**The SIMPLE manual**

*"Keep It SIMPLE, Stupid!"*

(Kelly Johnson, lead engineer at the Lockheed Skunk Works, coined the famous KISS principle stating that systems work best if they are kept simple rather than made complex, therefore simplicity should be a key goal in design and unnecessary complexity should be avoided.)

*"Everything should be made as SIMPLE as possible, but no simpler"*

(A. Einstein)

**About SIMPLE**

SIMPLE (the Single-particle IMage processing Linux Engine) is an open source software package for semi-automated ab initio 3D reconstruction from challenging single-particle cryo-electron microscopy data (asymmetrical particles, significant degree of heterogeneity). SIMPLE focuses on the computer intensive tasks of unsupervised image classification, reference-free 3D alignment, high-resolution orientation refinement, heterogeneity analysis, and volume reconstruction. It is assumed that the windowed single-particle images represent 2D projections, and hence, that the projection slice theorem applies. In practice, this requires correction of the micrographs due to the CTF of the electron microscope and particle windowing using other software than SIMPLE. The philosophy of SIMPLE is to provide an easy-to-use interface to cutting edge image processing methods based on multivariate statistical analysis and large-scale optimization. The SIMPLE framework combines the high performance of compiled Fortran code with the ease of development via object oriented design paradigms available in modern Fortran dialects. The modular design of SIMPLE makes the generation of new methods for Fourier-based alignment and reconstruction effortless for more advanced users. Most of the SIMPLE methods are parallelized for shared-memory architectures using the OpenMP protocol. The SIMPLE suite offers potent means for efficient single-particle data processing and can be used to generate verifiable ab initio 3D reconstructions for single-particle with arbitrary symmetry in a matter of days.

**Installation of SIMPLE**

If you are compiling SIMPLE from source, the compiler options used by the compile script “simple\_compile.pl” are valid for the Intel Fortran compiler, assumed to be executed from the prompt with the command “ifort”. Currently, no other compilers are supported. Execute the compile script to get compilation instructions. To install the compiled or pre-compiled versions, execute the “simple\_install.pl” script and follow the instructions.

**PROGRAM DESCRIPTIONS**

This section of the SIMPLE manual describes the individual programs used to do ab initio 3D reconstruction, high-resolution refinement, and heterogeneity analysis. The programs are listed in the order in which they are normally executed during the reconstruction process. Each paragraph begins with an overview description of the algorithms used by the program, followed by a paragraph describing how to execute the program, and ending with a paragraph of comments describing dirty tricks, alternative execution routes, and parameter tweaking strategies. All SIMPLE programs are executed using the command line (no GUI:s here, and rest assured, there will never be!). Input is almost always a SIMPLE 2D Fourier transform stack, generated from a Spider image stack. Output consists of SIMPLE 2D Fourier stacks, Spider image stacks, Spider volumes, Spider document files, and .dat SIMPLE text files (or whatever you select to call them). The output is written directly to the working directory, which gives you complete freedom to organize a directory structure for the files in whatever way you may please. Just remember that another round of execution in the same directory will overwrite your old files. In this manual, the notation <what> denotes input input parameter “what”. <this|that> denotes alternative input parameters “this” and “that”, [<noway>] denotes an optional parameter “noway”, and {val} denotes the suggested value “val”. Command line arguments are passed as *simple=*<what>, where “*simple”* is the key in the hash used to store the parameters. You are encouraged to contact the author of this manual and bark at him if the instructions contain errors (e-mail: hael@stanford.edu). Each program prints a short version of the description presented here when executed without command line arguments.

**PROGRAM: stk\_fts**

Stk\_fts is a program for generating and stacking 2D Fourier transforms from a spider stack. Output consists of the files *outbdy.fim* (stack of transforms) and *outbdy.hed* (header).

**Usage:**

**./stk\_fts** *stk*=projs.spi *stktyp=<*spider|eman> *box=<*box size (in pixels)> *nptcls=*<nr of images in stack> *smpd=*<sampling distance (in Å)> *outbdy*=<body of output files> [*msk*=<mask radius (in pixels)>] [debug=<yes|no>]

**Comments:**

*stktyp* refers to which program has been used to generate the spider stack (the byte order may differ). If you are not sure about how the stack has been generated, set the optional parameter debug to “yes” and the program will produce a spider stack (debug.spi) with all transformations (masking & shifting) applied to the first image of the stack and halt. Make sure that the data looks healthy. If the CTF phase corrected images in the input spider stack have been centered beforehand, you should apply a soft mask (radius *msk*), otherwise run the shift alignment program shalgn\_fstk (described below) on the unmasked data and apply the mask there instead.

**PROGRAM: rev\_fstk**

Since there is a program stk\_fts for generating stacks of Fourier transforms there is naturally also a program for back transformation of the Fourier stack: rev\_fstk. Output consists of a spider image stack.

**Usage:**

**./rev\_fstk** *fstk*=<fprojs.fim> *outstk=<*outstk.spi> [debug=<yes|no>]

**Comments:**

Set the optional parameter debug to “yes” if you want to print the stack header. This can be useful for checking that the stack has been generated using the correct input parameters.

**PROGRAM: shalgn\_fstk**

Shalgn\_fstk is a program for shift alignment, which is crucial for all subsequent refinements. Most other methods in common use create an average of the stack and refine this average iteratively with respect to the origin shift parameters. This generally gives a very poor approximation of the true rotational centers, especially for particles with elongated shapes. Shalgn\_fstk instead determines the center of mass of the positive pixels in the low-pass filtered real image and shifts the Fourier transform accordingly. The outcome is sensitive to the inputted low-pass limit, which should be set to very low resolution 60-80 Å. Examine the shifted stack with program rev\_fstk to ensure satisfactory outcome.

**Usage:**

**./shalgn\_fstk** *fstk*=fprojsin.fim *outfstk*=fprojsout.fim *lp=<*low-pass limit(Å){60}> *nthr*=<nr of openMP threads> *outfile*=shiftalgndoc.dat [msk=<mask radius (in pixels)>] [debug=<yes|no>]

**Comments:**

This program is used once in order to phase shift the particle Fourier transforms so that the center of mass of the positive pixels is in the center of the real image. Please, do play around with the low-pass limit; it will make a large difference to the quality of the outcome.

**PROGRAM: classify**

Classify is a program for unsupervised classification based on agglomerative hierarchical clustering (AHC) with group average linkage (REF). The AHC solution is refined (moving element consolidation-style (REF)) using greedy adaptive local search (REF). What is different from most other AHC algorithms in use is that the low-pass limited correlation is used to control the grouping directly, and no linear compression is performed. Also, the algorithm uses a balancing constraint for generating evenly populated class sums and improve the noise robustness of the classification (REF).

**PROGRAM: abini\_rec**

Abini\_Rec is a program for reference-free 3D reconstruction based on a random initial reference generation, refinement with soft orientation assignment, and correlation weighted Fourier gridding based reconstruction. Class averages must have been generated by the SIMPLE program classify, described below, because a dimensionality reduction trick that depends on the SIMPLE 2D alignment is used to reduce the complexity of the reconstruction problem. Abini\_Rec is used for generating a low-resolution starting volume for bootstrapping refinements. Abini\_Rec is sensitive to the inputted low-pass limit, which should be kept at low resolution (typically ~30 A). However, if the data is really good one can push the limit slightly. The parameter *epsilon* is the correlation variance, assuming a Gaussian distribution of correlations, which determines the softness of the orientation assignment. I have found a value of 0.08 to work well for generation of low-resolution reconstructions. The correlation variance needs to be set high enough for the process to converge on a random initial reference, but low enough to avoid ballification of the reconstructed volume. The parameter *nbest* serves to limit the number of orientations used for weighted gridding. I have used *nbest*=10 for asymmetric reconstruction problems. However, for symmetrical molecules *nbest* needs to be set to the number of symmetry operations in the point-group times some factor n>1. Note that Abini\_Rec does not rely on common lines, and hence, it is not sensitive to the distribution of orientations. For example, reconstruction of class averages distributed in single-axis tilt geometries should be possible, but this needs to be tested thoroughly. The SIMPLE Abini\_Rec algorithm borrows the conceptual idea from the expectation-maximization-compression (EMC) framework developed by Elser and coworkers (REFS) for reconstruction free electron laser data. The main difference from EMC is that instead of modeling the exact noise distribution, Abini\_Rec uses the low-pass limited correlation and its variance to do soft orientation assignment. Repeat the procedure a few times to make sure that process converges to similar solutions. Remember that no unique solution exists for noisy class averages and that the goal here is to reconstruct a low-resolution envelope that will be used as a starting point for refinements.

**PROGRAM: evol\_align**

Evol\_Align is a program for continuous reference-based 3D alignment of individual images (REFS) given input 3D reference volumes.

**Usage:**

**./evol\_align** *mode*=<mode nr> *fstk*=<input Fourier stack (\*.fim)> *vol1*=<refvol\_1.spi> [*vol2*=<refvol\_2.spi> … etc.] *lp*=<low-pass limit (in Å)> [*fromp*=<start ptcl index>] [*top*=<stop ptcl index>] [*trs*=<half interval of origin shift (in pixels), default 0>] [outfile=<output text file>] [*oritab*=<input orientations and state assignments (\*.dat)>] [*pgrp*=<cn|dn>] [*nthr*=<nr of openMP threads>] [*hp*=<high-pass limit (in Å)>] [nbest=<population size DE, default 100>] [debug=<yes|no>]

**Comments:**

Available modes are:

**mode=20:** alignment with fixed low-pass limit (*lp*), no input orientations required

**mode=21:** alignment with spectral self-adaptation, *lp* now puts a lower bound on the dynamically adjusted low-pass limit.

**mode=22:** mode 21 with orientation keeping

**mode=23:** for finding filtering threshold or do spectral sorting, input orientations are required

**mode=24:** for calculating spectral common lines-based score for a set of aligned projections, no volumes required

**mode=25:** same as 22, but with neighborhood refinement only

The strategy is to start off on the starting volume(s) with a very low-resolution low-pass limit (*lp* typically set to 30-40 Å) and a rather large shift (*trs* set to 5). However, with a broad shift search, the search space increases dramatically in size. In order to compensate for this, we need to increase the population size of the evolutionary algorithm (*nbest* set to 200). After the first round, volumes are reconstructed with rec\_master (described below), before shifting the Fourier transform stack with the shifts obtained. There are no interpolation errors associated with this operation, so if data is poorly centered, repeat the shifting procedure a couple of times. I can assure you that it is worth the effort; with well-centered data the refinement will converge much faster. With decrease the shift range inputted (*trs*=1-3) the performance of the optimizer can be substantially improved.

**PROGRAM: rec\_master**

Rec\_master is a program for reconstructing volumes from a Fourier transform stack (\*.fim) given input orientations and state assignments. The algorithm is based on a Fourier gridding technique that uses pre-weighting with a Gaussian window function to account for the finite extent of the interpolation window. The pre-weighted signal is interpolated using the ideal sinc convolution kernel. The feature sought when implementing this algorithm was to enable quick, reliable reconstruction with the possibility to apply particle dependent weights. This feature will be used heavily in future SIMPLE releases to do soft assignment in the continuous orientation refinement and improve noise robustness.

**Usage:**

**./rec\_master** *fstk*=<fstack.fim> *vol1*=<recvol\_1.spi> [*vol2*=<recvol\_2.spi> … etc.] *oritab*=<algndoc.dat> *amsklp=*<mask low-pass limit (in Å)> [*nthr*=<nr of openMP threads>] [msk=<mask radius (in pixels)>] [debug=<yes|no>]

**Comments:**

<input comments regarding filtering of ptcl images>

**PROGRAM: het\_master**