DRAFT: Iterative Algorithm Tutorial Notes

2024-07-30

▼ 1. Create RACE env.

Open a new AWS instance (Ubuntu).

NOT the Ubuntu for Development

Use C08 General - 2xLarge

8 vCPUs / 32 GiB Memory

Up to 3.8 GHz 4th generation Intel Xeon Sapphire Rapid processor.

Estimated Cost: 0.50 USD/Hour (12.10 USD/Day)

and 512GB Storage.

▼ 2. Set up RACE env. (OPTIONAL)

Update and upgrade

```
sudo apt update
sudo apt upgrade
```

Install github cli tool

```
sudo apt install gh -y
gh auth login
```

Install zsh and OhMyZsh

```
sudo apt install zsh -y
sh -c "$(curl -fsSL https://raw.github.com/ohmyzsh/ohmyzsh/
```

Get vimrc, zshrc and tmuxconf

```
git clone https://github.com/YellowSub17/zshrc.git
git clone [https://github.com/YellowSub17/vimrc.git](https:
git clone https://github.com/YellowSub17/tmuxconf.git
mv vimrc/vimrc ./.vimrc
mv zshrc/zshrc ./.zshrc
mv tmuxconf/tmux.conf ./.tmux.conf
```

Open and install vim plugins

```
vim  # to download and initalize ~/.vim directory
mv vimrc/hi.vim ~/.vim/
sudo apt install cmake -y
python3 ~/.vim/plugged/YouCompleteMe/install.py
```

Open and install tmux plugins

```
tmux #(to download and initalize ~/.tmux directory)
git clone https://github.com/tmux-plugins/tpm ~/.tmux/plugi
sudo apt install xsel #requirement for tmux-yank
tmux #(install plugins with ctrl+B ; I)
```

Install conda

```
mkdir -p ~/miniconda3
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-
bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3
rm -rf ~/miniconda3/miniconda.sh
```

NOTE:

If you grabbed the `.zshrc` file, you will need to change paths from `/home/pat/` to `/home/ec2-user` .

You will also need to comment out the last line that activates an environment that doesn't exist.

Example: `#### conda activate env`

You can also comment out irrelevant alias commands.

Example: `#### alias phd_pp='cd ~/Documents/phd/python_projects'` and `#### alias envp='conda deactivate;conda activate phd;clear'`

Reboot the RACE instance

```
sudo reboot
```

▼ 3. Install `scorpy`

Install `conda`

Follow the `conda` install instructions for package management and version control.

https://docs.anaconda.com/miniconda/

```
mkdir -p ~/miniconda3
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-
bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3
rm -rf ~/miniconda3/miniconda.sh
```

Initialize `conda` in your shell.

```
~/miniconda3/bin/conda init bash
~/miniconda3/bin/conda init zsh
```

Setup a `conda` environment

Create and activate the environment

```
conda create -n scorpy python==3.9 --yes
conda activate scorpy
```

Install the required packages

```
conda install -c conda-forge numpy==1.26.4 --yes
conda install -c conda-forge matplotlib==3.8.4 --yes
conda install -c conda-forge pyshtools=4.12.2 --yes
conda install -c conda-forge h5py==3.11 --yes
conda install -c conda-forge pycifrw==4.4.6 --yes
conda install -c conda-forge scipy==1.13.1 --yes
conda install -c conda-forge scikit-image==0.24.0 --yes
conda install -c conda-forge regex==2024.5.15 --yes
conda install -c conda-forge numba==0.60.0 --yes
conda install sqlite==3.45.3
```

Download and install `scorpy`

```
git clone https://github.com/YellowSub17/scorpy-pkg.git
cd scorpy-pkg
pip install -e .
```

▼ 4. Set up data

Now we are ready to run scorpy. Download the algorithm tutorial example repo into the home directory. Then set up the data directory, and download some structure factors of lysozyme to correlate.

```
cd ~
git clone https://github.com/YellowSub17/algo-tute
mkdir -p ~/algo-tute/data/algo
wget https://files.rcsb.org/download/193L-sf.cif -O ~/algo-
```

▼ 5. `make-algo-tag.py`

The first script we will run is `make-algo-tag.py`. It will make a directory with in `data/algo` with the tag name specified in the file.

```
python make-algo-tag.py
```

The start of this scripts sets various algorithm parameters that are constant within an algorithm tag group. The last command initializes the `AlgoHandler` class and makes the tag directory, and saves a text file with the parameters.

```
ls ~/algo-tute/data/algo/
        >>
            lyso-test

cat ~/algo-tute/data/algo/lyso-test/algo_lyso-test_params.t
        >>
            ##Scorpy Algo Config File
          ##Created: 2024/07/29 14:31
            [algo]
            nq = 150
            qmax = 1.2
            qmin = 0.0725
            npsi = 180
            nl = 250
            lcrop = 45
            rotk = [1, 1, 1]
            rottheta = 0.5235987755982988
            dxsupp = 2
            pinv_rcond = 0.1
            eig_rcond = 1e-15
            lossy_iqlm = True
            lossy_sphv = True
```

Note that we have chosen a small `qmax`, `nq` and `npsi` for the purposes of this tutorial. Later, you can redo this tutorial by editing `make-algo-tag.py` to use `qmax=3.5`, `npsi=360*32` and `nq=300`

▼ 6. `make-targets.py`

The next script to run is `make-targets.py`. Given a `cif` file of structure factors, such as the one we previously downloaded from the PDB, it will create four new files in the tag directory. One of the new files, `lyso-test_targ-sf.cif` is an edited version of the input `cif`, with Bragg vectors between the `qmin` and `qmax` values specified for the algorithm tag. The file also expands all the symmetry mates of the Bragg vectors. The second file that is created `lyso-test_targ.hkl`, which has all the same Bragg vectors in the new `cif` file.

The other two files are `sphv_lyso-test_targ.npy` and `sphv_lyso-test_targ.log`. The `npy` file saves all the target reciprocal space intensity function in a `numpy` array, and the `log` file specifies the parameters to open the array as a `vol` object.

Sometimes the input `cif` file will have missing Bragg reflections within range of `qmin` and `qmax`. These vectors are also included in the target files, with intensity set to `nan`.

The `make-targets.py` scripts can be broken into two main components. The first component sets up the variables

```
import scorpy
import numpy as np
import matplotlib.pyplot as plt
data_dir = '/home/ec2-user/algo-tute/data'
tag = 'lyso-test'
cif_path = f'{data_dir}/lysozyme-sf.cif'
a = scorpy.AlgoHandler(tag=tag, path=f'{data_dir}/algo')
```

The next component opens the `AlgoHandler` object in read mode (`overwrite=0` is default), and makes the target files.

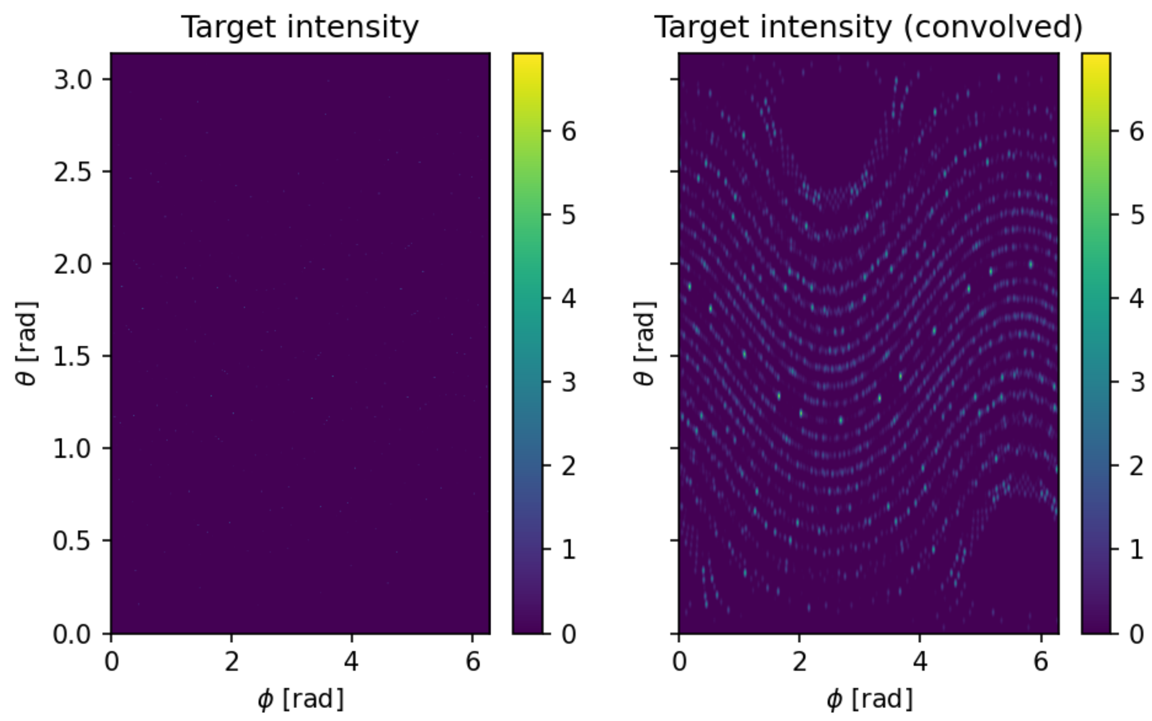
```
a = scorpy.AlgoHandler(tag=tag, path=f'{data_dir}/algo')
a.make_targets(cif_path)
```

Finally, the target intensity spherical vol object is loaded and plotted.

```
sphv = scorpy.SphericalVol(path=a.sphv_targ_path())
sphv.vol *= 1e-5

fig, ax = plt.subplots(1,2, figsize = (16/2.54, 8/2.54), dp
sphv.plot_slice(0, 150,fig=fig, axes=ax[0],
                title='Target intensity',ylabel='$\\theta$
sphv.vol[np.isnan(sphv.vol)] = 0
sphv.convolve(kern_L=2, kern_n=9, std_x=1, std_y=1, std_z=1
sphv.plot_slice(0, 150, fig=fig, axes=ax[1],
                title='Target intensity (convolved)',ylabel
plt.show()
```

In general, the `AlgoHandler` object can provide the paths for all the files it creates with the `*_path()` methods. The array can be accessed with the `sphv.vol` method. In the example, we scale the values to make them single digits. Next, we make the figure object to plot into. The `sphv` vol object has the `plot_slice` method, which will take a 2D slice of the 3D array. In this case, we take the 150th index from the 0th axis, the `q` axis in the `sphv` object. We can specify the `fig` and `axes` we want to plot on, and other plotting options. The next line sets all the `nan` values in the array to 0 so we can convolve the array with a Gaussian kernel that is 9 voxels wide, with a standard deviation of 1 in each axis direction across a range of +/- 2. This is done with the `convolve` method on the `sphv` object. Next, we plot the convolved intensity. The following image is the plot generated after running this script.



▼ 7. `make-support.py`

The next step is to make the support from the targets. This is done with the `make-support.py` script. Like in the last script, the first step is to create the `AlgoHandler` object in read mode.

```
import scorpy
import numpy as np
import matplotlib.pyplot as plt
data_dir = '/home/ec2-user/algo-tute/data'
tag = 'lyso-test'
a = scorpy.AlgoHandler(tag=tag, path=f'{data_dir}/algo')
```

Next, we run the `make_support_from_target` method.

```
a.make_support_from_target(verbose=99)
```

The `verbose` parameter defines how much should print when the method runs. If the method calls another verbose function, then it will run with a verbose level of one less. Hence, setting `verbose=99` means everything should print.

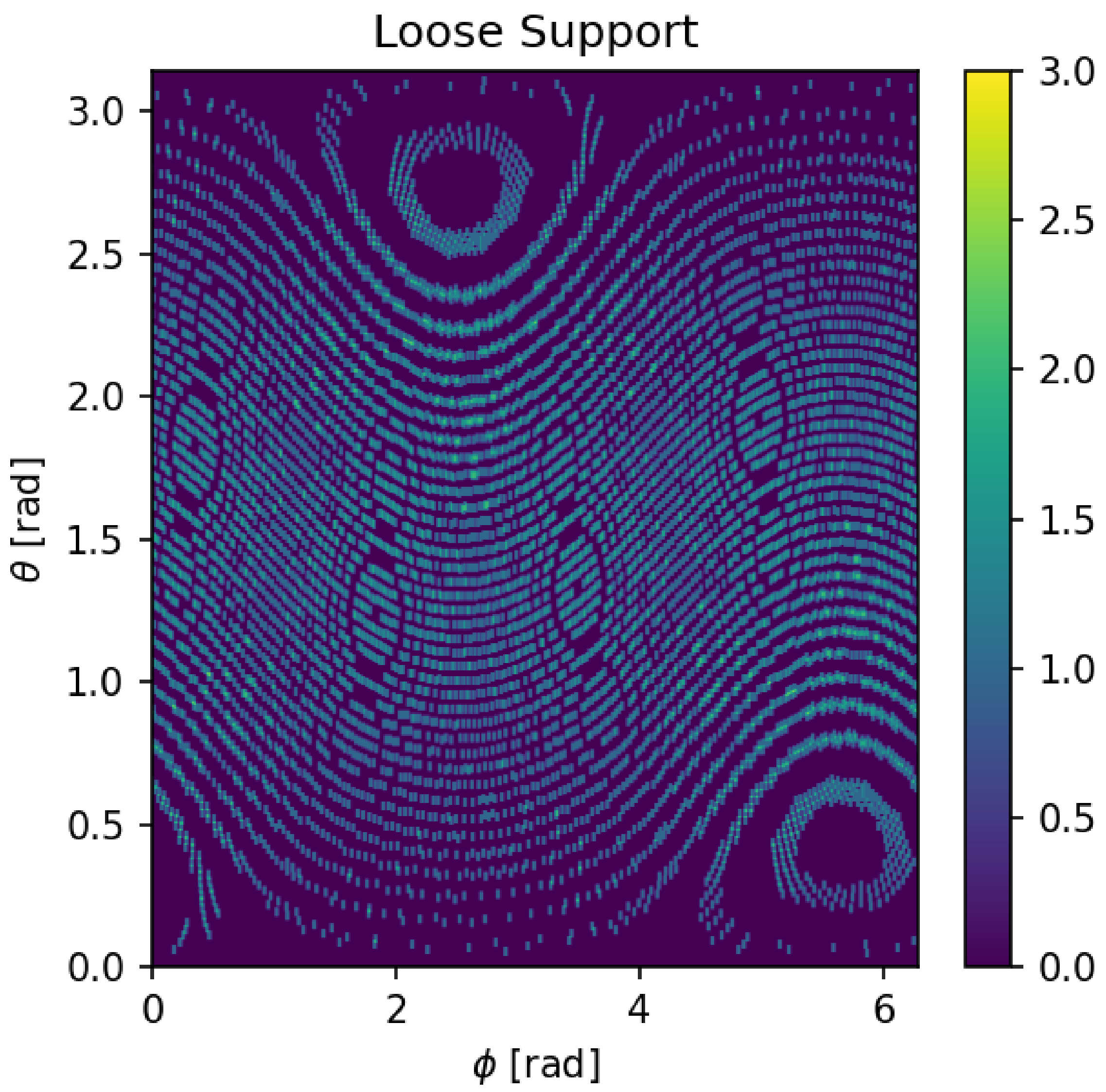
Running this function while verbose will print a warning if there is an overlap in the tight or loose support. Overlap occurs when the support region of a Bragg peak is wide enough that it overlaps with the support region of an adjacent Bragg pixel.

This script will create two `sphv` vols and one new `cif` and save them into the tag group directory. A tight support is made by setting all the intensities in `lyso-test_targ-sf.cif` to 1, including the `nan` values. This `cif` is saved as `lyso-test-_supp-sf.cif`. This `cif` is used to fill an `sphv` vol, which is saved as `sphv_lyso-test_supp_tight.npy` , with an associated `log` file. If there is an `OVERLAP IN TIGHT SUPPORT` message, then the sampling of the resolution of the algorithm may need to be increased. Overlap in the tight support means that two Bragg peaks are mapping to the same single voxel in the `sphv`.

Next, the loose support is made. The loose support is made by taking including every voxel that is `dxsupp` voxels away from the tight support voxel positions. For example, if a tight voxel position is `sphv.vol[i,j,k]`, then the loose voxel positions are `sphv.vol[i-dxsupp:i+dxsupp+1, j-dxsupp:j+dxsupp+1, k-dxsupp:k+dxsupp+1]`. If there is an `OVERLAP IN LOOSE SUPPORT` message, then the support regions of two Bragg peaks overlap.

Note that the volumes saved are the sums of support regions, so any overlap regions will have a value greater then 1. This can be shown by plotting the loose support.

```
sphv = scorpy.SphericalVol(path=a.sphv_supp_loose_path())
fig, axes = plt.subplots(1,1, figsize=(8/2.54, 8/2.54), dpi
sphv.plot_slice(0, 290, fig=fig, axes=axes,
                title='Loose Support',ylabel='$\\theta$ [ra
plt.show()
```



▼ 8. `make-corr.py`

The next step of the algorithm is to create a correlation function `corr` and harmonic order matrix `blqq`. This can be done by running the `make-corr.py` script. As in the previous script, we open an `AlgoHandler` object. To calculate the correlation function from the target intensities, use the following method.

```
a.make_data_from_target(verbose=99, save_corr=True, corr_nc
```

The `verbose` option works as before, it just prints relevant info to the screen. the `save_corr` parameters determines if the correlation `vol` is saved in the tag directory after it is calculated. For large correlation functions (`npsi=360*32`), the program may run out of RAM while saving. The `corr_nchunks` parameter determines how many chunks the correlating vectors are put into. For large sets of correlating vectors, the matrix multiplication can cause the computer to run out of RAM. Increasing the number of chunks will make smaller matrices and avoid running out of RAM. The `blqq` function is calculated from the correlation function and is always saved to the tag directory. The output from running the script should look something like this, and take approximately 1.5 minutes to run.

```
python make-corr.py
        >>
        Making Data
        Filling CorrelationVol from CifData.
        Started: Tue Jul 30 12:25:00 2024
        Correlating 6328 vectors.
        Chunk: 1/32
        Chunk: 2/32
        Chunk: 3/32
        Chunk: 4/32
        Chunk: 5/32
        Chunk: 6/32
        Chunk: 7/32
        Chunk: 8/32
        Chunk: 9/32
        Chunk: 10/32
        Chunk: 11/32
        Chunk: 12/32
        Chunk: 13/32
        Chunk: 14/32
        Chunk: 15/32
        Chunk: 16/32
        Chunk: 17/32
        Chunk: 18/32
        Chunk: 19/32
        Chunk: 20/32
        Chunk: 21/32
        Chunk: 22/32
        Chunk: 23/32
        Chunk: 24/32
        Chunk: 25/32
        Chunk: 26/32
        Chunk: 27/32
        Chunk: 28/32
        Chunk: 29/32
        Chunk: 30/32
        Chunk: 31/32
        Chunk: 32/32
        Finished: Tue Jul 30 12:26:22 2024

        ############
        Filling BlqqVol from CorrelationVol via Pseudo Matr
        Started: Tue Jul 30 12:26:22 2024
        Ended: Tue Jul 30 12:26:22 2024
        ############
```

Note that the saved `blqq` volume will save the values above `lcrop` parameter set by the algorithm. The `blqq` values about `lcrop` are excluded when the eigenvalues are calculated.

9. Make a recipe file

The final step to set up an algorithm run is to create a recipe file. The recipe file specifies what iterative schemes, such as HIO or ER should be used as the algorithm is run. To make a recipe file, create a file called `recipe.txt` in the `data` directory and fill it with the following content. Empty lines and lines beginning with `#` are ignored. Lines must be less then 64 characters.

```
##this is a comment
10 ER
5 HIO beta=0.9
```

```
#another comment
5 HIO beta=0.4
10 ER
```

Each line of the recipe file is run sequentially by the algorithm. Each line consists of a number of iterations, an iterative scheme, and keyword parameters for the schemes (if there are any). For example, the above recipe will run 10 iterations of ER, then 5 iterations of HIO with `beta=0.9`, then 5 iterations of HIO with `beta=0.4`, then 10 more iterations of ER.

A WORD OF WARNING.

The lines of the file are read and executed in the simplest (but fairly insecure way).

```
assert len(line) < 64, 'RECIPE LINE ERROR. Lines must be le
terms = line.split()
if terms == [] or line[0]=='#':
    continue
assert terms[0].isnumeric(),  'RECIPE LINE ERROR. First ter
assert terms[1] in dir(self),  'RECIPE LINE ERROR. Second t
niter = int(terms[0])
scheme = eval('self.'+terms[1])

kwargs = {}
for kwarg in terms[2:]:
    kwargs[kwarg.split('=')[0]] = eval(kwarg.split('=')[1])
```

Using `eval` is usually frowned upon, so be aware of what recipe files are run. The `assert` statements try to limit what kind of code could be injected into the `eval` function, such as the character limit, and the fact that the first word must be numeric, and the second term has to be a class method (iterative scheme).

▼ 10. `run-recon.py`

Now to finally run the algorithm. This can be done with the `run-recon.py` script. The first part of this script loads the `AlgoHandler` object, as in the previous examples. Next, the inputs are checked to see if they exist, that is, that the previous scripts have run or that appropriate replacements have been generated. Next, the inputs are loaded. The loaded inputs are support volumes and eigenvectors/values of the `blqq` matrix.

```
a.check_inputs(verbose=99)
a.load_inputs(verbose=99)
```

Finally the algorithm is run. Each algorithm run is given a `sub_tag` parameter, which identifies a single run within a `tag` directory. The path to the recipe file is also required as a parameter.

```
recipe_path = f'{data_dir}/recipe.txt'
a.run_recon(sub_tag='a', recipe=recipe_path, verbose=99)
```

Running the algorithm will produce the following output.

```
python run-recon.py
        >>
        Checking Inputs
        Blqq and Sphv Supports passed checks.
        Loading eigenvector/value and support inputs.
        Done Loading.
        Creating new sub_tag folder.
        Saved recipe and sphv_init to sub_tag folder.
        Running Recon lyso-test_a
        Started: Tue Jul 30 12:26:47 2024
        Running: 10 ER
        Running: 10 HIO beta=0.9
        Running: 10 HIO beta=0.4
        Running: 10 ER
        Finished: Tue Jul 30 12:31:33 2024
```

After the algorithm is finished running, there will be new saved outputs. Within `data/lyso-test` , a new directory with the `sub_tag=a` is created. Within `data/lyso-test/a/` , there is a saved copy of the recipe file, a copy of the final spherical volume, a copy of the random initial spherical volume, and a copy of the current iterating solution. Note that when the algorithm is finished, the final spherical volume and the current iteration solution are the same, but the current iterating solution is updated continuously while the algorithm runs. There is also a copy of the final intensities in a `cif` file.

```
ls data/algo/lyso-test/a
        >>
        hkls                          sphv_lyso-test_a_init.c
        lyso-test_a_final-sf.cif      sphv_lyso-test_a_init.l
        recipe_lyso-test_a.txt        sphv_lyso-test_a_iter.c
        sphv_lyso-test_a_final.log    sphv_lyso-test_a_iter.l
        sphv_lyso-test_a_final.npy    sphv_lyso-test_a_iter.r
```

There is also a `hkls` directory. This directory saves the integrated intensity of every Bragg peak in the support for every iteration.

```
ls data/algo/lyso-test/a/hkls
        >>
        lyso-test_a_count_0.hkl    lyso-test_a_count_28.hkl
        lyso-test_a_count_1.hkl    lyso-test_a_count_29.hkl
        ...
        lyso-test_a_count_21.hkl   lyso-test_a_count_40.hkl
```

The advantage of setting up `sub_tag` runs of an algorithm, is that it becomes trivial to re-run algorithms with the same conditions but with different random starts.

▼ 11. `cont-recon.py`

Suppose we have started a reconstruction with a recipe, and wish to continue that algorithm with a different recipe. This can be done by running another recon, but specifying the starting spherical volume. This is illustrated in `cont-recon.py` , by specifying the `sphv_init` parameter in the `run_recon` method.

```
recipe_path = f'{data_dir}/recipe.txt'
sphv_init = scorpy.SphericalVol(path=a.sphv_final_path('a')
a.run_recon(sub_tag='b', recipe=recipe_path, sphv_init=sphv
```

▼ 12. `troubleshoot-recon.py`

The methods for building the targets, supports and running the recon are simple ways to get an algorithm going. For greater control, each step of the algorithm can be run within a script. This is shown in `troubleshoot-recon.py` . In this script, a spherical volume is created, and the algorithm methods for `HIO` and the support constraint `Ps` are used. The spherical volumes after every 5 iterations are plotted (but not saved in any `sub_tag` folder).