Deconvolution Tutorial

Dr. Jack Dobinson

July 28, 2022

Abstract

Notes on the decovolution routines I've created. I've copied the relevant parts of my code-base into the 'scripts' folder. All the main routines are in the './scripts/fitscube/deconvolve/' sub-folder. Each of the './tutorial*/' folders contains an example implementation of one of the main routines I've used. Hopefully they should all run if you use the command `python3 -I ./run_XXX.py` from inside the folder.

1 Python Setup

1.1 Useful Information

Version
Python 3.9.6 ($\geq$ 3.8 required)

3rd Party Packages (used for the tutorials)

numpy
`https://numpy.org/install/`

scipy
`https://scipy.org/install/`

matplotlib
`https://matplotlib.org/stable/users/getting_started/`

astropy
`https://docs.astropy.org/en/stable/install.html`

Virtual Environments

pyenv
`https://github.com/pyenv/pyenv` Used to manage multiple versions of python without interfering with the system's installation. See this tutorial for details `https://towardsdatascience.com/managing-virtual-environment-with-pyenv-ae6f3fb835f8`

virtualenv
`https://github.com/pyenv/pyenv-virtualenv` A plugin for pyenv to manage multiple virtual environments

anaconda
`https://www.anaconda.com/products/distribution` A managed distribution and virtual environment manager. I've not used it really, but lots of people like it.

1.2 Background

Updating and upgrading Python and it's packages can be a horrible can of worms. The main problem is that many computers come with a *system* installation of Python. The *system* installation is required, as some utilities and system-level things need it. Unfortunately, if you update to a newer version of Python (especially if you swap your 'python' command to point to a 'python3' executable) nasty things can happen. Therefore, rather than solving the problem people stuck a patch on it called *version managers*, *virtual environments*, and *virtual environment managers*.

Technically, you can install a different version of Python onto your system just by providing some variables when installing it. However, it can become a massive hassle to rememeber which versions you need and where you kept them. 'pyenv' is a version manager, and can let you replace the commands 'python' and 'python3' with whatever you want without interfering with the *system* installation. I've managed to set up 'pyenv' on my machine such that I never have to do anything special unless I'm updating to a new version of python (and then I just need to change the 'global' version to the new one and ignore it again).

A virtual environment, just swaps out all of the environment variables that Python needs to different versions so they don't interfere with the *system* installation or any other installation. This is useful if you need a specific version of a package (to run some archaic scripts you found), but that version is incompatible with your normal setup. 'pyenv' has a plugin called 'pyenv-virtualenv' that can make and manage virtual environments for you.

Anaconda is a managed python distribution that comes with a version manager and virtual environment manager built-in. I've not used it much, but many people do and it's considered good. It's probably the simplest way to get everything working, they also have a 'miniconda' version that just installs their management software so you can customise everything yourself.

2 Deconvolution

2.1 Notation

$O(\underline{x})$ The *ground truth* signal (true-signal, or original-signal), i.e. what we would measure if our instruments were perfect.

$R(\underline{x})$ The response function of an instrument. Usually called the Point Spread Function (PSF) when talking about telescopes.

$N(\underline{x})$ Some noise function that corrupts $O(\underline{x})$.

$I(\underline{x})$ The observational data recorded by the instrument. I.e., $O(\underline{x})$ after it has been passed through the response function and corrupted by noise.

$\underline{x}$ A vector that holds the state of a system, in our case $\underline{x}$ is the position of a pixel in an image.

$\mathcal{F}[...](u)$ The fourier transform operation to some conjugate space denoted by u.

$\mathcal{F}^{-1}\left[...\right](x)$ The inverse fourier transform back to the original space denoted by x.

$\tilde{\square}$ An estimate of the decorated symbol.

$\square^{\star}$ Conjugate transpose of the decorated symbol.

2.2 PSF Normalisation

In the rest of this document, I've made the implicit assumption that the Response Function, $R(\underline{x})$ (or PSF for telescopes), is normalised. By that, I mean the following conditions are true:

- $\sum_{\underline{x}} R(\underline{x}) = 1$, I.e. convolving (or deconvolving) the observation with the response function does not change the total sum of the signal, except for edge effects in some cases.
- If $R(\underline{x})$ is represented as a vector or array, it's shape is *odd* in all of it's dimensions. I.e. A PSF needs to have a unambiguous central pixel.
- $R(\underline{x})$ is centered such that convolution (or deconvolution) with $R(\underline{x})$ does not shift a signal. I.e. A PSF's centroid (and hopefully brightest pixel) is coincident with it's central pixel. Hence the need for an unambiguous center.
- $R(\underline{x})$ Has no NANs in it, generally these play havok with convolution and deconvolution routines.

A failure of these conditions can typically be fixed after the fact (except for the NAN condition) by working out what the effect was and undoing it, but it's a hassle.

2.3 Background

For any original signal, $O(\underline{x})$, measured by an instrument there is some corruption due to non-perfect response $R(\underline{x})$, and noise $N(\underline{x})$, which gives an observation $I(\underline{x})$ that is related to the original signal by

$$\begin{aligned} I(\underline{x}) &= \int R(\underline{x} - \underline{x}_1)\, O(\underline{x}_1)\, d\underline{x}_1 + N(\underline{x}) \\ &= (R \star O)(\underline{x}) + N(\underline{x}) \end{aligned} \tag{1}$$

where $\star$ denotes the convolution operation.

The problem we face much of the time is recovering the original signal $O(\underline{x})$ from the observed data $I(\underline{x})$ as accurately and quickly as possible.

The convolution theorem

$$\mathcal{F}\left[(f \star g)(x)\right](k) = \mathcal{F}\left[f(x)\right](k)\mathcal{F}\left[g(x)\right](k), \tag{2}$$

where $\mathcal{F}\left[f(x)\right](k)$ denotes the fourier transform of $f(x)$, k is the conjugate variable to x, and $\mathcal{F}^{-1}\left[...\right](x)$ denotes the inverse fourier transform operation, helps us see some way to fix the problem.

As the Fourier transform is a linear operator,

$$\mathcal{F}\left[I(\underline{x})\right](\underline{k}) = \mathcal{F}\left[R(\underline{x})\right](\underline{k})\,\mathcal{F}\left[O(\underline{x})\right](\underline{k}) + \mathcal{F}\left[N(\underline{x})\right](\underline{k}),$$

which can be re-arranged to give

$$\mathcal{F}\left[O(\underline{x})\right](\underline{k}) = \frac{\mathcal{F}\left[I(\underline{x})\right](\underline{k}) - \mathcal{F}\left[N(\underline{x})\right](\underline{k})}{\mathcal{F}\left[R(\underline{x})\right](\underline{k})}, \tag{3}$$

the *fourier quotient* method of deconvolution. Unfortunately in practice there is a frequency cutoff to our measurements of $R(\underline{x})$ and close to that cutoff the noise $N(\underline{x})$ is amplified, leaving (3) unusable where there is any high-frequency noise, which there usually is.

Also, (1) is typically an ill-posed problem as there are many possible $O(\underline{x})$ functions that can give a $I(\underline{x})$ observation. Therefore, deconvolution in general is used to find a method which gives an *estimate*, $\tilde{O}(\underline{x})$, of the original signal.

2.3.1 Lucy-Richardson

Many approaches to deconvolution take inspiration from fitting algorithms and probability theory; one widely used method is Lucy-Richardson (LR) [1, 2] deconvolution [3, section 5.4]. LR-deconvolution is based on the principle of finding a maximum likelihood estimator (MLE) for $O(\underline{x})$

The likelyhood comes from Bayes theorem

$$p(O|I) = \frac{p(I|O)p(O)}{p(I)}, \tag{4}$$

where $p(O|I)$, the likelyhood, is the probability that our estimate $\tilde{O}(\underline{x})$ is the original signal $O(\underline{x})$ for some observed data $I(\underline{x})$, $p(I|O)$ is the probability that we could get our observed data $I(\underline{x})$ from our estimate $\tilde{O}(\underline{x})$, $p(O)$ is the probability our estimate is correct, and $p(I)$ is the probability our data is correct.

When minimising (4), our observational data $I(\underline{x})$ is constant, and $O(\underline{x})$ is the subject. Therefore $p(I)$ will not change, but $p(I|O)$ will vary depending on the different original signals we test. $p(I|O)$ is the probability of getting our observed data $I(\underline{x})$ from some original signal $O(\underline{x})$, i.e. it is the joint probability that, given some statistical model $\mathcal{N}$ we would draw $I(\underline{x})$ from the possible set of functions given by $(R \star O)(\underline{x})$, i.e.

$$\begin{aligned} p(I|O) &= \prod_{\underline{x}} (\mathcal{N}[(R \star O)(\underline{x})] = I(\underline{x})) \\ &= \prod_{\underline{x}} f_{\mathcal{N}}(I(\underline{x}); (R \star O)(\underline{x})) \end{aligned}$$

Or, as this is always positive we can use the log-likelihood instead

$$\begin{aligned} l(O|I) &= \ln[\mathcal{L}(O|I)] = ln[p(I|O)] \\ &= \sum_{\underline{x}} \ln[f_{\mathcal{N}}(I(\underline{x}); (R \star O)(\underline{x})] \end{aligned} \tag{5}$$

$$\tag{6}$$

where $f_{\mathcal{N}}(a(\underline{x});\theta(\underline{x}))$ is the probability mass function (or probability density function for continuous distributions) for a random variable to be $a(\underline{x})$ when it is drawn from the distribution $\mathcal{N}$ with the parameters $\theta(\underline{x})$, $\mathcal{L}(O|I)$ is the *likelihood* of $O(\underline{x})$ given we have the data $I(\underline{x})$, and $l(O|I)$ is the *log-likelihood*.

Since in Astronomy, images are made up of photons collected by a telescope and deposited on a CCD Poisson statisticis apply. Therefore

$$\begin{aligned}
\mathcal{N} &\sim \text{Poisson}((R \star O)(\underline{x})) \\
f_{\mathcal{N}}(k;\lambda) &= \frac{\lambda^k e^{-\lambda}}{k!} \\
\mathcal{L}(O|I) &= \prod_{\underline{x}} \frac{(R \star O)(\underline{x})^{I(\underline{x})} e^{-(R\star O)(\underline{x})}}{I(\underline{x})!} \\
l(O|I) &= \sum_{\underline{x}} I(\underline{x}) \ln[(R \star O)(\underline{x})] - (R \star O)(\underline{x}) - I(\underline{x})!,
\end{aligned}$$

after some manipulation [4], we get

$$O_{n+1}(\underline{x}) = \left[\frac{I}{(R \star O_n)} \star R^{\star}\right](\underline{x})\, O_n(\underline{x}) \tag{7}$$

where successive iterations give new approximations to the original signal.

In general LR-deconvolution works well for point-like sources, however where to stop the iteration is non-trivial. As n increases, higher and higher fourier frequencies are being deconvolved and you start running into the same problem as (3), noise becomes amplified.

2.3.2 Modified CLEAN

CLEAN [5] is used extensively in radio astronomy, and occasionally in the wider astronomy community [6]. The original algorithm builds up an estimate $\tilde{O}(\underline{x})$ (the *component map*) by subtracting the response function convolved with low-amplitude delta-functions from $I(\underline{x})$, and adding them to the estimate. I.e.

$$\begin{aligned}
\tilde{O}_0(\underline{x}) &= \underline{0} \\
I_n(\underline{x}) &= I(\underline{x}) - (R \star \tilde{O}_n)(\underline{x}) \\
x_i &= \text{argmax}[I_n(\underline{x})] \\
\tilde{O}_{n+1}(\underline{x}) &= \tilde{O}_n(\underline{x}) + g_{\text{loop}} I_n(x_i)
\end{aligned} \tag{8}$$

where g_{loop} is the *loop-gain*, the fraction of the brightest pixel added to the *components*, $\tilde{O}(\underline{x})$, each iteration. Stopping criteria are fairly standard, usually once $I_n(x_i)$ has reduced by some factor. Once the component map $\tilde{O}(\underline{x})$ is complete, it is convolved with the *clean beam* $\tilde{R}(\underline{x})$, a response function that does not have the wings of the original $R(\underline{x})$ (the *dirty beam*) this gives the *clean map* $\tilde{I}(\underline{x}) = (\tilde{R} \star \tilde{O})(\underline{x})$.

This method has various problems for our purposes, mostly that extended sources are not deconvolved efficiently and exhibit striping artifacts on the size scale of $R(\underline{x})$. To get around this limitation various methods have been proposed [7, 8, 9, 10, 11], for our purposes Modified CLEAN [12] seems to have the best properties.

Modified CLEAN works similarly to (8) except that instead of just selecting the brightest pixel, all pixels above the selection threshold, t_s, are treated as a single extended source. For sufficiently small values of $g_{\rm loop}$ and t_s the subtractions will be linear enough that the differences between regions of slightly different brightness do not matter. Modified clean has the following main iterative scheme:

$$\begin{aligned}
\tilde{O}_0(\underline{x}) &= \underline{0}, \\
I_n(\underline{x}) &= I(\underline{x}) - (R \star \tilde{O}_n)(\underline{x}), \\
\underline{x}_s &= \{x_i; I_n(x_i) > t_s; x_i \in \underline{x}\}, \\
\tilde{O}_{n+1}(\underline{x}) &= \tilde{O}_n(\underline{x}) + g_{\rm loop} I_n(x_s).
\end{aligned} \tag{9}$$

Variable Threshold

Another modification to (9) is to vary t_s, the selection threshold, to attempt to deconvolve small and bright features before more extended features.

One method is Otsu Thresholding [13, 14], a technique that divides a set into two different classes by choosing a dividing threshold, $t_{\rm otsu}$, in such a way that the variance between the two classes is minimised. Apply successive Otsu thresholds to an image, and then selecting the candidate threshold that is the most *exclusive* (i.e. the threshold that rejects the largest fraction of pixels per fraction of range.

$$e(t) = \frac{f_{\rm range}(t) - f_{\rm pix}(t)}{f_{\rm range}(t) + f_{\rm pix}(t)},$$

where $e(t)$ is the exclusivity of a threshold t, $f_{\rm range}(t) = (t - \min)/(\min - \max)$ is the fraction of range rejected by a threshold, and $f_{\rm pix}(t) = N_{\rm bright}/N$ is the fraction of pixels kept by the threshold. Here $N_{\rm bright}$ is the number of pixels above the threshold, and N is the original number of pixels. Finally, t_s is calculated via:

$$\begin{aligned}
t_{{\rm otsu},j} &= \text{The j}^{\rm th}\text{ Otsu threshold of our image data}\,, I(\underline{x}), \\
t_{\rm factor} &= \text{A user supplied factor}\,, > 0, < 1, \\
i &= \text{argmax}\{e(t_{{\rm otsu},j}); 0 <= j <= N_{\rm thresholds}\}, \\
t_i &= t_{{\rm otsu},i}, \\
t_s &= \max[I_n(\underline{x})] t_{\rm factor} + (1 - t_{\rm factor}) t_i.
\end{aligned}$$

As before, the final estimate of the original signal $\tilde{O}(\underline{x})$ is normally convolved with a *clean beam* $\tilde{R}(\underline{x})$ found from the main lobe of the response function (usually an approximate gaussian) to give the *clean map*,

$$\tilde{I}(\underline{x}) = (\tilde{R} \star \tilde{O})(\underline{x}).$$

When the components $\tilde{O}(\underline{x})$ of the image are not well correlated with their neighbours this process can result in huge gains. However, in various tests it has been found that this step is not always neccesary for the production of a smooth image. Infact using the components directly has been better in many cases, with the extra convolution step just degrading the spatial resolution of the map.

Flux Loss
Typically deconvolution does not change the total sum of an observation, or any differences are marginal. However if not enough iterations are performed to completely explain all of the signal in our observation, then some difference in $\sum_{\underline{x}} I(\underline{x})$ vs $\sum_{\underline{x}} \tilde{O}(\underline{x})$ may occur.

The simplest way to deal with this difference is to add back the residual such that the *clean map* is

$$\tilde{I}(\underline{x}) = (\tilde{R} \star \tilde{O})(\underline{x}) + \left[I(\underline{x}) - (R \star \tilde{O})(\underline{x}) \right],$$

i.e., the residual of the deconvolution is added to the output.

2.4 Usage of `./tutorial_0_deconv/run_deconv.py`

```
usage: run_deconv.py [-h] [--output.file str] [--output.file.overwrite]
                     [--CleanModified.show_plots] [--CleanModified.save_plots]
                     [--CleanModified.plot_dir str] [--CleanModified.verbose int]
                     [--CleanModified.n_iter int] [--CleanModified.loop_gain float]
                     [--CleanModified.threshold float]
                     [--CleanModified.n_positive_iter int]
                     [--CleanModified.noise_std float]
                     [--CleanModified.rms_frac_threshold float]
                     [--CleanModified.fabs_frac_threshold float]
                     OBS_FITS_PATH PSF_FITS_PATH

# DESCRIPTION #
        Front-end to deconvolution routines.

        ## Example ##
                $ python3 ./deconv.py ../data/test_rebin.fits{1}[229:230]
                      ../data/test_standard_star.fits{1}

        ## FITS File Path Format ##
                ../path/to/datafile.fits{ext}[slice_tuple](img_axes_tuple)
                        ext : int | str
                                Extension of the FITS file to operate upon, if not present,
                                will use the first extension that has some data.
                        slice_tuple : tuple[Slice,...]
                                Slice of the data to operate upon, useful for choosing
                                a subset of wavelength channels. If not present, will
                                assume all wavelengths are to be deconvolved.
                        img_axes_tuple : tuple[int,...]
                                Tuple of the spatial axes indices, uses FITS ordering
                                if +ve, numpy ordering if -ve. Usually the RA,DEC axes.

                ### Examples ###
                        ./some_datafile.fits{PRIMARY}[100:150]
                                Select the extension called 'PRIMARY' and deconvolve the
                                channels 100->150
                        /an/absolute/path/to/this_data.fits[99:700:50]
                                Try to guess the extension to use, deconvolve every 50th
                                channel in the range 99->700
                        ./deconv/whole/file.fits{SCI}
```

```
                                Use the extension 'SCI'
                        ./some/path/big_file.fits{1}[5:500:10](0,2)
                                use the 1st extension (not the 0th), deconvolve every 10th
                                channel in the range 5->500, the spatial axes are 0th and
                                2nd axis.


# END DESCRIPTION #

positional arguments:
  OBS_FITS_PATH         Observation file to operate upon, uses FITS file path format
  PSF_FITS_PATH         Point Spread Function (PSF) to use during deconvolution, uses FITS
                        file path format

optional arguments:
  -h, --help            show this help message and exit
  --output.file str     file to output deconvolved data to. (default: ./output/deconv.fits)
  --output.file.overwrite, --output.file.no_overwrite
                        Should we overwrite the output file or not? (default: True)

CleanModified:

        A modified verison of the CLEAN algorithm, designed to account for non-point
        objects better than standard CLEAN see
        https://articles.adsabs.harvard.edu/pdf/1984A%26A...137..159S

        Creation of the class sets up the attributes that define how the algorithm behaves,
        calling the class runs the algorithm on input data.


  --CleanModified.show_plots, --CleanModified.no_show_plots
                        should plots be displayed interactively when running?
                        (default: False)
  --CleanModified.save_plots, --CleanModified.no_save_plots
                        should plots be saved to a file?
                        (default: False)
  --CleanModified.plot_dir str
                        folder to save plots to (if they should be saved), relative to
                        current working directory
                        (default: ./plots)
  --CleanModified.verbose int
                        verbosity of messages from algorithm
                        (default: 1)
  --CleanModified.n_iter int
                        Number of iterations
                        (default: 1000)
  --CleanModified.loop_gain float
                        Fraction of emission that could be accounted for by a PSF added to
                        components each iteration. Higher values are faster, but unstable.
                        I may want a way to scale this with the number of iterations, or
                        with the amount of point-like sources found each step. Once the
                        image becomes noisy (i.e. is already noisy, or only noise left in
                        map), this can lead to stippling accross the image when it's too
                        high
```

```
                        (default: 0.5)
  --CleanModified.threshold float
                        Fraction of maximum brightness above which pixels will be included
                        in CLEAN step (want 0 < x < 1)
                        Some magic numbers are:
                               0
                                       Uses "otsu thresholding" to guess a good value
                               0 > x > -1
                                       Uses multiple "otsu thresholds" each iteration to
                                       get a value, and then applies the threshold value
                                       so that higher order otsu threholds result in a
                                       much larger effective "self.threshold" value.
                                       Useful for deconvolving bright features on a
                                       surface first.
                               -1
                                       Uses multiple "otsu thresholds" each iteration,
                                       chooses a higher order one based onhow many pixels
                                       the higher order threshold rejects. Selects for
                                       many pixels rejected as this should mean that
                                       small-but-bright features are deconvolved first.
                               -2
                                       Uses "otsu thresholding" each iteration, just uses
                                       the first value.
                        (default: -1)
  --CleanModified.n_positive_iter int
                        Number of iterations to do that only "adds" emission, before
                        switching to "adding and subtracting" emission
                        (default: 0)
  --CleanModified.noise_std float
                        Estimate of the deviation of the noise present in the observation
                        (default: 0.01)
  --CleanModified.rms_frac_threshold float
                        Fraction of original RMS of residual at which iteration is stopped
                        (default: 0.01)
  --CleanModified.fabs_frac_threshold float
                        Fraction of original Absolute Brightest Pixel of residual at which
                        iteration is stopped
                        (default: 0.01)

END OF USAGE
```

3 SSA Filtering

3.1 Background

Singular Spectrum Analysis (SSA) [15] was originally developed as a model-free technique to decompose a time series (including a non-stationary time series) into the sum of interpretable componets such as trend, periodic components, and noise, with no a-priori assumptions about the form of those components. For a full derivation of the 2D treatment, see [16] however the 1D form is fairly straight forward and has the same basic steps.

3.2 Why go to the trouble?

The problem I was trying to solve when I started using this technique was this: How can I reduce/flag instrumental artifacts that vary in position and intesity between wavelength channels without laboriously going through each image? I tried a few different techniques.

Low pass filtering had quite a few problems. It did reduce smaller artifacts, but did not affect larger ones that are extended in one dimension but not another. Furthermore, it mostly just blurred the observational data which is the exact opposite of what we are trying to achive with the whole process, namely removing blur to get good estimates of limb-darkening parameters.

Edge detection via different methods (e.g. Sobel filtering) was moderately successful, however all the techniques I found preferentially find edges along certain directions. That wasn't too much of a problem, as the routines are cheap (computationally) and can be run multiple times, but they also missed obvious artifacts and flagged non-artifact. I.e., their false positive and false negative rate are quite high.

I had a look into computer vision libraries to see if I could use motion estimation techniques like Optical Flow or Block Matching, as many artifacts move across the imaged field w.r.t. wavelength. However, this had a couple of problems. One, not all of the artifacts move and I would still have to find some way to deal with them. Two, I was starting to veer into having to learn a lot about computer vision which was probably just as much work as going through the images manually (at least for one observation).

SSA was suggested to me by a friend, and while it is certainly not perfect, it has a good balance of flagging obvious artifacts, not flagging actual data, and being fairly computationally efficient. In the implementation I'm using it's not using any wavelength-varying information, so there's probably a way to improve it (3D SSA?) or use a secondary technique to identify artifacts that move with wavelength.

3.2.1 Singular Value Decomposition

SSA uses Singular Value Decomposition (SVD) to split up a series into components, here's a quick overview of it. SVD factorises an $m \times n$ matrix into an orthogonal eigenbasis.

$$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\star}$$

Matrix	Shape
$\boldsymbol{M}$	$(m \times n)$
$\boldsymbol{U}$	$(m \times m)$
$\boldsymbol{\Sigma}$	$(m \times n)$
$\boldsymbol{V}$	$(n \times n)$

$\boldsymbol{M}$ is the matrix to factorise, $\boldsymbol{U}$ is the left singular matrix, $\boldsymbol{\Sigma}$ holds the singular values along its diagonal, and $\boldsymbol{V}$ is the right singular matrix. $\boldsymbol{U}$ and $\boldsymbol{V}$ are *unitary*, their conjugate transpose is their inverse, and $\boldsymbol{\Sigma}$ is a diagonal matrix with non-negative real numbers on its diagonal.

SVD gives a non-unique solution, however we can always choose Σ_{ii}'s to be in decending order. In which case $\boldsymbol{\Sigma}$ is unique, but $\boldsymbol{U}$ and $\boldsymbol{V}$ are not.

Sometimes the compact form of SVD is used, which only includes the non-zero terms in $\boldsymbol{\Sigma}$, in this case the shapes of the matrices are changes such that $\boldsymbol{U}$ is shape $(m \times r)$, $\boldsymbol{\Sigma}$ is shape $(r \times r)$, and $\boldsymbol{V}$ is shape $(r \times n)$. In this case $\boldsymbol{U}$ and $\boldsymbol{V}$ are *semi unitary*, and r is the number of non-zero elements in the original $\boldsymbol{\sigma}$.

3.2.2 Method

Let $\mathbb{X}$ be a real-values time series, $\mathbb{X} = (x_1, x_2, ..., x_n)$ of length N, let L (with $1 < L < N$) be an integer called the *window length*, and $K = N - L + 1$.

First form the *trajectory matrix* $\boldsymbol{X}$, the $L \times K$ matrix that holds each moving window of length L in its columns.

$$\boldsymbol{X} = \begin{pmatrix} x_1 & x_2 & ... & x_K \\ x_2 & x_3 & ... & x_{K+1} \\ \vdots & \vdots & \ddots & \vdots \\ x_L & x_{L+1} & ... & x_N \end{pmatrix}$$

$\boldsymbol{X}$ is a *Hankel* matrix, i.e., all its anti-diagonals have constant elements.

Secondly, perform SVD on $\boldsymbol{X}$. A slightly faster way than the direct version is the following. Compute

$$\boldsymbol{S} = \boldsymbol{X}\boldsymbol{X}^{\mathrm{T}},$$

S is of shape $L \times L$, and let $\lambda_1, \lambda_2, ..., \lambda_L$ be the *eigenvalues* of $\boldsymbol{S}$. Therefore,

$$\begin{aligned} \boldsymbol{S} &= (\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\star)(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\star)^\star \\ &= \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\star\boldsymbol{V}\boldsymbol{\Sigma}^\star\boldsymbol{U}^\star \\ &= \boldsymbol{U}\boldsymbol{\Sigma}^2\boldsymbol{U} \end{aligned}$$

the eigenvalues of $\boldsymbol{S}$ are the squares of the singular values of $\boldsymbol{X}$, and the eigenvectors of $\boldsymbol{S}$ are the columns of the left singular matrix $\boldsymbol{U}$ of $\boldsymbol{X}$. Also,

$$\begin{aligned} & \boldsymbol{X}^\star = \boldsymbol{V}\boldsymbol{\Sigma}^\star\boldsymbol{U}^\star \\ \rightarrow \quad & \boldsymbol{X}^\star\boldsymbol{U}\boldsymbol{\Sigma}^{-1} = \boldsymbol{V} \end{aligned}$$

means that the right singular matrices $\boldsymbol{V}$ of $\boldsymbol{X}$ can be found via

$$V_i = \frac{\boldsymbol{X}^\star\boldsymbol{U}}{\sqrt{\lambda_i}}.$$

So, the SVD can be decomposed into a sum via

$$\begin{aligned} \boldsymbol{U} &= (U_1, U_2, ..., U_L) \\ \boldsymbol{V} &= (V_1, V_2, ..., V_K) \\ \text{let} \quad d &= \min(L, K) \\ \boldsymbol{X} &= (U_1, U_2, ..., U_L)\boldsymbol{\Sigma}(V_1, V_2, ..., V_K)^\star \\ &= (U_1\Sigma_{11}, U_2\Sigma_{22}, ..., U_d\Sigma_{dd})(V_1, V_2, ..., V_d)^\star \\ &= \sum_{i=1}^{d} \Sigma_{ii} U_i V_i^\star \end{aligned}$$

such that

$$\begin{aligned}\boldsymbol{X} &= \boldsymbol{X}_1 + \boldsymbol{X}_2 + ... + \boldsymbol{X}_d \\ \boldsymbol{X}_i &= \sqrt{\lambda_i} U_i V_i^\star.\end{aligned}$$

The $(\sqrt{\lambda_i}, U_i, V_i)$ quantities are the *eigentriples* of the SVD of $\boldsymbol{x}$. Sometimes $\sqrt{\lambda_i} V_i$ are called the *principle components* of $\boldsymbol{X}$. $\boldsymbol{X}_i$'s are called the *elementary matrices.*

Thirdly, group the eigentriples. This step is kinda optional, you can just use the elementary matrices directly, called *elementary grouping*, but inspecting the eigentriples and grouping together those whose singular vectors vary at a similar frequency can reveal trends and periodic components to the data.

In maths terms, partition the indices $\{1, 2, ..., d\}$ into l disjoint subsets $I_1, I_2, ..., I_l$, with $I = \{1, 2, ..., p\}$. Therefore $\boldsymbol{X}_I$ corresponding to group I is $\boldsymbol{X}_I = \boldsymbol{X}_{i_1} + \boldsymbol{X}_{i_2} + ... + \boldsymbol{X}_{i_p}$, and

$$\boldsymbol{X} = \boldsymbol{X}_{I_1} + \boldsymbol{X}_{I_2} + ... + \boldsymbol{X}_{I_l}$$

Fourth, diagonally average the grouped matrices and extract the associated time-series. Each $\boldsymbol{X}_{I_j}$ matrix is *Hankelised* (i.e., their anti-diagonals are averaged), and the time-series associated with hankelised matrix $\tilde{\boldsymbol{X}}_{I_j}$ is extracted by takling the main diagonal to get a reconstructed time series

$$\tilde{\mathbb{X}}^{(j)} = (\tilde{x}_1^{(1)}, \tilde{x}_2^{(2)}, ..., \tilde{x}_N^{(N)}).$$

Therefore, the initial series $\mathbb{X} = (x_1, x_2, ..., x_n)$ is decomposed into a sum of l reconstructed subseries

$$x_n = \sum_j^l \tilde{x}_n^{(j)}, \quad \text{where} \quad n \in (1, 2, ..., N)$$

such that

$$\mathbb{X} = \tilde{\mathbb{X}}^{(1)} + \tilde{\mathbb{X}}^{(2)} + ... + \tilde{\mathbb{X}}^{(l)}.$$

Hankelisation is the process of taking some matrix $\boldsymbol{A}$ and replacing each anti-diagonal with the mean of the elements on that diagonal. E.g.

$$\boldsymbol{A} = \begin{pmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{pmatrix} \quad \rightarrow \quad \tilde{\boldsymbol{A}} = \begin{pmatrix} \tilde{a_1} & \tilde{a_2} & \tilde{a_3} \\ \tilde{a_2} & \tilde{a_3} & \tilde{a_4} \\ \tilde{a_3} & \tilde{a_4} & \tilde{a_5} \end{pmatrix}.$$

In general for an $m \times n$ matrix,

$$\begin{aligned} \tilde{a_k} =& \frac{\sum_{(i,j)\in D_k} a_{ij}}{\text{Count}(D_k)} \\ D_k =& \{(i,j) : i + j = k + 1, 1 \leq i \leq m, 1 \leq j \leq n\} \\ \text{Count}(D_k) \coloneqq& \text{The number of elements in } D_k \end{aligned}$$

To summarise, window length L determins the resolution of the SSA, larger L captures more data and therefore leads to better separability of different components. Also, L defines the largest periodic features that can be captured. Trends are found by grouping eigentriples with slowly varying eigenvalues, periodicity is found by grouping similarly sized eigenvalues with periodically varying eigenvectors, noise is generally small (but not always) eigenvalues with stochastically varying eigenvectors.

Applying SSA to an image instead of a time series uses 2D-SSA [16]. In image space, instead of trends over time, you are looking at variation over space. The $\tilde{\mathbb{X}}^{(j)}$ decomposed images therefore show decomposed spatial trends, periodicity, and noise.

In the SSA filtering routine, I've implicity assumed that noise is separated into components with smaller eigenvalues. For each decomposed image, $\tilde{\mathbb{X}}^{(j)}$, the cumulative distribution function $p(\tilde{x}_i^{(j)} \leq x; \tilde{x}_i^{(j)} \in \tilde{\mathbb{X}}^{(j)})$ is calculated, and then transformed to give each pixel a score $s_i^{(j)}$ via

$$s_i^{(j)} = 4(p(\tilde{x}_i^{(j)} \leq x; \tilde{x}_i^{(j)} \in \tilde{\mathbb{X}}^{(j)}) - 0.5)^2,$$

which is intended to capture the squared distance away a pixel is from the mean value of its component image.

As low-eigenvalue components make up the majority of the noise, and the original image is the sum of the decomposed SSA component images. It stands to reason that any pixels that are consistently far from the median of the low-eigenvalue components will be pixels with a large component of noise. Therefore the pixels which, when averaged over all low-eigenvalue components, have a large score should be those that are most influenced by noise. So,

$$s_i = \sum j = l_1 l_2 s_i^{(j)},$$

where l_1, l_2 are the component images included in this analysis ($l_1 = 3$ and $l_2 = 12$ works well for a 4×4 window) and $0 \leq s_i \leq 1$. We set some fraction

s_f that is the cutoff of pixels we consider "noisy" (I've found $s_f = 0.95$ to be a good value).

$$I_{\text{noisy}} = \{i \text{ where } s_i > s_f\}$$

The indices of I_{noisy} can then be interpolated over to remove those noise artifacts and enable faster and more accurate deconvolution.

4 Interpolation

4.1 Usage

Interpolation is the process of "filling in" values that are in-between data that we already have. There are various well documented methods [17] to perform this task.

For the implementation in the deconvolution tutorials, I've used `scipy`'s `interpolate.griddata` function for the actual interpolation. I've built an interface in `utilities.sp.interpolate_at_mask` that interpolates a `numpy` array at the `true` positions of a boolean *bad pixel* array.

The prototype

```
def interpolate_at_mask(data, mask, edges=None, **kwargs):
```

has three arguments:

`data` a `numpy` array holding the data to interpolate

`mask` a `numpy` boolean array that is `true` in the places that should be interpolated

`edges` an optional string describing how the edges should be treated, mostly used when an image has a "frame" of `NAN`s.

`None` Don't worry about edges, ignore them

`"convolution"`

- Convolve the whole image with a square kernel that is 1/2 its shape (i.e., a 40×30 array is convolved with a square 20×15 to give a 78×58 array when you include partial overlaps). That way we should create a map that, in the regions outside the original `data`, has some relation to `data` and is somewhat continuous with the edges of `data` that are not NANs.
- Embed the `data` we are not interpolating over into the convolved map so we now have no edges that are NANs and can't be interpolated over.
- Interpolate the embedded `data` array.
- Remove the 'frame' of convolved data and just return the embedded `data`.

`kwargs` other keyword arguments are forwarded to `scipy.interpolate.griddata`.

5 PSF Modelling

5.1 Notation

Pupil Function (PF) How the electric field is transmitted through a telescope to the exit pupil. Usually modelled as a real function of the amplitude of the E-field, but in principle can be complex and contain phase information.

Amplitude Transfer Function (ATF) A scaled version of the PF such that frequencies with perfect transmission have a value of 1. Typically is the same as the PF.

Amplitude Spread Function (ASF) $\mathcal{F}[\text{ATF}]$, the optical system's response to *coherent* light, i.e., phase differences do not matter.

Point Spread Function (PSF) $|\text{ASF}|^2 = \mathcal{F}^*[\text{ATF}]\,\mathcal{F}[\text{ATF}] = \mathcal{F}[\text{ATF} * \text{ATF}]$. The squared magnitude of the ASF, and the inverse fourier transform of the Optical Transfer Function (OTF). The spatial response of the system to *incoherent* light.

Optical Transfer Function (OTF) $\mathcal{F}^{-1}[\text{PSF}]$ the frequency responce to *incoherent* light. Is the autocorrelation of the ASF, and the fourier transform of the PSF.

Power Spectral Density (PSD) The amount of power in each fequency component of a signal, equivalent to $\mathcal{F}[\text{Autocorrelation of signal}]$

5.2 Background

Modelling MUSE's PSF is based on [18]. Their approach is based upon modelling the power spectral density (PSD) of the atmosphere + adaptive optics (AO) system, finding the Optical Transfer Function (OTF) of that system, and combining it with the telescope's non-AO diffraction limited OTF to get the combined PSF.

Let $R(\underline{x})$ be the PSF,

$$\begin{aligned}
\hat{R}(\underline{k}) =& \mathcal{F}[R(\underline{x})](\underline{k}) \quad \text{is the OTF of that system, and} \\
\hat{R}_T(\underline{k}) =& \mathcal{F}[R_T(\underline{x})](\underline{k}) \quad \text{the OTF of the telescope} \\
\hat{R}_A(\underline{k}) =& \mathcal{F}[R_A(\underline{x})](\underline{k}) \quad \text{the OTF of the atmosphere + AO system} \\
\therefore \hat{R}(\underline{k}) =& \hat{R}_T(\underline{k})\hat{R}_A(\underline{k}) \qquad (10)
\end{aligned}$$

where $\underline{x}$ is the light position at the exit pupil, and $\underline{k}$ is spatial wavenumber vector. Sometimes, $\underline{x}$ and $\underline{k}$ have other intepretations, but this is the simplest. Compare with [18, eq 5, eq 9]

The $\hat{R}_T$ is pretty easy to get, you just need to work out the pupil function (usually just a circle of 1's, or an annulus of 1's for a telescope with a secondary mirror), then autocorrelate it to get the OTF.

$\hat{R}_A$ is more difficult, we need to build some model for how the atmosphere and AO system influences the light. Fetick's method is to build up the PSD of this system. They use a Moffat function to model the AO-systems contribution, and kolmogorov turbulence to model the atmosphere's contribution, it's

assumed that below some cutoff spatial frequency f_{AO}, the adaptive optics are the dominant factor, and above that frequency the AO-system can't correct on such small scales so the atmosphere is the dominant factor. Therefore their model of the PSD is [see 18, eq 11, eq 2, eq 3, eq 10]

$$W_\phi(\underline{f}) = \left[\frac{\beta-1}{\pi\alpha_x\alpha_y}\frac{M_A(f_x,f_y)}{1-\left(1+\frac{f_{AO}^2}{\alpha_x\alpha_y}\right)^{(1-\beta)}}+C\right]_{f\le f_{AO}} + \left[W_{\phi,\text{kolmo}}(\underline{f})\right]_{f>f_{AO}} \tag{11}$$

where

$$\begin{aligned} M(x,y) =& \frac{A}{\left(1+\left(\frac{x}{\alpha_x}\right)^2+\left(\frac{y}{\alpha_y}\right)^2\right)^\beta} \\ A =& \frac{\beta-1}{\pi\alpha_x\alpha_y} \end{aligned} \tag{12}$$

and

$$W_{\phi,\text{kolmo}}(\underline{f}) = 0.023 r_0^{-5/3} f^{-11/3} \tag{13}$$

with α_x, α_y, β being parameters to the Moffat function [19]. C is a constant offset usually set so there is approximate continuity between the two different parts of the equation. r_0 is the *Fried Parameter*, and $\underline{f}$ is spatial frequency, with f being its magnitude.

The power of f in (13) ($-11/3$) and the coefficient (0.23) are due to the dimensionality of the turbulence, with a power of $-11/3$ for 3D, $-8/3$ for 2D, and $-5/3$ for 1D. The coefficient is fairly arbitrary, as it's the relative sizes of the AO and atmospheric parts that is important, but a value of 0.000023×10^D, where D is the dimensionality of the turbulence, gives a good continuity between the two parts of equation (11). I.e.,

$$W_{\phi,\text{kolmo}}(\underline{f};D) = 2.3\times10^{D-5} r_0^{\frac{-5}{3}} f^{\frac{3D-2}{3}} \tag{14}$$

Equation (14) was found by working through the equations for Kolmogorov turbulence for other dimensions, and comparing the $f > f_{AO}$ parts of modelled PSFs with example PSFs.

Finally, getting $\hat{R}_A$ from (11) can go two ways. In [18] they use the relationship

$$\mathcal{F}^{-1}\left[W_\phi(\underline{f})\right] = B_\phi(\rho)$$

and

$$\hat{R}_A = e^{-B_\phi(0)} e^{B_\phi(\rho)}. \tag{15}$$

However, it's also mentioned [18, section 2.3] that for extreme AO correction, the PSF = PSD. I assume this is because you can expand the exponential as a series. Therefore to first order, we can just use

$$\hat{R}_A = \mathcal{F}^{-1}\left[W_\phi(\underline{f})\right]. \tag{16}$$

I've programmed in both versions to the modelling code, but I've had most success wth (16) as it seems to fix the problem of the full treatment having a large delta-function-like spike that is not found in example PSF observations.

Once you have the value of $\hat{R}_A$, use (10) to get the full OTF and from that find the PSF.

One more thing to consider is the impact of supersampling on the modelled PSF. [18] mentions this briefly, but I found bad fits to example PSF observations when running the calculations at MUSE's native resolution. The modelled PSFs look much better when running the caclulations by supersampling at factors of 2x or even 4x and smoothing the results to the native resolution.

5.3 Why I Think Fetick's Treatment Is Being Weird

So, when I've used (15) to calculate $\hat{R}_A$ I've found a curious problem. The central lobe of the PSF is dominated by a very large central spike, this is not found in any of the observational PSFs and I couldn't find a good way to remove it without arbitrarily altering (15). Supersampling starts to mitigate the problem, and practically fixes it when you use enough of it.

> The supersampling I am refering to here is in spatial frequency. I.e., when calculating the Pupil Function use many more pixels, but the size of each pixel is the same. That way, when the PF is fourier transformed you have better resolution in spatial frequency as the maximum frequency is the same (its the spatial frequency of a single pixel), but there are many more data samples. That way you get better spatial frequency resolution.

But why does this problem occur in the first place? I think it's due to how the PSD is parameterised in (11). From reading books and articles about Kolmogorov turbulence and its interaction with optics (see [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]), what we are measuring with the PSF is the intensity of light that passes through our optical system. So, in 1D for simplicity,

$$\begin{aligned} I(x) =& \left\langle |\psi(x,t)|^2 \right\rangle_{\text{av}} \\ =& \left\langle \psi^\star(x,t)\psi(x,t) \right\rangle_{\text{av}} \\ =& \left\langle |\psi_0|^2 e^{ik(ct-x)} e^{ik(-ct+x)} \right\rangle_{\text{av}} \\ =& \psi_0^2 \quad \text{assuming } \psi_0 \text{ is real} \end{aligned}$$

where $I(x)$ is the intensity of light, $\psi(x,t)$ is the wavefunction of the light's electric field for a light wave with wavenumber k, and $\langle ... \rangle_{\text{av}}$ means to take the average of the quantity inside the brackets over whatever space is applicable (e.g. time average). For combination of light waves, you get the following relationship

(simplifying the notation to avoid having to write the brackets all the time):

$$
\begin{aligned}
I_k(x) =& \left\langle |\psi(x,t)|^2 \right\rangle_k \quad \text{avg. over wavenumber} \\
I_k(x) =& I_0(x) + I_1(x) + ... + I_n(x) \quad \text{For incoherent light,} \qquad (17) \\
I_k(x) =& |\psi_0^(x,t) + \psi_1^(x,t) + ... + \psi_n^(x,t)|^2 \quad \text{For coherent light.} \\
=& (\psi_{0,0} e^{ik_0(ct-x)} + ... + \psi_{0,n} e^{ik_n(ct-x)})(\psi_{0,0} e^{ik_0(-ct+x)} + ... + \psi_{0,n} e^{ik_n(-ct+x)}) \\
=& \sum_i \sum_j \psi_{0,i} \psi_{0,j} e^{ik_i x'} e^{ik_j(-x')} \\
=& \sum_i \psi_{0,i} e^{ik_i x'} \sum_j \psi 0, j e^{ik_j(-x')} \\
=& \mathcal{F}\left[\psi(x')\right](k) \mathcal{F}\left[\psi(-x')\right](k) \\
=& \mathcal{F}\left[\psi(x')\right](k) \mathcal{F}^*\left[\psi(x')\right](k) \quad \text{which is the autocorrelation} \\
=& \sum_i \sum_j \psi_{0,i} \psi_{0,j} e^{i\,\delta k_{ij}\, x'} \qquad (18)
\end{aligned}
$$

where $x' = ct - x$, and $\delta k_{ij} = k_i - k_j$. For incoherent lignt, the phase differences in (**??**) cancel out (they are uncorrelated, so averaging. For coherent light, the phase differences are still important.

In terms of atmospheric turbulence affecting optics, the intensity is written in terms of the *structure function* of the phase difference. A structure function describes how the differences between two quantities varies over their domain, e.g.

$$D_{gf}(t) = \int_{-\infty}^{\infty} |g(t'+t) - f(t')|^2 dt'$$

characterises the difference between two functions $f(t)$ and $g(t)$ over their domain. Also, when $g = f$ and $t = 0$, we have $D_{gg}(0) = 0$. This case is abbrivated to $D_g(t)$.

Turbulence affects the distribution of practically all of the properties of the atmosphere, one of these is the refractive index, which changes the optical path length, which leads to differences in the phase of light from a source when observed at the ground. As noted in (18) earlier, the intensity of light is just its autocorrelation. When considering the effects of turbulence on the light from a point source, we just care about how correlated the light is over our detector. Let $D_\phi(r)$ be the structure function that describes how phase of light, ϕ, varies with separation from some point described by $r = 0$. Therefore,

$$
\begin{aligned}
B_\psi(r) =& \left\langle \psi(x)\psi^\star(x+r) \right\rangle_\phi \\
=& e^{-\frac{1}{2} D_\phi(r)} \qquad (19) \\
=& e^{B_\phi(r) - B_\phi(0)} \qquad (20)
\end{aligned}
$$

describes how the intensity of light changes over different length-scales r. I.e. $B_\psi(r)$ is the fourier transform of the PSF. I.e., The PSF describes how the intensity of light varies in space, $B_\psi(r)$ descibes how the intensity of light varies as the spatial scale increases. Linking to (15), $B_\psi(r) = \hat{R}_A(\underline{k})$, but just the Kolmogorov turbulence part.

Ok, as $D_\phi(r)$ describes the difference in phase from some point $r = 0$, we can safely assume that $D_\phi(0) = 0$ by definition, and that in most cases $D_\phi(r) \to \infty$

as $r \to \infty$. Therefore, from (20) we can see that $B_\psi(0) = 1$ and $B_\psi(r) \to 0$ as $r \to \infty$. So there should not be a delta-function-like spike at the center of the PSF (as that requires a constant value as $r \to \infty$).

So, why the difference? Well, if we expand the exponential into its series representation

$$\begin{aligned} B_\phi(r) =& e^{-B_\phi(0)} \sum_{n=0}^{\infty} \frac{B_\phi^n(r)}{n!} \\ =& e^{-B_\phi(0)} + e^{-B_\phi(0)} B_\phi(r) + e^{-B_\phi(0)} frac B_\phi^2(r) 2 + ... \end{aligned}$$

we can see that as $B_\psi(r) = \hat{R}_A(\underline{k})$, and $B_\phi(r) = \mathcal{F}^{-1}\left[W_\phi(f)\right]$,

$$\begin{aligned} R_A(\underline{x}) =& e^{-B_\phi(0)} \left(\mathcal{F}^{-1}\left[1\right] + \mathcal{F}^{-1}\left[B_\phi(r)\right] + \text{smaller terms}\right), \\ =& e^{-B_\phi(0)} (\delta(\underline{x}) + \mathcal{F}^{-1}\left[\mathcal{F}^{-1}\left[W_\phi(f)\right]\right](\underline{x}) + \text{smaller terms} \\ =& e^{-B_\phi(0)} (\delta(\underline{x})) + W_\phi(\underline{x})) + \text{smaller terms} \end{aligned}$$

where $e^{-B_\phi(0)}$ is a constant, and due to the double inverse fourier transforms $\underline{x} = -\underline{f}$.

As $\delta(\underline{x})$ is an infinitestimal quantity, as we increase our numerical precison by super-sampling it will become less and less important. Similarly, the other terms of the exponential series will also be less important, and even if they never tend to zero as a first approximation we can just use (16) which is much more computationally efficient anyway.

6 PSF Asymmetries

6.1 Usage

This routine is slightly different from the others in that I've written an example bash script `run_asymmetric_psf_correction.sh` that invokes the Python code `asymmetric_psf_correction.py` that does the heavy lifting. This is because the Python code is a bit more user-friendly than the other routines in this tutorial. `run_asymmetric_psf_correction.sh` is just an example of calling the Python code from the command line, therefore I'll detail the Python code here as that's where things are happeining. Please see Tbl. 1 for argument descriptions and defaults.

6.2 Description

The main problem encountered that `asymmetric_psf_correction.py` tries to solve is that if a PSF is not completely symmetric, there can be some shifting of the images between the original FITS cube, and the CLEANed FITS cube. This usually only shifts the images by a fraction of a pixel, but this can be noticeable when comparing the original and deconvolved dataset and the images seem to 'jump' slightly when blinking between them.

Firstly the PSF is normalised by dividing by its sum, centering on the brightest pixel, removing NANs, and ensuring it has an odd shape (so there is a central pixel). Then for each wavelength, the autocorrelation of the PSF is calculated, then cubic-interpolated at `n_subdivisions` points for each pixel, and

the position of the maximum value of the interpolated autocorrelation is the fractional-pixel-coordinate.

The difference between the fractional-pixel-coordinate and the center-pixel-coordinate is then used to define the asymmetry correction which the data is re-gridded (by linearly interpolating the data at the offset points) to adjust for.

Large and small values of `window_shape` can lead to weird behaviour, I found that smaller sizes are more stable, but sometimes don't capture the entire range of adjustment needed. It's best to use a few different values and inspect the results.

My process for inspecting the results:

- Open the corrected file in a FITS viewer (DS9 or QFitsView both work well).
- Use the 'region' tool in the viewer (or some other way of providing a fixed reference) to encircle the disk.
- Scrub through the image-planes and see if there is any jitter away from the region.

If neccesary, you can also use the PSF Modelling routines (Sec. 5) to fit a symmetric PSF very closely to the original PSF (lots of iterations, low stopping thresholds) and use the fitted PSF in the deconvolution. It should be practically identical to the original PSF, but symmetric and centered as that's how the modelling routine outputs the PSF model.

7 Overall Procedure

1. Collate data.
   - Observation data
   - PSF data
2. Normalise and pre-process data.
   - Create PSF model (if needed)
   - Normalise PSF, ensure that it
     - sums to 1
     - has an odd number of rows/columns
     - is centered, usually the brightest pixel is ok, but sometimes it may have to be more exact.
     - contains no NANs, INFs, or negative values. Try interpolating across them if needed.
   - Pre-process observation data: interpolate over NANs and INFs, use SSA filtering to remove artifacts if needed, remove particularly bad artifacts by hand if needed.
   - Trim down "extra space" around observation and PSF. The fewer pixels an image has, the faster the algorithm will run.
3. Deterimine deconvolution parameters, this usually involves educated guesses and a bit of iteration.

4. Inspect results and goto step 3 to tweak parameters. Generally want to tweak them even if results are good, as you want to know how delicate the solution is. Some general classes of problems and their possible cause are:

   Speckling
   : You generally want to use one of the variable threshold modes. Try setting 0 > *threshold* > -1, *threshold*=-1, or *threshold*=-2.

   Very uncorrelated and/or a stripe accross the field
   : Usually due to a non-centered PSF i.e., the subtractions from the algorithm are not correlated with the pixels it is selecting.

   Large areas of Zeros
   : Normally means that you have to few iterations, try increasing it or try increasing the *loop gain*. Increasing the *loop gain* technically makes the algorithm less accurate, but speeds it up by the factor it was increased by.

   Very persistent problems
   : In the case you need a nuclear option, you can try smoothing the components with a gaussian (usually around the size of the core of the PSF). This has the effect of regularising the over-fitted components, removing the high-frequency noise. This is what the traditional CLEAN algorithm does to create the *clean map*, but so far for us the components have not been troublesome to use by themselves.

   Trouble getting flux/irradiance constant
   : If the total flux/irradiance of the image is varying wildly, consider adding the residual map to the components (or to the *clean map* if you need to). Usually the difference is very small, but sometimes there's information you can't ingore in the residual.

8 Other Routines

8.1 Disk Fitter (tutorial 5)

Shows a dialog that enables the user to fit an ellipse to a disk. See the usage via `python3 ./run_disk_fit.py -h`. An explanation of the interface is given in Fig. 1. The code that runs the GUI is in `.../scripts/fitscube/fit_disk.py`, and ellipse calculations are in `.../scripts/fitscube/geometry/ellipse.py`. Requres the 'astroquery' package to run properly. Use `pip3 install astroquery`, if there is already a version, ensure you have the newest version by running `pip3 install --upgrade astroquery`.

8.2 Webservices

Located in `.../scripts/webservices`, there are two python files. `jplhorizons.py` has a couple of helper-functions based on 'astroquery' to get ephemeris data from JPL-Horizons. `observatory_codes.py` holds routines that grab observatory codes from the internet and associate them with their names.

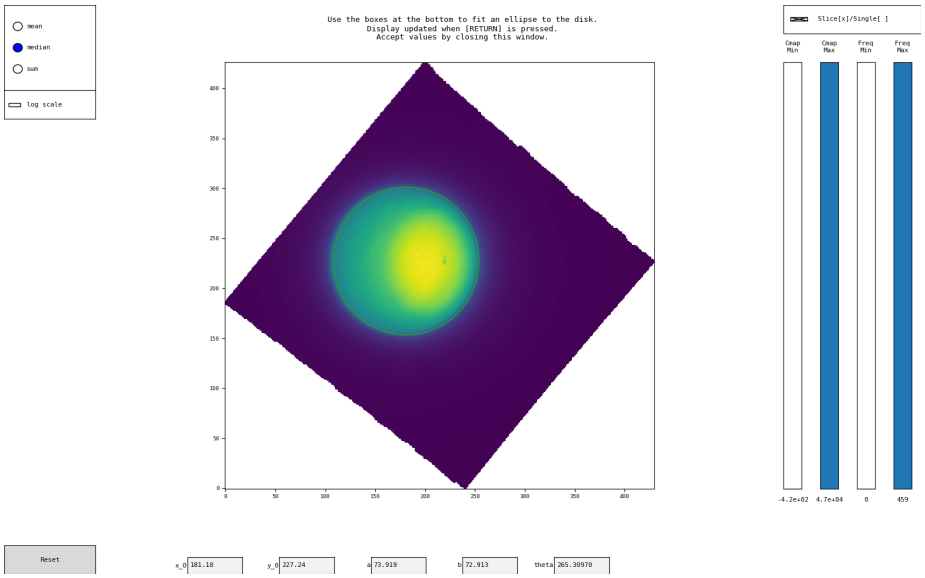

Figure 1: **Main Center Panel:** Shows the datacube with an ellipse (green, hard to see here) superimposed upon it. **Top Right:** Changes between multi and single channel mode, in multi-channel mode the image is a combination of all channels in the range as set by the selector in the top left. **Middle Right:** Colour map limits (Cmap Min, Cmap Max) and frequency channel limits (Freq Min, Freq Max). Clicking will set the relevant bar, values are shown at the bottom of the bars. Single-channel mode only shows a single frequency selector. **Top Left:** Aggregate selector swaps between combining channels via mean, median, or sum in multi-channel mode (unused in single-channel mode). Logarithmic scale can be toggled on or off. **Bottom Row:** 'Reset' button resets all controls to their defaults. 'x_0' moves the ellipse center in the x-direction, 'y_0' moves the ellipse center in the y-direction. 'a' changes the semi-major axis of the ellipse. 'b' changes the semi-minor axis of the ellipse. 'theta' rotates the ellipse. All numbers are given in pixels, fractions are possible. Close the dialog to accept the current values and continue with calculation.

8.3 Nemesis

Located in `.../scripts/nemesis` most of these routines are superceeded by NEMESISPY. However, some of them may be useful and other scripts that interface with nemesis require them.

8.4 fitscube/process

Contains processing scripts for SINFONI data. Most of it is old and obsolete.

8.5 astro_resources.py

`../scripts/astro_resources.py` contains some data about the physical parameters of planets. Used by the disk-fitting routines to get the sizes correct.

References

[1] William Hadley Richardson. Bayesian-Based Iterative Method of Image Restoration. *Journal of the Optical Society of America (1917-1983)*, 62(1): 55, January 1972.

[2] L. B. Lucy. An iterative technique for the rectification of observed distributions. *Astrophysical Journal*, 79:745, June 1974. doi: 10.1086/111605.

[3] J. L. Starck, E. Pantin, and F. Murtagh. Deconvolution in Astronomy: A Review. *PASP*, 114(800):1051–1069, October 2002. doi: 10.1086/342606.

[4] UNKNOWN AUTHOR. Lucy ricardson derivation. https://en.wikipedia.org/wiki/Richardson-Lucy_deconvolution#Derivation, 2022. [Online; accessed 2022].

[5] J. A. Högbom. Aperture Synthesis with a Non-Regular Distribution of Interferometer Baselines. *Astronomy and Astrophysics Supplement*, 15:417, June 1974.

[6] William C. Keel. A Simple, Photometrically Accurate Algorithm for Deconvolution of Optical Images. *PASP*, 103:723, July 1991. doi: 10.1086/132871.

[7] Giovannelli, J.-F. and Coulais, A. Positive deconvolution for superimposed extended source and point sources. *A&A*, 439(1):401–412, 2005. doi: 10.1051/0004-[]6361:20047011. URL `https://doi.org/10.1051/0004-[]6361:20047011`.

[8] Young-Jae Choi and In-Sik Choi. A novel fast clean algorithm using the gradient descent method. *Microwave and Optical Technology Letters*, 59: 1018–1022, 05 2017. doi: 10.1002/mop.30448.

[9] B. P. Wakker and U. J. Schwarz. The Multi-Resolution CLEAN and its application to the short-spacing problem in interferometry. *Astronomy and Astrophysics*, 200(1-2):312–322, July 1988.

[10] T. J. Cornwell. Multiscale CLEAN Deconvolution of Radio Synthesis Images. *IEEE Journal of Selected Topics in Signal Processing*, 2(5):793–801, November 2008. doi: 10.1109/JSTSP.2008.2006388.

[11] T. J. Cornwell. A method of stabilizing the clean algorithm. *Astronomy and Astrophysics*, 121(2):281–285, May 1983.

[12] D. G. Steer, P. E. Dewdney, and M. R. Ito. Enhancements to the deconvolution algorithm 'CLEAN'. *Astronomy and Astrophysics*, 137(2):159–165, August 1984.

[13] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979. doi: 10.1109/TSMC.1979.4310076.

[14] Mehmet Sezgin and Blent Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13(1):146 – 165, 2004. doi: 10.1117/1.1631315. URL `https://doi.org/10.1117/1.1631315`.

[15] Michael Ghil, M.R. Allen, Michael Dettinger, Kayo Ide, D. Kondrashov, Michael Mann, Andrew Robertson, A. Saunders, Yudong Tian, Ferenc Varadi, and P. Yiou. Advanced spectral methods for climate time series. *Rev. Geophys.*, 2002:1003–1043, 01 2002.

[16] Nina Golyandina, Anton Korobeynikov, Alex Shlemov, and Konstantin Usevich. Multivariate and 2D Extensions of Singular Spectrum Analysis with the Rssa Package. *arXiv e-prints*, art. arXiv:1309.5050, September 2013.

[17] Paul Bourke. Interpolation methods. http://paulbourke.net/miscellaneous/interpolation/, 2022. [Online; accessed 2022].

[18] Fétick, R. J. L., Fusco, T., Neichel, B., Mugnier, L. M., Beltramo-Martin, O., Bonnefois, A., Petit, C., Milli, J., Vernet, J., Oberti, S., and Bacon, R. Physics-based model of the adaptive-optics-corrected point spread function - applications to the sphere/zimpol and muse instruments. *A&A*, 628:A99, 2019. doi: 10.1051/0004-[]6361/201935830. URL `https://doi.org/10.1051/0004-[]6361/201935830`.

[19] A. F. J. Moffat. A Theoretical Investigation of Focal Stellar Images in the Photographic Emulsion and Application to Photographic Photometry. *Astronomy and Astrophysics*, 3:455, December 1969.

[20] Various. Ao summer conf. 2004 book. https://cfao.ucolick.org/aosummer/book/, 2004. [Online; accessed 2022].

[21] Paul A. Durbin. *Advanced Approaches in Turbulence*. Elsevier, 2021. ISBN 978-0-12-820774-1. doi: https://doi.org/10.1016/B978-[]0-[]12-[]820774-[]1.00007-[]0. URL `https://www.sciencedirect.com/science/article/pii/B9780128207741000070`.

[22] Patrick McMurty. Turbulence. https://my.eng.utah.edu/ mcmurtry/Turbulence/, 2022. [Online; accessed 2022].

[23] Larry C. Andrews. *Field Guide to Atmospheric Optics*. SPIE, 2 edition, 2019. ISBN 9781510619371.

[24] Robert K. Tyson. *Topics in Adaptive Optics*. IntechOpen, Rijeka, Jan 2012. doi: 10.5772/1086. URL `https://doi.org/10.5772/1086`.

[25] C. E. Coulman. Fundamental and applied aspects of astronomical seeing. *Annual Review of Astronomy and Astrophysics*, 23(1):19–57, 1985. doi: 10.1146/annurev.aa.23.090185.000315. URL `https://doi.org/10.1146/annurev.aa.23.090185.000315`.

[26] Max Born and Emil Wolf. *Principles of Optics: 60th Anniversary Edition*. Cambridge University Press, 7 edition, 2019. doi: 10.1017/9781108769914.

[27] Avijit Lahiri. *Basic Optics*. Elsevier, Amsterdam, 2016. ISBN 978-0-12-805357-7. doi: https://doi.org/10.1016/B978-[]0-[]12-[]805357-[]7.00007-[]1. URL `https://www.sciencedirect.com/science/article/pii/B9780128053577000071`.

[28] DANIEL J. SCHROEDER. *Astronomical Optics (Second Edition)*. Academic Press, San Diego, second edition edition, 2000. ISBN 978-0-12-629810-9. doi: https://doi.org/10.1016/B978-[]0-[]12-[]629810-[]9.50023-[]2. URL https://www.sciencedirect.com/science/article/pii/B9780126298109500232.

[29] Michael P. Keating. *Geometric, Physical, and Visual Optics (Second Edition)*. Butterworth-Heinemann, Burlington, second edition edition, 2002. ISBN 978-0-7506-7262-7. doi: https://doi.org/10.1016/B978-[]0-[]7506-[]7262-[]7.50005-[]9. URL https://www.sciencedirect.com/science/article/pii/B9780750672627500059.

[30] Francois Roddier. *Adaptive Optics in Astronomy*. Cambridge University Press, 1999. doi: 10.1017/CBO9780511525179.

Table 1: Arguments of `asymmetric_psf_correction.py`

Argument	Type	Number	Description	Default
`file_obs`[1]	string[2]	$N \geq 1$	Path to FITS file of observations to operate upon.	No Default, is a positional argument.
`ext_obs`	integer	$\leq N$[3]	Extension number of the observation FITS files to operate upon, associated with members of `file_obs` by position.	Will use the following named extensions if they are present, ('DATA', 'SCI'). Otherwise will use primary FITS extension.
`file_psf`	string[2,4]	$\leq N$[3]	Path to the FITS file of the PSF associated to the associated `file_obs`.	'`../⟨basename⟩_standard_star⟨ext⟩`'
`ext_psf`	integer	$\leq N$[3]	Extension number of the PSF FITS files to operate upon, associated with members of `file_psf` by position.	Will use the following named extensions if they are present, ('DATA', 'SCI'). Otherwise will use primary FITS extension.
`corr_cache`	string[2,4]	$\leq N$[3]	Path to a cache file that holds lengthy intermediate calculations associated with a `file_psf`.	'`./cache/⟨basename⟩_corr_cache.npz`'
`file_out`	string[2,4]	$\leq N$[3]	Output file associated with each `file_obs`.	'`./⟨basename⟩_asym_correction⟨ext⟩`'
`window_shape`	integer	1	Size of the window that is centered on the brightest pixel of a PSF that is used to calculate asymmetry correlation. Have to limit the window size sometimes as there can be weird effects otherwise.	51
`n_subdivisions`	integer	1	Number of sub-pixels to divide each 'real' pixel into when calculating the asymmetry correlation.	64
`index_shift_obs`	integer	$\leq N$[3]	A constant shift to apply between the `file_obs` indices and the `file_psf` when calculating the asymmetry correlation. (Mostly unused, and may need re-working at some point)	0
`dry_run`	boolean	0	A switch that, if present, will not actually perform any calculations but will print out any information messages. Useful for debugging or just to see what is supposed to happen when the script is invoked with a specific set of arguments.	False

(1) Is a positional argument. (2) Must be a string that represents a path. (3) If less elements are present than the number of strings in `file_obs` the last value will be repeated, if more elements will be truncated. (4) If starting with a '/' or '@' character, the path will be absolute or relative to the calling location, otherwise the path will be relative to the *associated file*. For ease of operation, the *associated file* basename and extension are available as special placeholders '`⟨basename⟩`' and '`⟨ext⟩`' respectively, which can be used in the path and are expanded when the arguments are read.