Basic Installation

Your Python version must be 2.7 or newer and you'll need numpy and the python image library (Pillow). Download both from https://www.enthought.com/products/canopy via either the free or academic version of Enthought Canopy.

Method 1:

- Install by running `git clone https://github.com/samedling/MCSAS.git` to download the entire repository.
- For future updates, run `./mcsas -u`. (Or run `git pull origin master` from the MCSAS folder and possibly recompile).

Method 2:

- Go to https://github.com/samedling/MCSAS and click Download ZIP in the lower right to install. Update the same way.

There are two optional but recommended ways of speeding up the code:

1. Install PyOpenCL following the directions at http://wiki.tiker.net/PyOpenCL/Installation and then the first time your run it, it will ask you which platform and device you want to use. Try the GPU first; if it doesn't work, use the CPU. On my dual core CPU, I obtained a 25x speedup; quad core CPUs should be nearly twice as fast and GPUs should be even faster!
2. F2Py acceleration should work automatically on OS X or Linux. If not, or if you want maximum performance and you have gfortran installed, run `make` to compile the Fortran code (or if you have ifort installed, edit the makefile before running `make`). On my dual core CPU, I obtained a 10x speedup; quad core CPUs likely not much faster.

Run `./mcsas` on the command line or open it in Canopy and click run. (Note: you may discover running `nice ./mcsas` results in your system being a lot more responsive.)

If you want to try out the latest features, you can try the develop branch (by either running `git pull origin develop` or via the web GUI clicking on 'master', selecting 'develop', and then downloading the zip) but consider editing global_vars.py so debug = False.

Python Installation

If you're trying to install the required python packages on a computer you don't have root access on, run these commands:

cd mkdir python echo "export PYTHONPATH=$PYTHONPATH:~/python" >> .bashrc pip install --target=python Pillow

OS X Fortran Installation

Apple doesn't provide a recent version of gfortran, but you can download one from http://hpc.sourceforge.net

If you also don't have Xcode Tools, follow the directions at https://wiki.helsinki.fi/display/HUGG/Installing+the+GNU+compilers+on+Mac+OS+X

1. Install XCode Tools from the App Store.
2. Install the Command Line Tools by running xcode-select --install
3. Download the latest stable gfortran version from http://hpc.sourceforge.net

Tested on OS X 10.10 "Yosemite".

Some Troubleshooting

If things ran fine before, but after updating returns KeyError, remove the file called default.txt.

OS X and Windows: You need to close all the old plots before you can run things again. Otherwise, it's otherwise unresponsive for some reason.

OS X/EPD/Tkinter: Make sure you have Canopy. EPD might tell you it's updated everything, but it's still not the same as Canopy.

If you receive compiler OpenCL compiler warnings when starting the program it's probably due to your OpenCL device not supporting 64-bit floating point numbers; it should be fine, but if you get errors later, try using a different OpenCL device.

Linux/OpenCL: apt-get on Ubuntu 14.04 wasn't helpful to me; follow the directions linked above for more success.

PIL: On older systems you may need to manually remove PIL and install Pillow (`sudo pip uninstall PIL` and `sudo pip install Pillow`); newer systems should simply come with Pillow. Otherwise Image won't be able to read the funny SAXS TIF files.

Scientific Linux/F2PY: Make sure you are using the version of F2PY which matches your version of Python (2.7+), otherwise just loading the Fortran module causes a Segmentation Fault.

Fortran/Hyperthreading: If you have a Core i7 processor, (or other CPU with hyperthreading) performance may be slightly improved by adding `export OMP_NUM_THREADS=<n>` to your .bash_profile (where n = the number of physical cores).

Due to the large number of shapes, it's possible some of the sped-up versions of these have bugs. If you find that the shapes don't look right, edit the global_vars.py file so `accelerate_points = False` to disable the erroneous speedup.

Running the Program

Individual Monte Carlo Calculations

After selecting the Monte Carlo model, **press the Parameter Help button** to rename individual model parameters to be more descriptive and grey out unused parameters.

'Real Space' will show you the points.

'Calculate Intensity' will show you the detector image.

To activate the advanced options, click on the Pop-Up Window buttons at the right.

If the Save Images box is checked it automatically overwrites anything with the same filename.

- The Radial Symmetry checkbox speeds the program, so check if appropriate. The Small Angle Approximation checkbox doesn't speed things up, so only check if necessary.
- Neighboring point distance and z-direction scaling can provide speedups at the expense of decreased background contrast; doubling the neighboring point distance decreases the number of points by a factor of 8, while doubling z-direction scaling decreases it by a factor of 2.
- **Opening multiple copies of the same window is not recommended.** If you do, it will use the values in the one you opened most recently but it will constantly reopen uneditable old ones. Or sometimes it will produce an error until you close them and reopen one. Either way you'll get confused, so avoid opening more than one of each window.
- Taking into account a non-infinte coherence length really complicates things. By default, the code uses a rough approximation where the object is broken into chunks. This should work reasonably well, but if you want to really calculate it the long way, there's a button for that in Detector Options and I recommend no more than 2500 (50x50) pixels and 5000 points if you want it to take less than an hour.

Performing Fits

1. Input the name of the experimental data file (relative to the directory the program is in) to fit and click "Plot Exp Data". If "Center of Beamstop" is left blank ("0 0") then it will plot the original experimental data (which takes a minute). The lower bounds option in the center column is quite useful here. Try a value in the range 1e-8 to 1e-6. Then, move the mouse over the center of the beamstop and read the x,y-coordinates from the plot screen. Use these values and replot the experimental data. It will crop a sqaure around the center and downsample it so the side length is equal to the Pixels parameter.
2. Input known values, uncheck relevant parameter boxes, make a good guess of unknown parameters. To see how good your guess is, press "Calc Residuals".
3. When you have a satisfactory guess, click "Fit Exp Data". Each iteration, it prints out the time and the sum of the residuals; be aware that it is normal for the sum of the residuals to go several iterations without changing

significantly.
4. **Read the fit results from the terminal.** If you had a grid compression >1 and now you want more printable results, copy the fit results back into the GUI and Plot Residuals. Or run again to verify you aren't in a local minimum.
5. Images are not automatically saved, so save them yourself.

Some comments: *Grid compression only works with reliably with fortran; it works with OpenCL if the number of points/pixels does not exceed OpenCL's capabilities.* If the fit steps are each taking less than 10 seconds, there would probably be very little additional time taken by increasing pixels by 40% or halving the grid compression or z_scaling. *Be careful when letting the background vary. If the background gets set too high by the computer, it cannot normalize properly and all bets are off.* Checking the radial symmetry box decreases the time taken by 30-50% (30% for Fortran, 40% for OpenCL, 50% for pure Python). * When fitting or calculating residuals, the settings in the Sequence Options window will affect your fits. It's relatively flexible in that if you set the length to 100nm in the main window, but tell the length to vary from 180 to 220 in the sequence window, it will use lengths of 90 to 110.

Adding Models

First, make sure you have the most recent version.

To add a Monte Carlo model:

Open density_formula.py

1. In density(coords), at the top, created another elif line at the bottom like all the others.
2. Add an element to the end of g.model_parameters. Note, you'll need to put a comma at the end of the previous last line and then put in useful descriptors for each of the parameters you are using (to show the user when they click the help button). (Do not add to the middle of the list as this will cause errors when Fortran or OpenCL are enabled.)
3. Create a new density function d##name near the bottom using one of the two templates at the very bottom.

Save and close the file.

Optionally: If you want to speed up the calculation using Fortan, edit fastmath.f90 with a Fortran version of your new density and edit the elif blocks in Points_For_Calculations() in Monte_Carlo_Functions.py. If you want to speed up the calculation using OpenCL, edit density.cl with an OpenCL version of your new density and edit the elif blocks in opencl.py, class OpenCL, def density().

To add an analytic model:

1. Open analytic_formula.py. Near the bottom of the file but above where theory_csv is defined, create another "elif:" block like the ones above it. Remember the number you assigned. Save and close the file.
2. Open newgui.py and go to the line defining Analytic_options. After the last

number (currently around line 90), add a line like the ones above it using the number from step 1. Save and close the file.

Copyright