# The  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 sets (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 contrast transfer function 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 methodological research oriented toward Fourier-based alignment and 3D reconstruction in single-particle imaging 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

To install the compiled or pre-compiled versions, execute the “simple_install.pl” script and follow the instructions. 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.

## Program descriptions

This section of the SIMPLE manual describes the individual programs that are used in concert to accomplish 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 and output consists of SIMPLE 2D Fourier stacks, Spider image stacks, Spider volumes, Spider document files, and .dat 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 in whatever fashion you 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 the 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.

## Command line dictionary

| | |
|---|---|
| *angres* | angular resolution in the plane (in degrees) |
| *box* | image size in pixels (image assumed to be a square *box*box* array) |
| *fromp* | from particle index |
| *fstk* | SIMPLE Fourier stack name (*.fim suffix required) |
| *hp* | high-pass limit (in Å) |
| *lp* | low-pass limit (in Å) |
| *maxits* | maximum number of iterations |
| *maxp* | maximum number of particle images in a class |
| *minp* | minimum number of particle images in a class |
| *mode* | mode of operation for certain multitasking programs |
| *msk* | circular or spherical mask radius (in pixels) |
| *mw* | molecular weight (in kD) |
| *nbest* | population size for the optimization |
| *ncls* | number of classes |
| *nptcls* | number of particle images |
| *nrnds* | number of restart rounds |

| | |
|---|---|
| *nspace* | number of projection directions in search space |
| *nthr* | number of openMP threads |
| *nvars* | number of eigenvectors |
| *oritab* | SIMPLE text file with orientations and state assignments |
| *outbdy* | body of output files, if file==“boring.job”, then “boring”==bdy |
| *outfile* | SIMPLE output text file |
| *outstk* | output spider image stack (*.spi suffix required) |
| *pgrp* | point-group symmetry |
| *smpd* | sampling distance (in Å) |
| *stk* | spider image stack name (*.spi suffix required) |
| *to* | to particle index |
| *trs* | origin shift search range parameter, search range is [-*trs*,*trs*] |
| *vol1* | spider volume name (*.spi suffix required) |

## Program: classify

**Classify** is a program for unsupervised classification based on rotational alignment (REF), iterative principal component analysis for generation of feature vectors (REF), and 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).

## Usage:

**./classify** *stk*=<spistackin.spi> *box*=<box size (in pixels)> *nptcls*=<nr images in stack> *smpd*=<sampling distance (in Å)> *msk*=<mask radius (in pixels)> *lp*=<low-pass limit (in Å){30-40}> *ncls*=<nr classes{*nptcls*/40}> *minp*=<minimum nr ptcls in class{20}> *maxp*=<maximum nr ptcls in class{80}> *nvars*=<nr eigenvectors{20-40}> *nthr*=<nr openMP threads> [*oritab*=<algndoc.dat>] [*doalign*=<yes|no>] [*hp*=<high-pass limit (in A){100}>] [*debug*=<yes|no>]

## Comments:

The program generates a stack of Fourier transforms and rotationally aligns it using low-pass limited (by parameter *lp*) correlation search. Use only low-resolution terms in this operation (30-40 Å). Fourier interpolation is used for rotation of the stack before inverse Fourier transformation and extraction of the pixels within the circular mask defined by mask radius *msk*. Iterative principal component analysis (PCA) generates feature vectors from the vectors of extracted pixels. The number of classes is a critical parameter that I will attempt to determine automatically in future SIMPLE releases by using Bayesian methods for model selection. For now, I recommend using a value around *nptcls*/40. The hierarchical classification of the feature vectors uses a balancing constraint (*maxp*). The minimum class population (*minp*) is not used by the classification procedure. The *minp* parameter is only used to exclude classes below a certain population from being represented by an average. The number of eigenvectors is a critical parameter that will be targeted for automatic adjustment in future releases. The good news is that it is easy to test

different settings, since the method used for doing PCA is iterative and VERY fast. As a rule of thumb, use 20 eigenvectors if you do are not inputting in-plane parameters generated by program **align** (described below) and use 40 eigenvectors if you do input in-plane parameters. As will become clearer later, in-plane parameters from the refinement will at some stage be inputted to improve the quality of the classification and resolve heterogeneity in 2D. Note that the classification correlation matrix is kept in RAM, so for large data sets you need LOTS of internal memory (I have managed to classify 65,000 images on a machine with 96 GB RAM). This quirk will also be addressed in future releases. The optional parameter *oritab* is used to provide in-plane parameters to the program. This option is used for generating class averages that are going to be subjected to heterogeneity analysis (see program **resolve** below). Set the optional parameter *doalign* to "no" if the data has already been in-plane aligned somehow. An optional high-pass limit (*hp*) is included for those working with very large assemblies and wants to exclude significant amounts of information in the lower frequency bands. The default high-pass limit is set to the second Fourier index.

## Program: abinirec

**Abinirec** is a program for reference-free 3D reconstruction. The algorithm is based on a random initial reference generation, deterministic annealing-based refinement with soft orientation assignment, and correlation weighted Fourier gridding-based reconstruction. Abinirec is used for generating a low-resolution envelope for bootstrapping refinements. Abinirec is sensitive to the choice of low-pass limit, which should be kept at low resolution (typically 30-40 Å). However, if the data is really good one can push the limit slightly. Note that abinirec no longer relies on common lines, and hence, it is not that 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 abinirec algorithm borrows the conceptual idea from the expectation-maximization-compression (EMC) framework developed by Elser and coworkers (REFS) for reconstructing free electron laser data. The main difference from EMC is that instead of modeling the exact noise distribution, each iteration of abinirec uses the low-pass limited correlation to define a local neighborhood of 'best' orientations. Soft orientation assignment is done in this neighborhood assuming a Gassian distribution of correlation values and unit correlation variance. Annealing is introduced by varying the size of this neighborhood from large to small. The procedure should be repeated a few times to ensure convergence to similar solutions. Remember that no unique reconstruction 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.

## Usage:

**./abinirec** *fstk*=cavgstk.fim *lp*=<low-pass limit(in Å){20-30}> *msk*=<mask radius in pixels> *maxits*=<maximum nr of iterations{6-10}> *nthr*=<nr of openMP threads> [*vol1*=<startvol.spi>] [*nbest_start*=<start neighborhood size>] [*nbest_stop*=<stop neighborhood size>] [*hp*=<high-pass limit (in A){100}>] [*debug*=<yes|no>]

## Comments:

This is new software that has not yet been published. I have used it to reconstruct simulated as well as experimental data of RNA polymerase II and the ribosome. The algorithm seems to be more robust towards noise and skew orientation distributions than our earlier common lines-based procedures, but the results obtained so far are of very similar quality. The main advantage is speed. The possibility to do multiple restarts on random starting configurations in feasible time aids consistency evaluation and may open up for efficient and robust validation procedures in future SIMPLE releases. Since each Fourier plane (class average) is assigned to many orientations with correlation-dependent weights, the reconstructed surface is constrained to be smooth and noise artifacts are reduced. The process can be restarted on previously generated volumes via optional parameter *vol1*. This may come in handy if one wants to be brave and test a low-pass limit of higher resolution or modify the neighborhood evaluation. Optional parameters *nbest_start* and *nbest_stop* give starting and stopping values for the neighborhood size. The default neighborhood size start and stop values are 30 and 10, respectively.

# Program: cenvol

**Cenvol** is a program for centering spider volumes. This may be needed after starting model generation by program abinirec, before entering the refinement stage. Cenvol low-pass filters the volume (limit *lp*) and binarizes it according to the molecular weight given. The center of mass of the binary volume is determined and the result obtained is used to shift the Fourier representation of the original volume. Back Fourier transformation generates a centered volume.

## Usage:

**./cenvol** *vol1*=<invol.spi> *box*=<box size (in pixels)> *smpd*=<sampling distance (in Å)> *lp*=<low-pass limit (in Å){15-30}> *mw*=<molecular weight (in kD)> [*msk*=<mask radius (in pixels)>] [*debug*=<yes|no>]

## Comments:

Play with low-pass limit and molecular weight if you are not satisfied with the outcome.

# Program: spi_to_fim

Having generated a starting model it is time to prepare our images for Fourier-based alignment. **Spi_to_fim** 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:

**./spi_to_fim** *stk*=<spistackin.spi> *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:

Setting the optional parameter debug to "yes" will produce a spider stack (debug.spi) with all transformations (masking & shifting) applied to the first image of the stack.

# Program: fim_to_spi

Since there is a program spi_to_fim for generating stacks of Fourier transforms there is naturally also a program for back transformation of the Fourier stack: **fim_to_spi**. Output consists of a spider image stack.

## Usage:

**./fim_to_spi** *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: align

Align is a program for continuous reference-based 3D alignment of individual images, given input reference volumes. The algorithm is based on an advanced differential evolution (DE) algorithm for continuous global optimization (REFS). Previously, SIMPLE used a Fourier-based interpolation scheme that extracted common lines between reference central sections and the particle section (REF). In this release, the Fourier-based interpolation scheme has been reconciled with that used in Frealign (REF). The entire reference section used for matching is now interpolated directly from the 3D Fourier volume, which requires only little extra computation and gives an additional dimension contributing to the interpolation. In addition, this scheme increases the number of Fourier components used to calculate the correlation as compared to the polar common lines-based representation. These factors are important for noise robustness of the orientation search. The method of spectral self-adaptation is applied to estimate a particle dependent low-pass frequency limit (REF). Align offers two modes of interpolation: (1) direct sinc or (2)

Gaussian pre-weighted sinc. Interpolation mode (2) is somewhat slower but reduces the interpolation artifacts that may limit resolution progression in the later refinement rounds. The mode of interpolation is under automatic control by the low-pass limit. If the resolution limit of a particle image is estimated to be better than 20 Å, the more costly and accurate interpolation kernel is used.

## 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*=<SIMPLE text file with input orientations and state assignments>] [*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** for multi-reference alignment with fixed low-pass limit (*lp*), no input orientations required, **mode=21** for multi-reference alignment with spectral self-adaptation, *lp* now puts a lower bound on the automatically estimated low-pass limit, **mode=22** for mode 21 with neighborhood refinement, Q improvement is enforced, and **mode=23** for finding filtering threshold or do spectral scoring, input orientations are required. The strategy is to **begin with mode=20 alignment** on the starting volume(s) using a low-resolution low-pass limit (*lp* typically set to 30-40 Å). In the first rounds, the origin shift parameters will be far from optimal. If we apply a broad shift search, the search space increases dramatically in size. In order to compensate for this, an optimization method that is tailored for shift alignment is used by mode=20. The mode=20 method is based on multiple random restarts on the same grid of translations, obtained by distributing translation vectors evenly on a polar grid, and initializing the Euler angles by uniform random assignment. Due to complexity limitations of the DE optimizer, the translation search range is limited to [-5,5]. In fact, this is the only applicable range of shift search in mode 20. The user should make sure that a significant portion of the solutions does not lie on the shift interval borders. If this is the case, the Fourier stack should be shifted (using program **shift_fim**, described below) and the mode 20 search re-run. After completing the shift alignment and the first orientation assignment, volumes are reconstructed with program **reconstruct** (described below). In order to bring the projections as close as possible to their origin of rotation, the Fourier stack is again shifted after reconstruction (using program **shift_fim**, described below). This is of crucial importance, as it reduces the complexity of the optimization problem and improves significantly the quality of the solutions obtained in later refinement rounds. **In subsequent refinement rounds, mode=21-based alignment** is combined with reconstruction until quasi-convergence. The shift range is now limited to [-3,3]. It is certainly worth the effort to shift the Fourier stack halfway trough the refinement and further narrow the origin shift search to [-1,1]. The optimization strategy in mode=21-22 is designed to

make use of the fact that we have spent considerable computation on finding good origin shift parameter. The process begins with an exhaustive angular search in a discrete angular space, while keeping the origin shift parameters fixed at their old values. The 100 best exhaustive solutions are used to initialize continuous DE optimization with the population of origin shift parameters assigned to a random Gaussian distribution around their old values. **When the mode=21 refinement has converged, as measured by the outputted Q values or by the Fourier Shell Correlation (FSC) plot calculated between reconstructions from successive rounds, it is time to apply mode=22-based refinement.** Mode=22 searches a neighborhood of the quasi-converged solution. The search is further constrained by enforcing improvement in Q. In the mode=22 refinement one may therefore experiment with increasing the resolution of the lower bound of the resolution of the low-pass limit (*lp*). However, **it is never a good idea to increase the resolution of *lp* beyond the best automatically estimated limit**.

## Program: shift_fim

**Shift_fim** is a program for shifting a SIMPLE Fourier stacks according to the origin shifts printed in alignment documents produced by program align (described above). A linear phase shift is applied in Fourier space, so there are no interpolation errors associated with this operation. Happy shifting!

## Usage:

**./shift_fim** *fstk*=<fstack.fim> *outfstk*=<shifted_fstack.fim> *oritab*=<algndoc.dat>

## Comments:

Note that shift_fim produces a new text file of orientations in which the translations are zeroed. Use this file as an input for mode=21-based alignment in program align (described above). If you plan to execute another round of mode=20-based alignment or move into classification and heterogeneity analysis you need not to care about this file.

## Program: reconstruct

Reconstruct is a program for reconstructing volumes from a SIMPLE Fourier transform stack (*.fim) given input orientations and state assignments (obtained by program align). The algorithm is based on a Fourier gridding technique that uses pre-weighting of the sinc window with a Gaussian window function to account for the finite extent of the interpolation window. This reduces the real-space ripple artifacts associated with direct moving window sinc interpolation. The feature sought when implementing this algorithm was to enable quick, reliable reconstruction with the possibility to apply particle dependent weights (as is done in programs abinirec and resolve). An additional bonus is that the gridding method is very easy to parallelize. For really high-resolution work I would replace the gridding with a SIRT (Simultaneous Iterative Reconstruction Technique) algorithm,

such as the one implemented with the 'bp cg' command in Spider. Perhaps one sunny day I will implement a SIRT code in SIMPLE.

## Usage:

**./reconstruct** *fstk*=<fstack.fim> *vol1*=<recvol_1.spi> [*vol2*=<recvol_2.spi> … etc.] *oritab*=<algndoc.dat> *nthr*=<nr of openMP threads> [debug=<yes|no>]

## Comments:

The number of input volume file names must be equal to the number of states.

# Program: automask

Auto_mask is a program for doing solvent flattening of an input spider volume.

## Usage:

**./auto_mask** *vol1*=<invol.spi> *amsklp*=<mask low-pass limit (in Å)> *box*=<box size (in pixels)> *smpd*=<sampling distance (in Å)> *voltyp*=<spider|simple> [msk=<mask radius (in pixels)>] [debug=<yes|no>]

## Comments:

The algorithm for background removal is based on low-pass filtering. First, the volume is low-pass filtered to *amsklp* before clustering the voxels into two groups using the k-means algorithm and assigning them background (=0) and foreground (=1) values. Low-pass filtering to *amsklp/2* before multiplication with the input volume softens the sharp-edged envelope. The mask and the masked volume are written to files. *voltyp* refers to which program has been used to generate the spider volume (the byte order may differ).

# Program: resolve

**Resolve** is a program for heterogeneity analysis.

## Usage:

./resolve fstk=cavgstk.fim nstates=<nr of states to resolve> lp=<low-pass limit (in A){15-30}> msk=<mask radius in pixels> maxits=<maximum nr of iterations{6-10}> nthr=<nr openMP threads> [vol1=<startvol.spi>] [nbest_start=<start neighborhood>] [nbest_stop=<stop neighborhood>] [hp=<high-pass limit (in A){100}>] [debug=<yes|no>]

## Comments:

The algorithm for

./dock vol1=invol1.spi vol2=invol2.spi box=100 smpd=2.33 lp=<low-pass limit(in

A){15-30}> mw=<molecular weight(in kD)> msk=<mask radius(in pixels)> nthr=<nr o
f openMP threads> nrnds=<nr of rounds> [nspace=<nr of projection directions>] [
angtres=<angular resolution in the plane>] [debug=<yes|no>]

## Strategies

### Stage 1: Starting model generation

### Stage 2: Refinement

### Stage 3: Heterogeneity analysis