance boost.
  - *Launch.*
  - You may consider opening the Task Manager (icon) to monitor the progress of this and other running jobs (may not work reliably on some platforms). Otherwise, watch the console where you ran e2projectmanager for the output to finish.

➡ When working with your own data, box size is an absolutely critical factor. As mentioned when discussing particle extraction, you should optimally use a box size at least 1.5x the longest axis of your particle. If you are using particles extracted from some other program, and used a box size much smaller than this, you should consider going back to the original micrographs and re-extracting the particles with a larger box for several reasons: With a small box, you cut off some of the information recovered when phase-flipping the particles. Also, several of EMAN's algorithms (such as automatic centering of class-averages) make assumptions about the relative size of the particle and box.

➡ The algorithm EMAN uses for SSNR estimation and CTF fitting is somewhat sensitive to very high particle density. If 2 or 3 neighboring particles happen to impinge on the central particle's box, it isn't normally a significant problem. However, if you have truly dense particle packing, with much of the space outside the central particle occupied by edges of other particles, it can lead to issues with alignment and SSNR estimation. The *highdensity* checkbox can partially compensate for this effect, but you may also have to take other steps, such as apply a soft mask to (fairly well centered) particles as part of the CTF generate output step below.

6. Interactive Tuning

- When automatic fitting is done, then use *CTF→interactive tuning*
  - Select *allparticles* and *sortdefocus*, then press *Launch.* You will get 4 windows (shown below). You may arrange these windows as you like the first time you run the program. If you run it again in this project, it should put them back in the same location.
  - This analysis is based on particles rather than tiled regions from the micrograph, as used in the earlier micrograph assessment step (which we skipped with this demo data set). Basing the analysis on particles allows us to accurately estimate the SSNR and structure factor (next step) from the particle data. While it is possible to estimate defocus and astigmatism from random boxed out regions of the micrograph, the method used here relies on the fact that each box contains a particle reasonably close to the center of the



```
particles/BGal_000193_ptcls.hdf
particles/BGal_000273_ptcls.hdf
particles/BGal_000446_ptcls.hdf
particles/BGal_000229_ptcls.hdf
particles/BGal_000251_ptcls.hdf
particles/BGal_000103_ptcls.hdf
particles/BGal_000289_ptcls.hdf
particles/BGal_000493_ptcls.hdf
particles/BGal_000091_ptcls.hdf
particles/BGal_000238_ptcls.hdf
particles/BGal_000419_ptcls.hdf
particles/BGal_000150_ptcls.hdf
particles/BGal_000106_ptcls.hdf
particles/BGal_000213_ptcls.hdf
particles/BGal_000230_ptcls.hdf
particles/BGal_000188_ptcls.hdf
particles/BGal_000288_ptcls.hdf
particles/BGal_000311_ptcls.hdf
particles/BGal_000319_ptcls.hdf
particles/BGal_000072_ptcls.hdf
particles/BGal_000255_ptcls.hdf
particles/BGal_000327_ptcls.hdf
particles/BGal_000316_ptcls.hdf
Bgsub & fit

43 particles    SNR = 0.0917
Defocus: 1.335
B factor: 75.0
DF Diff: 0.0
Df Angle: 0.0
% AC: 10.0
Voltage (kV): 300.0
Cs (mm): 2.7
Quality (0-9): 5
Show 2D Sim: ☐
Show Zeroes: ☐
```

box. If too many "bad" particles are present, the SSNR and structure factor estimates will be impacted.

- The control panel window (titled CTF) allows you to select which image to work on, and adjust the various parameters for that image interactively. The particles window shows the average of the first 20 particles (without alignment) and the first 20 particles individually (use up and down arrow in that window). The 2-D FFT window shows the average 2-D power spectrum for the particles in the current image, optionally (if Show 2D Sim is set) the simulated 2-D curve, and optionally (if Show Zeroes is set) the first several zeroes.
- Finally, the plot window shows several curves. In "Bgsub & Fit" mode, the black curve is the background subtracted, rotationally averaged power spectrum. The dotted black curve is the same thing, but using a different background subtraction scheme which exaggerates the CTF oscillations. The blue curve is the fit based on the parameters shown in the control panel. We do not yet have a structure factor curve, so, while ideally the blue and black curves would match perfectly, at this point we can expect significant deviations at low resolution. Don't bother trying to 'fix' this by adjusting sliders, it is normal for this point in the process. Even later on, there is no need for a perfect fit in amplitudes as long as the zeroes are in the right places.

➡ In the plot window, to zoom in on a region of the plot, drag with the right mouse button. To rescale, click the right mouse button without moving the mouse. Left-mouse drag will show cross-hairs. For the X-axis it will show both the value and the reciprocal of the value (resolution). As usual, middle-click will open the control-panel.

- While there are many interesting things we can do with this program, at this point we mainly need to insure that the defocus values are correct. The automatic fitting is quite good, but with some specimens, or with particularly low quality data, it will make occasional errors. Generally if it's wrong, it is significantly wrong. If you find an image with a significantly incorrect defocus, adjust it so it's approximately correct, and hit the *refit* button (you can use it several times if necessary). This will be much easier if you zoom in on plot window as shown above. Typically a vertical range of ~0-1 and a horizontal range of ~0-0.25 will work well.
- If this doesn't solve the problem, you can fit manually, then press the *save parms* button instead.
- This project is small enough that you can take the time to look at all of the images. If you have 1000 images, and the automatic fitting seems to work fairly well, you may just check

the first dozen closest to focus and last dozen far from focus images. If all of those are fit correctly, odds are good that the rest are too.

- If you didn't screen and eliminate bad micrographs before boxing (which would have been difficult in this case since I didn't give you the micrographs), this can be a good time to try and identify any bad images in the data set. You can do this by looking for artifacts like drift (directional falloff in the 2-D power spectrum) or strong astigmatism (asymmetry in the 2-D power spectrum), or overall poor high resolution signal. This particular subset of the full B-gal challenge data set is almost entirely 'good' images, but I left 4 marginally bad images in the set you can try and find. If you don't find and exclude them they aren't bad enough that they are likely to impact your structure significantly.

➡ Amplitude contrast can only be determined experimentally through some rather tricky experiments, and frankly, slightly different values will not have a strong impact in most reconstructions. Values of 5-15% generally work best for cryo data. However, for negative stain data, you will likely want to use a much larger value than the default 10% (likely ~40-80% will work better).

➡ When screening manually, there are some keyboard shortcuts available to speed the process. You must select one of the micrograph names in the list to move the keyboard focus on that widget for this to work: up and down arrow select different images; left and right arrow make small adjustments to defocus; pressing a number 1-9 will set the quality for the current image to that value; 'b' will set the B-factor to 100; 's' will save the current parameters; and 'r' will restore the currently saved parameters.

7. Structure Factor

➡ The structure factor we are referring to here is the 1-D structure factor, rather than the 3-D structure factor(s) often referred to in X-ray crystallography. If you had a perfect noise-free 3-D electron density map, took a 3-D Fourier transform, then rotationally averaged its magnitude, that 1-D curve (basically a power spectrum) is what we refer to as the structure factor. Having a good estimate of this curve permits us to do a good job with CTF amplitude correction. Unfortunately, it is difficult to derive an accurate structure factor from the data at high resolution, so what we do instead is determine the low resolution structure factor from the data, typically to a resolution of 15-20 Å, then at resolutions higher than this use a standard protein structure factor with a mix of alpha and beta content. This works well in the majority of situations.

- CTF→generate structure factor
    - Either select ‘allparticles’ or use the browser to select a subset of the data for this computation. Typically ~10-15 data sets are enough for a good structure factor, and using fewer sets will make it run faster. If you select a subset, generally you should use the images with the largest number of particles, insuring that you cover a range of defocus values.
    - After pressing *Launch*, the process will normally only take a couple of minutes. As before you can monitor progress either by looking at the task manager or the terminal where you launched the project manager from.

➡ Once complete, there are 3 additional files in your project folder, *strucfac*.* The file *strucfac.txt* contains the actual structure factor as a function of spatial frequency in a 2-column text file. You can plot it by double-clicking, but you will need to put the Y-axis on a log scale to see anything useful. If present, this file will be used automatically in the remainder of processing. If you have a structure factor curve from some other source, such as an X-ray solution scattering experiment, you can replace this file with whatever curve you like, and it will be used from that point on. If converting a solution scattering curve, however, beware of the differing spatial frequency conventions. In CryoEM we use 's' (the reciprocal of the real-space periodicity) and in solution scattering they normally use q (momentum transfer). i.e. - there is normally a 2π conversion factor.

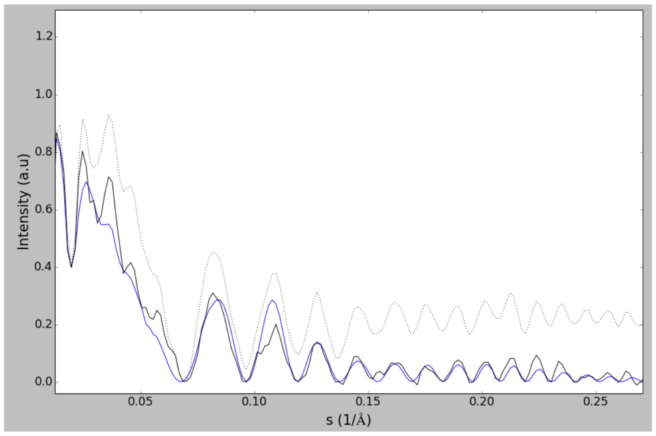
8. Rerun Automatic Fitting

With a decent structure factor in hand, sometimes automatic CTF fitting will be marginally improved. Certainly it should get no worse anyway. There isn't usually much point in iterating this process further, though. That is, you don't need to compute another structure factor after refitting.

- Rerun *CTF → Automated Fitting*
  - Default options should be correct.

9. Interactive Tuning

If you like you can run interactive tuning again and you should see that the blue and black curves are a much better match. In cases where they do not match well, adjusting the B-factor will generally produce an improved fit. If you find that adjusting the B factor to match at low resolution requires sacrificing high resolution oscillations, that means the initial B-factors or the Amplitude Contrast you selected before determining the structure factor may not have been optimal. While you can go back and play with these values and redetermine the structure factor, unless you used extremely inaccurate values (like an %AC of zero), it isn't likely to have a significant impact on your reconstruction. If you find a few images which really don't seem to match well at all, it is very likely that those images have a large fraction of bad particles, or for some reason have a very different particle orientation distribution than the average image.



- Now run *CTF→interactive tuning, again*
  - If you manually adjust parameters at this stage, you should focus on a good fit at high resolution, and not be concerned about any remaining poor low-resolution match, but it should not really be necessary to make adjustments at this stage.
  - If you would like to assess data quality, you need SSNR curves rather than the basic CTF fit. In the pop-up menu under the list of images, you'll find *SNR* and *Smoothed SNR*. For good data, the low resolution SSNR (between s=0.01-0.03, 30-100 Å) should peak higher than ~0.5. Images that do not achieve an SSNR of at least 0.5 at low resolution are unlikely to align properly, and may act as pure noise in your refinement.
  - You may also look at high resolution and consider whether there is strong enough signal in the image to improve your reconstruction at the desired resolution. Once the SSNR falls much below 0.02 it usually won't contribute much to the reconstruction, even if you have a lot of particles.
  - If you decide one image is worse than the others, you may use the quality slider, and set it to a lower than the default (5) value. Similarly if you see any unusually good images,

  you may consider increasing this slider. It is not necessary to press *save* after adjusting the quality slider.
- The actual values you use for quality are arbitrary. They have no direct meaning to EMAN2, but will be displayed to you later in the process when you are deciding which images to include in the reconstruction

10. CTF Generate Output

- run *CTF→generate output*
  - The refinebysnr checkbox should normally be checked. While in some cases (low resolution data) it won’t do any good, it will rarely do anything harmful either. This option will make a final pass at making very small adjustments to the defocus value to optimize the measured SSNR of your data at high resolution.
  - The other options are used to select what types of output files to generate. New in EMAN2.12, there are some new options which give you much greater flexibility in project management, and efficiently achieving a good structure. It is now possible to have downsampled and filtered versions of your particles present in the same project folder, and to run refinements at different levels of sampling (different A/pix values) within a single project without any conflicts.
  - You will select one of these two options to produce the phase flipped particles you use for your final high resolution refinements:
    - *phaseflip* - this will produce unfiltered phase-flipped particles
    - *phasefliphp* - This will produce phase flipped particles which have been high-pass filtered to eliminate the very low resolution peak present in the data (ice gradients, etc.). Normally this is a good thing. Sometimes it is not, and can end up filtering out important low-resolution signal. You won’t really know until you try. On the GroEL data we’re using here, this is a good option to use.
  - The next two options produce specialized versions of your particle data for specific purposes.
    - *wiener* - This will produce Wiener filtered particles. Note that these particles are not normally useful for reconstructions or 2-D averaging. If many Wiener filtered particles are averaged together the result is heavily over filtered (blurrier than they should be). Really the only good purpose for these particles is manually identifying bad particles, and there are other options which can serve this purpose similarly well.
    - *phaseflipsmall* - This is a convenience item. It downsamples your data by 2x, does a mild high-pass filter, and low pass filters to ~17 Å. These particles are often useful for generating reference-free class-averages, identifying bad particles, and making initial models. The parameters are all fixed, so if you wish to modify any of them, you will need to use the *phaseflipproc* options below.
  - For the demo data set we need only phaseflip from the options above.
  - The next set of options allow you to produce particles with up to 4 arbitrary processors applied to them. This allows for some very powerful filtration, masking and resizing possibilities (see http://blake.bcm.edu/emanwiki/EMAN2/Programs/e2ctf and Interlude #1 for tips).
  - In this case, we don't want to use the default 'small' option, but instead will roll our own. That means we will need to launch the generate output stage twice, once for each type of output. First, downsampled by 2 but not lowpass filtered:
    - *proctag*: **small**
    - *phaseflipproc:* **filter.highpass.gauss:cutoff_pixels=2**

        - *phaseflipproc2:* **math.fft.resample:n=2**
    - Now press *Launch*
    - When that finishes running, select *Generate Output (e2ctf)* again. This time we are going to make another set which is downsampled even more:
        - uncheck: *refinebysnr*, *phaseflip*
        - *proctag*: **tiny**
        - *phaseflipproc:* **filter.highpass.gauss:cutoff_pixels=2**
        - *phaseflipproc2:* **filter.lowpass.gauss:cutoff_freq=0.07**
        - *phaseflipproc3:* **math.fft.resample:n=3**
        - Then press *Launch* again

11. Building sets

➡ Particle sets allow you to try reconstructions with various subsets of your data without taking large amounts of disk space, while automatically keeping track of exactly what you're doing. For example, if you generate a filtered version of your particles, and mark some of them as bad, all of the other versions of the same particles will also be marked as bad. Sets also allow you to easily try reconstructions with various subsets of your data without clogging your hard drive with multiple copies of the image data.

- *Particle sets→ build particle sets*
    - check the *allparticles* and *excludebad* checkboxes
    - enter "all" int the *setname* box (you can use whatever name you like)
    - If you gave any micrographs a lower quality value when manually assessing them, you could also enter 5 in the *minqual* box, which will exclude any images with the quality set below 5.
    - *Launch*

We are going to kill 2 birds with one stone. Normally you would create a subset of the data for purposes of quickly making some class-averages for making an initial model. However, the B-gal data set we are using here has a pretty substantial number of bad particles mixed in with the good. Most of these bad particles are ice contamination which is pretty easily classified by class-averaging. So, we will make our class averages from all of the data (excluding any bad micrographs) rather than just a subset.

12. Generating reference free class averages (2D refinement)

Since we are processing all of the data, and trying to identify bad particles, we want to make a significant number of class averages to reduce the mixture of good and bad particles within a class.

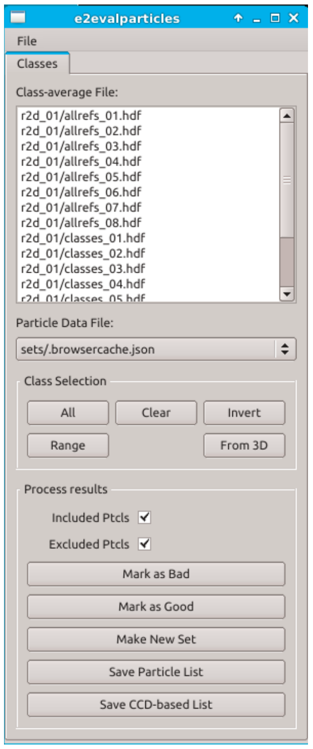
- Select *Reference free class averages → generate classes*
    - *Input* = **sets/all__ctf_flip_tiny.lst**
    - *Ncls* = ***80***, iter = **6**, *naliref* = **12**
    - parallel= **thread:N** where N is the number of cores on your computer
    - The defaults should be fine for the other options, *Launch*

➡ For most projects, this is an ideal point at which to look for structural heterogeneity in your data (not a signficant issue for the workshop sample data except for 'bad' particles). If you see several class averages apparently in near-identical orientations, but with subtly different internal features, this may be a sign that your particle is moving in solution. For example, a molecule like mammalian fatty acid synthase can be observed to move as much as 50 A in solution. It can be difficult in some cases to distinguish between variability and differences in orientation, so you may not get a definitive answer to this question at this stage. See: http://blake.bcm.edu/emanwiki/EMAN2/Programs/e2evalparticles

13. Eliminate some of the bad particles

Once class-averaging has finished, we need to take a critical look at the averages and see if we can identify any classes containing predominantly bad particles. While it is possible to look at class-averages using the browser, and even see the particles within each class by double-clicking on a class-average, if we wish to mark particles as bad, we need to use e2evalparticles.py instead.

- 2D Analysis → Mark bad particles by class
- There are no options, just press Launch
- This will cause (initially) only one window to open:



```
r2d_01/allrefs_01.hdf
r2d_01/allrefs_02.hdf
r2d_01/allrefs_03.hdf
r2d_01/allrefs_04.hdf
r2d_01/allrefs_05.hdf
r2d_01/allrefs_06.hdf
r2d_01/allrefs_07.hdf
r2d_01/allrefs_08.hdf
r2d_01/classes_01.hdf
r2d_01/classes_02.hdf
r2d_01/classes_03.hdf
r2d_01/classes_04.hdf
r2d_01/classes_05.hdf
```

sets/.browsercache.json

- The window is divided into 3 sections:
  - The upper section shows a list of all class averages generated so far in the current project, both reference-free and (if you have run a 3-D refinement) reference based
  - The middle section permits you to modify which class-averages are currently selected for operation
  - The bottom section allows you to perform various operations on the particles associated with the selected class-averages. The only option we will use right now in this section is *Mark as Bad*. See: http://blake.bcm.edu/emanwiki/EMAN2/Programs/e2evalparticles for other uses.
- At present, we need to look at the results of our 2-D reference free classification. In the top section, select *r2d_01/allrefs_08.hdf* (or whatever the highest number is).

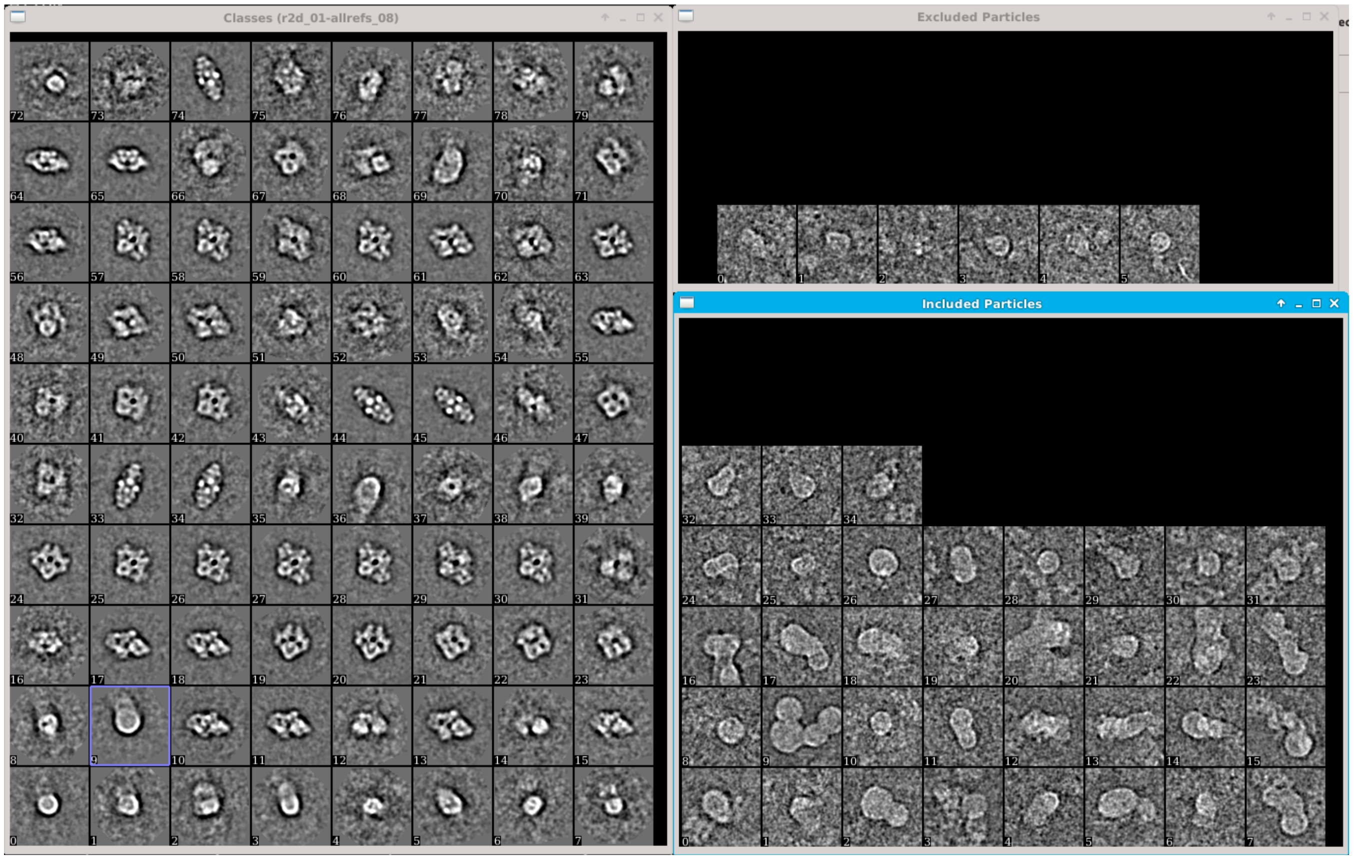
➡ While we could look at *classes_08.hdf* instead, the allrefs files contain the same averages, but sorted and aligned to make it easier to compare the different averages visually.

- This will cause 3 additional windows to open. Two of these windows, titled: *Included Particles* and *Excluded Particles*, will be empty initially. The third will show the class-averages from the selected file. Position these windows for convenient viewing of all three. It would be good to see all of the class-averages at once (maybe with some scaling). The *Excluded Particles* window can be smaller.

➡ There is a random element to the 2-D class-averaging process, so everyone will see a somewhat different set of class-averages, but everyone's averages should be at least similar to each other.

- Single-click on any of the class-averages. After a short delay, you will see particles appear in the Included and Excluded particles windows. These correspond to the particles associated with this class-average. The Included particles were used (after alignment) to produce the actual average you selected. The Excluded particles are particles which were members of the class, but looked little enough like the final average that they were excluded from the averaging process.

- We now need to identify any class-averages which consist predominantly of “bad” particles, yet do this without eliminating too many “good” particles. Below you will see my class-averages, and the particles associated with average #9.



- Hopefully it is obvious that these “particles” are actually just ice contamination. In most cases we won’t have such a clean separation and the particles will consist of some bad and some good particles. We would now like to mark these particles as “bad” so they aren’t used during our 3-D reconstruction. You should now double-click on any class-averages you think are mostly bad particles, you will see a small blue mark appear in the corner of each.

➡ This process requires some careful judgement. While we would clearly like to eliminate all of the bad particles, it is also possible to incorrectly identify some particular shape as bad particles, when it is really just an unusual orientation of the particle with a projection you weren’t expecting. If you eliminate all (or most) of the particles in some view like this, you will wind up with a significantly worse reconstruction with an anisotropic resolution, so be quite sure that you are identifying actual junk.

➡ It is often difficult for people to understand why we can’t computationally get rid of bad easily and automatically. Bad particles come in several different types. The type we are looking at above is quite obvious, and is often eliminated when selecting particles from micrographs, and is also readily excluded automatically during processing. The far more difficult sort of bad particles to identify are those which are pure (or nearly pure) noise. 2-D classification is grouping similar looking particles together, and the one clear characteristic of noise images is that they don’t look like anything, even each other. That means they will tend to be randomly distributed through any set of class-averages. Even worse, very often the good particles with the best high resolution signal are those taken quite close to focus. Since operating close to focus decreases the low resolution contrast, eliminating low contrast particles can easily have the side-effect of eliminating the very best data along with the junk. So.. not an easy problem at all!

➡ Generally speaking one bad particle like the ones shown above will do far more damage to a reconstruction than one good particle helps the reconstruction. In fact, particles like this can easily do more damage than 5 good particles would help the structure. So, if you have to sacrifice a fraction of your good particles to get rid of really bad particles, this is usually a worthwhile trade to make.

- Note that it is NOT possible to mark individual particles as bad using this interface. If you want to manually mark particles as bad, there is a section in the Appendix which explains how to do this. For now, mark all of the class-averages you wish to exclude with the little blue mark. It will not be the end of the world if you leave a few bad particles in the set. The goal is to eliminate 80-90% of the bad particles, to make our overall population healthier.
- Once you are satisfied with your blue marks, press the *Mark as Bad* button, and confirm that you want to do this in the dialog that appears. Do not exit *e2evalparticles* yet, though!

14. Selecting Good Class-averages

To make an initial model, we need a set of 15-20 good class-averages representing different views of our particle. The more different views you have, the more likely it is that your starting model will be good, and lead to a quick reconstruction.

➡ The lower your particle's symmetry, the more averages you will want to use. For GroEL, with a preferred orientation you may get away with as few as 10. For ribosomes, you might want 20-30. While larger is better, note that the amount of time the initial model generation takes is proportional to the number of averages you use, so if you are in a hurry, you may not want to go crazy and use 50 or 100 averages.

- While some of our class-averages represented bad particles, the rest should be very nice averages we can use to generate an initial model, we just need to extract them:
- Middle-click on the window showing the class-averages to bring up the control panel.
- In the control panel press the *Sets* button under the histogram, then also select the *Sets* tab (next to *Main*).
- You will see one set shown: *evalptcl*. We need to make another set for our good class-averages. Press the *New* button and type **good** in the box that appears. You should now have a green colored set with this title. Click on it (the green word *good*) to make it active.
- Now double click on class-averages you think are good. You will see a green mark appear instead of a blue mark.
- Once you have selected all of the good class-averages, press the *Save* button in the lower right corner of the control panel (not the one in the upper right). This will save the particles in the highlighted class to a specified file. Name the file **good_classes.hdf**
- You can now close e2evalparticles

15. Making an initial model

There is a lot of controversy in the cryoEM community on this point. Some people feel that initial model generation is the most critical step in refinement, and you need to use difficult and time-consuming experimental methods to get an initial model before you can proceed. In the vast majority of cases, we have shown that the simple approach used in EMAN2 can give a completely reliable initial model with no additional experiments required. However, there are a few important caveats here:

➡ When you are uncertain about the quaternary structure of your molecule, tilt validation is a critical test. You collect pairs of images at 0 degrees and typically 10-20 degrees tilt, box out tilt pairs of particles, then run them through a tilt validation procedure against your final 3-D map. This method is fully implemented in EMAN2, but we will not cover it during this workshop (there is a tutorial for this: http://blake.bcm.edu/emanwiki/OxfordWs2012).

➡ Heterogeneity is a potential issue. If you have a particle that is highly heterogeneous, the EMAN initial model strategy is likely to fail to produce a unique answer (since there isn't one). Single particle tomography may offer the best solution towards understanding the heterogeneity in your specimen. Once you understand the heterogeneity, EMAN2 includes e2refinemulti.py and e2classifyligand.py for purposes of processing such data sets using normal single particle data.

➡ Poor angular distribution. If your particles have a single, strongly preferred orientation, especially if this is combined with a low symmetry, there may not be enough information to produce an unambiguous starting model. However, it is also important to note that in this situation, even if you get a good starting model, refinement will also tend to degrade rather than improving the model. To perform a proper 3-D reconstruction, you must have a reasonable number of particles in orientations covering at least one *great circle* around the unit sphere.

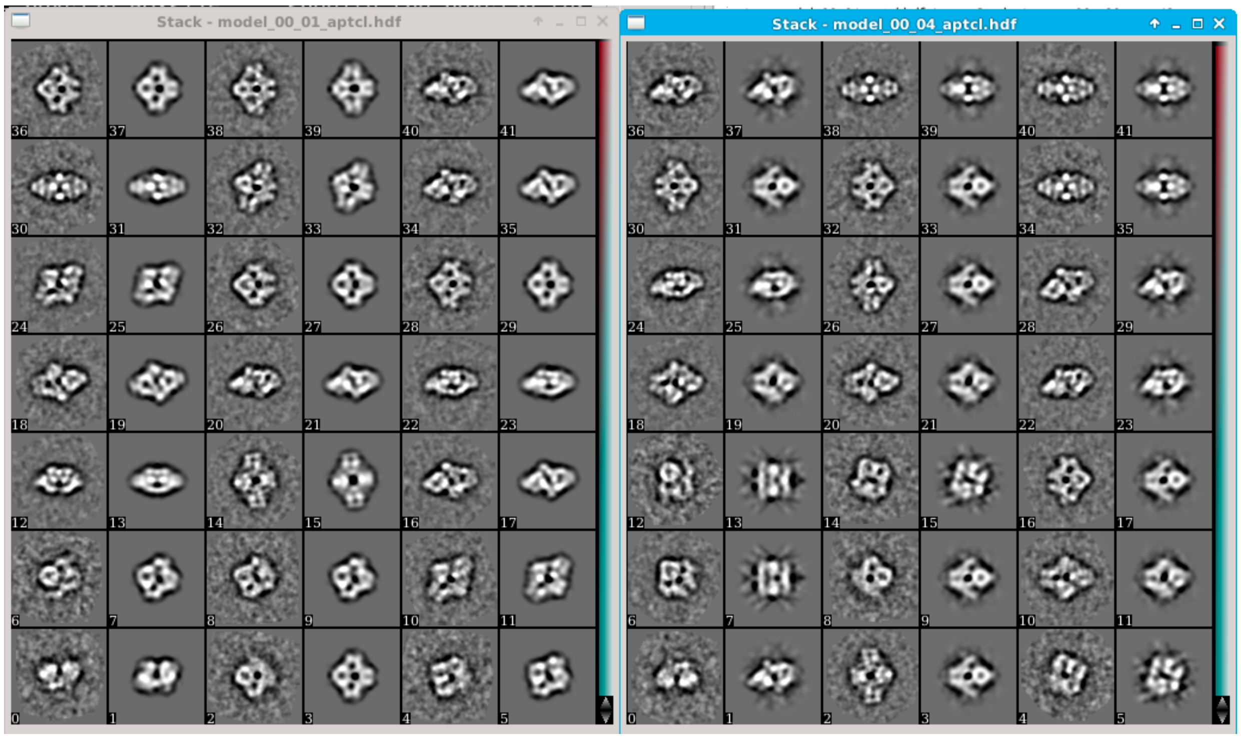
➡ If you do have a difficult structure, single particle tomography is one good solution (http://blake.bcm.edu/emanwiki/SPT/Spt). Random Conical Tilt is another possibility (http://blake.bcm.edu/emanwiki/RctTutorial).

- Despite all of the lengthly caveats and descriptions, the actual initial model generation program is quite easy to use. Just select: *Initial Model→Make Initial Model*
- This program is effectively a Monte-carlo. That is, it tries a bunch of random starting models then refines each against the class-averages. At the end, it sorts the results based on how well they agree with the inputs. In most cases ~10 tries is sufficient to get a good starting model, but sometimes you get unlucky or have a particularly tricky structure, and may need additional attempts.
- Since we are under time constraints at the workshop, I suggest starting with 10 initial models, then if you don't get one you like, run another 10. If your laptop is particularly slow, you may start with even fewer and hope for the best.
- *Input* = good_classes.hdf, *sym* = D2, *iterations* = 8
- *parallel*= "thread:N" (again, N is the number of cores on your machine)
- Use the default values for the other options, and *Launch*

➡ This program will take a few minutes to run. What it does is fairly straightforward. It treats the class-averages you provided as input as particles, generates a randomized blob as a starting model, and runs a highly optimized version of the normal EMAN2 refinement procedure to refine a structure. It does this *models* times, and in the end sorts the various output maps in order of apparent quality.

➡ For most structures, there are a number of 'local minima' in the energy space. What that means is, there are a number of incorrect structures which can still appear to agree fairly well (but not as well as the correct structure) with the input data. So, some fraction of the answers you get out are likely to be bad starting models. On the bright side, such bad starting models are usually quite obvious. The severity of this problem varies considerably with the shape of the molecule and the amount of orientation coverage you have. The B-gal structure is reasonably good in this respect, but there are still some wrong answers you will see. Interestingly, particles like ribosomes, generally viewed as 'difficult' have virtually no local minima, and will produce a usable starting model most of the time.

- When the program is done, open the browser (). You will see a new folder called *initial_models.* The program shows you all of the initial models it generated, not just the best one. For each, 4 files will be produced in the initial_models folder. We will only look at two of them in the tutorial: *model_NN_MM* and *model_NN_MM_aptcl*. The first time you run the program NN will be 00, and MM will indicate the number of the attempt. If you need to run the program again, new results will have 01 for NN.
- model_00_01.hdf ostensibly contains the best 3-D refined initial model, but looking at it is not the best way to detect whether it is bad or good. Instead, start by looking at model_00_01_aptcl.hdf. Here are two possible results, the left good, and the right bad:



- Each of these files contains adjacent pairs of images. The first image is one of the class-averages you provided and the second image is a projection of the initial model it created. If you have a good initial model, the class-averages and projections should agree with each other very well (aside from noise).
  - In the left (good) case above, you will see that this is the case every class is a good qualitative match to the corresponding projection, with the possible exception of 0-1.
  - In the right (bad) case, while several of the pairs agree well, such as 38-39, many of them do not. In a good initial model, agreement should be good for ALL of the class averages.
- If one or more do not match, there are 3 possibilities: 1) a bad initial model, 2) a bad class-average, 3) heterogeneity in the particle population. In this case 0,1 in the left model is likely

just not a very good class average, as other than this, agreement is excellent. Clearly having seen the left case, we know that the right case is just a (random) bad initial model.

- If your best model doesn't agree for more than 1 or 2 class-averages, you should try running the program again. If you persistently fail to get completely self-consistent results, the conclusion would be that you have some structural heterogeneity in your particles. We will deal with that situation in the second tutorial on heterogeneity. This B-gal data set should not have significant enough heterogeneity to be an issue.
- Once you identify a starting model with a good _aptcl file, go ahead and open the corresponding model_NN_MM.hdf file in 3-D and have a look at it. This will be the starting model we use for refinement. Note that the handedness of this model will be arbitrary. There is a 50/50 chance that it is correct. We deal with that after refinement.

16. Building sets - redux

Above we went through a lot of hassle to identify bad particles. However, so far all we have done is put a mark on the bad particles. At present they are all still included in the sets/*.lst files which we use for our refinements. This is done intentionally, so you can retain the original set of particles, in case you want to go back and test what effect eliminating the bad particles had. So, to process the data without bad particles we need to build new sets excluding the newly marked bad particles.

- *Particle sets→ build particle sets*
  - check the *allparticles* and *excludebad* checkboxes
  - enter **all-bad1** int the *setname* box, this name indicates to me that I have made one pass at removing bad particles, but otherwise the set includes everything. If I made another pass and excluded even more bad particles, I would call it **all-bad2**.
  - *Launch*

We'll make one more set containing less of the data for our initial refinement (for speed)

- *Particle sets→ build particle sets*
  - Uncheck the *allparticles* box
  - Press the *Browse* button. Click on the *snr-hi* column to sort, and select ~1/2 of the images with the highest *snr-hi* values, then click *OK*.
  - enter **best-bad1** int the *setname* box.
  - *Launch*

➡ If you aren't pressed for time, you could consider going back and repeating steps 12-13, using the all-bad1 set as input. You can then see if you are able to eliminate a few additional bad particles. Each time you repeat the process it should become more difficult to find bad particles to eliminate (diminishing returns). If you are trying to keep up with the workshop, you don't need to do this again, a single pass is perfectly adequate for a very nice reconstruction.

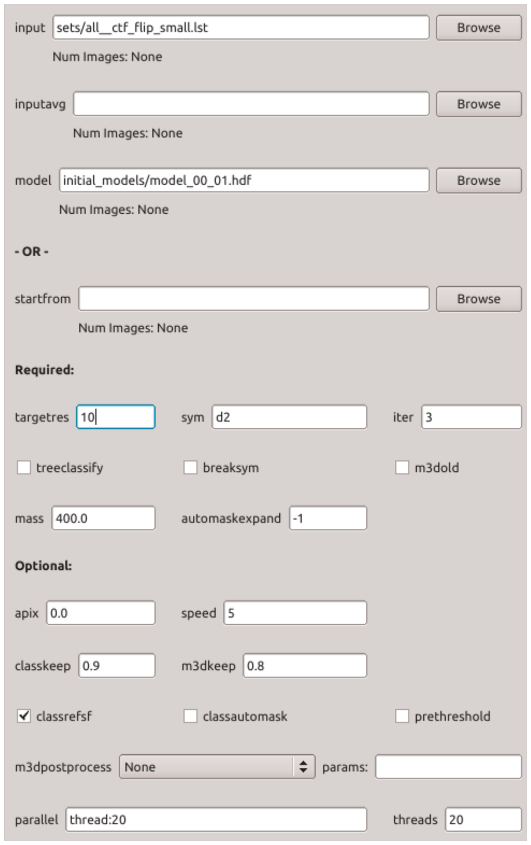
17. 3D Refinement

➡ e2refine_easy has a built in system of heuristics to automatically select the best options for your refinement. These are <u>not</u> just default values. The program actually looks at all of the information it has about your project to decide the best set of options and parameters to use. It describes the options it selected and the reasoning behind it in the refinement report it produces as described below. You can override any of these options from the command-line as required, but we strongly recommend trying the automatic system first, before making arbitrary modifications.

➡ e2refine_easy computes a ‘gold standard’ resolution as part of the refinement process, for each iteration of the refinement algorithm, and uses this information with the structure factor we already computed to near-optimally CTF correct and filter the final 3-D maps. The ‘gold standard’ does not eliminate noise bias, but it simply prevents noise bias from entering the resolution calculation, which in turn influences the filter EMAN uses.

In our first refinement we will use the data we downsampled by 2x, to make the refinement run faster. We are also using the smaller set with only ~1/2 of the particle data. The goal of this first refinement is to take the very coarse starting model (which was 3x downsampled) and quickly get something at ~15 Å or better based on the particle data directly, rather than a few class-averages. We will then immediately do a second run targeting ~7 Å with the downsampled data, followed finally by a refinement with our complete data set with full sampling, targeting ~4.5 Å resolution. The first 2 refinements should be easily possible on your laptops. The final refinement might take a full day or more on some laptops.

- *3D refinement → Single map refinement (e2refine_easy).*
- *input* → **sets/best-bad1__ctf_flip_small.lst**
- *inputavg* → leave empty
- *model* → select your good starting model (model_00_01.hdf usually)
- *targetres→***15**, *sym→***d2**, *iter→***2**, *mass→***400**, *speed→***7,** *apix→***0**
- fill in *parallel* and *threads* as previously
- *Launch*
- e2refine_easy creates an HTML (web page) report as it runs explaining the various decisions it makes and the parameters it used, as well as summarizing the results. Browse into the refine_01/report folder and select index.html. You can press the “Firefox” button (or just browse to the file in your web-browser if that doesn't work on your machine).
- The report is updated continuously, including resolution and convergence plots. You will need to reload the index.html page to display the new content.
- With these parameters, on a machine with a total speed factor of ~4.0, the refinement will take ~1 hour to complete (ie - 4 CPU-hr).



input sets/all__ctf_flip_small.lst
Num Images: None
inputavg
Num Images: None
model initial_models/model_00_01.hdf
Num Images: None
- OR -
startfrom
Num Images: None
Required:
targetres 10 sym d2 iter 3
☐ treeclassify ☐ breaksym ☐ m3dold
mass 400.0 automaskexpand -1
Optional:
apix 0.0 speed 5
classkeep 0.9 m3dkeep 0.8
☑ classrefsf ☐ classautomask ☐ prethreshold
m3dpostprocess None params:
parallel thread:20 threads 20

➡ You’ll note that the apix (Å/pixel) parameter is set to 0, not its actual value. This prompts the program to read the value from the header of the particles being used for the reconstruction. That way if the data is downsampled (as it is in this case) the *apix* is always properly adjusted. In addition, *e2refine_easy* will automatically rescale and resize the starting model based on the *apix* values it finds in the two files.

- ➡ The *speed* parameter is the primary control the user has over the refinement process (unless you want to override all of the detailed options). A speed of 1 strongly weights the heuristics towards a high quality reconstruction at optimal resolution, but pays little attention to how long it will take. A speed of 7 (the maximum) will run very quickly, but has only barely sufficient sampling for the target resolution. 5, the default should give a very solid reconstruction reasonably efficiently. Normal usage would be to use 5 for most refinements, then once you've done the very best you can with 5, perhaps try something in the 1-3 range to see if it improves your map or resolution.
- ➡ Mousing over individual parameters will give help on each, though on some OS versions the default font coloring makes these difficult to read. If you want more information on each option, run e2refine_easy.py --help from the command-line, and it will give full details on all of the options.

- The initial refinement should not take very long to run. As soon as it finishes, without even looking at the results, go ahead and start the second refinement running. Make the following changes to the last run:
  - *model* → **refine_01/threed_02.hdf**
  - *targetres→***7**, *iter→***4**, *speed→***5**
  - *Launch*

- Now that you have a real refinement running, let's quickly go over what all of these options are and what they mean:
  - *input* - this is simply the set (.lst file) containing the phase-flipped particles to be refined.
  - *inputavg* - if specified, this permits you to use a different stack of particle images for reconstruction and alignment. input is used for alignment, inputavg is used for the final class-averages and 3-D reconstruction. This is used in cases such as direct-detector movie mode processing when you want to use high-dose particles to determine orientation, but low dose (minimal radiation damage) particles for the final map.
  - *model* - Perhaps an unfortunate historical term. We refer to the 3-D map used to seed the iterative refinement as the initial model. Model is also often used to describe the PDB model derived from a high resolution map. That is not the meaning here. This file must be a 3-D map file with a valid A/pix value in the header. You do not need to resize/rescale the map. This is handled automatically.
  - *startfrom* - this is used instead of all 3 parameters above. If you have already completed a refinement run, and wish to continue without changing the input particle set, but perhaps wish to specify a higher target resolution, specify the name of an existing refine_XX folder, and it will pick up where that refinement left off. It is not possible to change the input particle stack with this option, as that would violate the assumptions for the 'gold standard' method.
  - *targetres* - This is the resolution (in Å) you are trying to achieve in this particular refinement run. If you select a much higher resolution than you can actually achieve, it will make the refinement take much longer, and in extreme cases may even make the refinement somewhat worse (not typical). If you set the resolution much lower than the provided data can achieve, it will not sample sufficiently, and you may get some odd resolution curves with apparently exaggerated resolution. If your refinement ever achieves a resolution better than your targetres, you need to rerun the refinement with a higher resolution target to have a valid result.

- *sym* - the enforced symmetry of your structure. Your initial model must be in a symmetry-aligned orientation matching the specified value here. See: http://blake.bcm.edu/emanwiki/EMAN2/Symmetry
- *iter* - Number of refinement iterations to run. Linearly related to time required for the job to run. With a decent starting model, most EMAN2 jobs will converge after ~3 iterations, and we often run 4 just to make sure we have achieved pseudoconvergence. If you are starting with a very poor starting model, such as trying to refine a ribosome from a featureless ellipsoid, you may need to run 10-12 iterations to converge to something sensible, much like the initial model generator.
- *treeclassify* - don't use this at present. It is an experimental new option.
- *breaksym* - A powerful option for studying pseudosymmetric objects, such as icosahedral viruses with portal complexes or multimeric proteins with highly homologous subunits. This will determine the orientation of each particle first with the stated symmetry. It will then search that orientation in each of the N asymmetric units for the best match. Converges much more quickly if a symmetry broken starting model is provided, but after enough iterations you can even start with a symmetrized starting map.
- *m3dold* - Not normally used. Reverts the 3-D reconstruction to an older version of the code. If you experience any strange artifacts in your 3-D reconstruction you could try turning this on. If it helps, please report the situation to sludtke@bcm.edu.
- *mass* - in kDa. However, that isn't really what this parameter means. The goal is to set 1.0 to be a reasonable isosurface threshold for the map. Historically this was often done by assuming a mean protein density of 1.35, and using the mass to compute a volume which should be enclosed by an isosurface (at a threshold of 1). However, if you were to visualize a 4 Å resolution map using this method, you would see a solid density in space that was impossible to interpret. An isosurface where the backbone and sidechains are visible is mostly open space, and has perhaps 1/2 the volume you would normally assume. So in a project targeting high resolution this mass value should normally be significantly less than the true mass of the target molecule.
- *automaskexpand* - A 'solvent flattening' procedure is a standard part of most single particle refinement procedures. That is, a mask is constructed just outside what is believed to be the protein density, the mask is softened with a gaussian falloff to the mean solvent level. This improves both refinements as well as resolution values by eliminating pure noise. However, if you have a ligand with weak fractional association, or a dynamic component on the exterior of the molecule or somesuch, it can create a weak density just outside the main molecule, which can sometimes get sliced off by this automatic masking. This parameter allows you to specify an additional number of 1 voxel shells to expand the mask beyond the normal value. Doing this may degrade the resolution slightly, but will include weak peripheral densities that would otherwise have been excluded.
- *classkeep* - the fraction of particles in each orientation class which should be included in the class-average. When making class-averages, the particles are combined, then each is compared against the just-created average. If this value is, for example 0.9, then the 10% of the particles that look least like the average will be excluded from the class-average.
- *m3dkeep* - similar to *classkeep*, but the fraction of class-averages to include when building the 3-D reconstruction. Classes are already automatically weighted based on number of particles or SSNR, but if you have some class-averages in underpopulated orientations which are really simply bad, this allows you to exclude them from the reconstruction.

- *classrefsf* - causes the class-averages to be filtered to have a similar structure factor as the projections. This doesn't actually influence the information content in the class-average, but makes them easier to visually compare with the projections.
- *classautomask* - an experimental option for performing 2-D masking of the class-averages during iterative class-averaging.
- *prethreshold* - An experimental option for applying a threshold to the 3-D map before using it as a reference for the next round of refinement.
- *m3dpostprocess* - allows you to specify a manual processor to be applied to the 3-D map at the end of each iteration. Normally not necessary as the maps are automatically filtered.

➡ Clearly we won't be running any jobs on Linux clusters at the workshop, but if you continue to do single particle analysis, eventually it is almost inevitable that you will need to do this. The final refinement below with speed=1 can take ~150 CPU-hr to finish. With the full data set that has 10x as many particles and even finer sampling, we'd be talking something closer to 1000 CPU-hr. Even if you had a powerful 24 core workstation, this is still more than a day to finish the job. On a cluster you could bring the time down to a couple of hours. There is fairly extensive documentation on the topic in the Wiki (http://blake.bcm.edu/emanwiki/EMAN2/Parallel) where you can find many of the necessary details. If you have problems, don't be shy about asking!

18. Final refinement

If everything went well, you should now have a refinement which has achieved at least ~8 Å resolution. If the resolution curve has gone beyond this point due to the high quality of the data, then you can't completely trust the resolution value (ie - the resolution value can only be trusted to a bit past the target resolution you specify), and you need to run another refinement with a better target resolution. In this case we will go ahead and switch to our final refinement targeting ~4 Å resolution.

This will be similar to the last run, but we will use the complete data set with full sampling and target higher resolution. Doubling the box size will increase the computational load by at least 4, probably more like 6 or 8 times. Doubling the number of particles will increase the time by a factor of 2. Increasing the resolution target is probably only a 20% increase. So if your previous refinement took 2 hours, we're looking at something like a full day to complete this refinement on a laptop. We're still talking something like 100 CPU-hours here, so in normal cluster computing terms, which is often measured in thousands or tens of thousands of CPU-hr, this is still a pretty small project.

While you can try running this overnight, if you want to look at the final results without the wait, we have made a copy of the refinement folder containing the full project described here available for download (but not at the workshop site). Of course this will have results of the specific choices I made rather than the choices you made, but it will give you some idea of what to expect. I included a final refinement with both speed=5 and speed=1 so you can see the range.

- *3D refinement → Single map refinement (e2refine_easy)*
- Change model to *refine_02/threed_04.hdf*
- Change input to sets/all-bad1__ctf_flip.lst

- Change *targetres* to **4.0**
- *speed* - you have a choice here. If you use **5**, you should be able to hit 4.5 Å reasonably quickly (~15 CPU-hr/iteration), and the structure will look reasonably nice. If you go all the way down to **1***,* you should readily hit 4 Å, and should have nicely resolved beta-strands with good helical pitch, but the refinement will take ~3x longer to run (~40 CPU-hr/iteration).
- *iter* can be set to *3* or *4* depending on how long you're willing to wait. Note that you will make good progress after even 1 or 2 iterations.
- Launch. If you monitor progress, make sure to look at the report in the refine_02 folder.
- This will take longer to run.
- If you want to really optimize the refinement, you can set speed to 1 and may play with some of the other options. If you really want to push resolution, though, you will need to download the full challenge data set, and work with that.

➡ Note that setting startfrom to an existing refine_xx folder is NOT equivalent to setting model to the final map from the same refine_xx folder. If you specify a single input model, it will be phase-randomized two times to relatively low resolution as starting models for the even/odd half refinements (that is, you take a step backwards if you do this). If you use startfrom, the existing even/odd references are used, and the refinement can progress naturally, with no model bias, but you are not able to change the input data set.

19. Looking at your results

Look at the final refined map (refine_03/threed_XX.hdf). You can open it in Chimera and compare it to PDB map 3j7h. If you have gotten to 4 Å resolution you should be able to resolve a lot of the strands in the beta-sheets, and the alpha helical pitch (there are only a few short helices) should be visible. If you refined a little less aggressively, you should still see the beta-sheets as planar structures with the helices as clear rods. If you can't get the maps to line up properly (using Chimera's 'fit in map' tool), you may need to flip the handedness of the final map. As mentioned earlier, the handedness of a single particle reconstruction without a tilt experiment is arbitrary. To do this, you can open the final map in *e2filtertool* and use *xform.flip* with *axis=z* to invert (then save the modified map using the appropriate menu item). Similarly you could use the command-line

➡ If your resolution curves are not "healthy". That is, they have strange peaks, don't fall all the way to zero, or have other artifacts, then some sort of bias still managed to creep its way into your project. If this happens, I'd suggest asking me about it. Sometimes you will see this in the tightly masked curve, but the curve with the normal mask will be fine. This simply indicates that the automatic tight mask was a little too aggressive, and you should ignore that curve.

Particularly if you observe any anisotropy in the resolution (if the map looks smeared or blurry in some particular direction), it is a good idea to take a look at the orientation distribution of your map. If you run *Validation and Analysis → Run e2eulerxplor*, it will give you a window showing the distribution of particle orientations within one asymmetric unit. As usual, middle-clicking on the orientation display window will give a control panel allowing you to select which iteration of which refinement to look at.

Overview/Quickstart

This is an abbreviated form of the full single particle reconstruction tutorial. The numbering follows the same scheme as the detailed tutorial above. You are welcome to try and follow this short-form tutorial, and then only skip to the above long tutorial when you get stuck.

1. **unzip Bgal.zip** (on Windows you will need to use an extraction program)
2. **cd bgal**
3. **e2projectmanager.py**
   a. NOTE - on the Mac you may see a number of OpenGL related errors appear in the terminal window where you launch GUI commands. These are related to an OpenGL bug in OSX and can be ignored. They don't impact the actual display.
4. *Project → edit project* (on the menu bar, not the task list)
   a. *Mass* = **400**, *Cs* = **2.7**, *voltage* = **300**, *apix* =**1.275**
5. *CTF → automated fitting*
   a. Check the 'allparticles' box. This will use everything in the 'particles' folder. No need to browse for files (leave that box blank)
   b. *Oversamp* = **2**, ac = **10**, *constbfactor* = **75**, *autohp* = checked, *curdefocushint* = checked
   c. *threads* = number of processors on your machine (see full tutorial)
   d. Do not use astigmatism correction until you have successfully refined without it. You can go back later and refit with astigmatism, then redo CTF->generate output, and you can then immediately rerun the refinement with astigmatism
   e. *Launch*
6. *CTF→interactive tuning*
   a. leave 'allparticles' selected. Check the 'sortdefocus' box.
   b. Fits should be good, but check to make sure defocus is right.
   c. Don't worry about low resolution mismatch, if concerned read full tutorial.

➡ The first time you run a GUI program like CTF→interactive tuning, the windows will appear in random locations on the screen. If you position them nicely, the next time you run the same program in this project, it will remember your preferred window locations.

7. *CTF→generate structure factor*
   a. Either select 10-15 of the images with the most particles (and a good defocus variation), or check *allparticles* (takes longer to run, sometimes results worse).
8. *CTF→automated fitting*
   a. Run again with same options above (this time it will use the structure factor). You only need to perform this process once. Iterating between 5 and 7 is not useful.
9. *CTF→interactive tuning*
   a. Observe the improved fit (see comments in main tutorial above)
   b. You can adjust B-factors if you like, but they aren't really used for anything at present.
   c. Remember to press 'save' or 'refit' on each image if you adjust anything.
   d. You may also want to put the plot in SSNR mode to observe data quality. Data is only useful when above ~0.02 (2% contrast, 50x more noise than signal !)
10. *CTF→generate output*
   a. Select *refinebysnr* and *phaseflip*
   b. *proctag*→**small**

c. *phaseflipproc* → **filter.highpass.gauss:cutoff_pixels=2**
d. *phaseflipproc2* → **math.fft.resample:n=2**
e. *Launch*
f. When done, *CTF→generate output a second time*
g. This time, uncheck *refinebysnr* and *phaseflip*
h. *proctag*→**tiny**
i. *phaseflipproc* → **filter.highpass.gauss:cutoff_pixels=2**
j. *phaseflipproc2* → **filter.lowpass.gauss:cutoff_freq=0.07**
k. *phaseflipproc3* → **math.fft.resample:n=3**
l. *Launch* again
m. If you go back later and refit with astigmatism, you need to redo this step, too. All of your "sets" will be immediately updated with the astigmatism corrected data after this step. No need to regenerate sets.

11. *Particle sets→ build particle sets*
    a. *allparticles* checked
    b. if you manually lowered the quality of any images during CTF processing, you can set *minqual* to an appropriate value now
    c. *excludebad* = checked (now, only a few particles have automatically been marked bad. in the next few steps we will expand the list of bad particles)
    d. *setname* = all
    e. *Launch*

12. *Reference free class averages → generate classes*
    a. *Input* = **sets/all__ctf_flip_tiny.lst**
    b. *Ncls* = **80**, *iter* = **6**, *naliref* = **12**
    c. *parallel* = **thread:N** where N is the number of cores on your computer
    d. Other options: defaults ok
    e. *Launch*. (about 30 minutes on my 4 core laptop)

13. Eliminate bad particles by identifying bad classes
    a. *2D Analysis → Mark bad particles by class* (and Launch)
    b. <u>If you are not intimately familiar with this program, strongly urge you to read the detailed tutorial on this process in step 13 above</u>.
    c. Mark classes containing mostly bad particles, then use the *Mark as Bad* button.
14. *Select good class-averages*
    a. *Using the same tool as in step 13*
    b. Open the control panel for the classes window, and create a new set called **good**
    c. Select at least 15-20 good averages into this set
    d. save this set as **good_classes.hdf**

15. *Initial Model→Make Initial Model*
    a. *Input* = ***good_classes.hdf***
    b. *Sym* = **D2**, *tries* = **8,** *iterations* = **8**
    c. *parallel*: set as above
    d. Launch
    e. When initial model generator is done, open browser and look in "initial_models"
    f. If you don't know how to evaluate the results, please read the full tutorial section above for step 15. You may need to rerun the program if the Monte-carlo didn't produce a good result on the first try.

16. Building Sets Again
    - *a. Particle sets→ build particle sets*
    - *b.* select *allparticles* and *excludebad*
    - *c.* Enter **all-bad1** as the *setname*
    - *d. Launch*
    - *e. Particle sets→ build particle sets*
    - *f. de*select *allparticles*
    - *g.* press *Browse* and select 1/3 - 1/2 of the files with the highest SSNR-hi
    - *h.* Enter **best-bad1** as the *setname*
    - *i. Launch*

*17.*Coarse refinement, *3D refinement → Single map refinement (e2refine_easy)*
    - a. input → **sets/best-bad1__ctf_flip_small.lst**
    - b. model → select your good starting model (model_00_01.hdf usually)
    - c. targetres → **15**, sym → **d2**, iter → **2**, mass → **400**, speed → **7**, apix → **0**
    - d. fill in parallel as above. Also fill in the number of cores in the "threads" box
    - e. Launch

    - f. This first refinement should not take very long to run. We need to launch one more <u>after the first finishes</u> before we have anything useful:
    - g. 3D refinement → Single map refinement (e2refine_easy)
    - h. change *model* → **refine_01/threed_02.hdf**
    - i. *targetres→***7,** *iter→***4,** *speed→***5**
    - j. *Launch* again. This one will take a bit longer

18. Final refinement, *3D refinement → run e2refine_easy*
    - a. One big refinement to go. This one may not be practical to do overnight on a laptop, though you should be able to finish an iteration or two. See main tutorial step 18 for details.
    - b. 3D refinement → Single map refinement (e2refine_easy)
    - c. Change *model* to r**efine_02/threed_04.hdf**
    - d. Change *input* to **sets/all-bad1__ctf_flip.lst**
    - e. Change *targetres* to **4.0**
    - f. *speed →* **1**-**5** (see main tutorial)
    - g. *iter* can be set to **3** or **4** depending on how long you're willing to wait
    - h. Launch.

19. Evaluate results!

Appendix

Manually marking bad particles

EMAN2.1 will automatically detect particles which are off the edge of the micrograph and mark them as bad (these particles can cause a lot of problems). There are also several places during refinement where individual particles which seem to agree poorly with other results are temporarily excluded from the reconstruction (they come back at the next iteration) The method we describe here will put a mark on individual particles which tells the set builder to exclude that particle completely from all variants of the set it creates. The ONLY place in processing where this flag is taken into account is when building sets. If you include bad particles in the set, then run a refinement, the bad particles will be part of the refinement!

- This method marks particles as bad in exactly the same way as the program in tutorial step 13 does, so if you have gone through that process, you will find a lot of bad particles already marked (and a few good ones mistakenly marked, most likely).

➡ We are (still) working on a new tool for integrated 'bad particle' marking in EMAN2. The description here is a temporary solution, which takes more effort than it should.

- Open the browser and go into the 'particles' folder. Select one of the particle stacks, and double-click it. You should see a tiled image view appear.
- Open the control-panel (middle click)
- There is a *Sets* button and a *Sets* tab in this window. Select both of them.
- If there was already a bad particle in this image, you will see "bad_particles" in the sets list. If there wasn't the list will be empty.
- Scroll through the particles and click on a bad one. "bad_particles" will appear if it wasn't there previously. Now, highlight the word "bad_particles" in this list.
- Click on other bad particles in this image stack. Click a second time if you need to unmark a marked particle.
- Continue the process with the other images in the "particles" folder. Any images marked as bad can be excluded when you build a set of particles (step 16 above). Note that this exclusion happens when the set is created. If you mark additional particles as bad after creating a set, they will only be excluded from future sets, not already created sets.