

MAOPPY:

Modelization of the Adaptive Optics Psf in Python

Romain J.L. FETICK
romain.fetick@lam.fr

July 15, 2024

Abstract

MAOPPY is a library written in Python. It stands for **M**odelization of the **A**daptive **O**ptics **P**SF in **P**ython. Its goal is to provide point-spread-function (PSF) models for astronomy. Although different models are implemented, the main one is `Psfao`. This model is adapted for adaptive optics corrected images, and has been described in the refereed paper [Fétick, R. et al., 2019]. This document is a user manual on how to use MAOPPY and detailing the actual implementation of `Psfao` in Python.

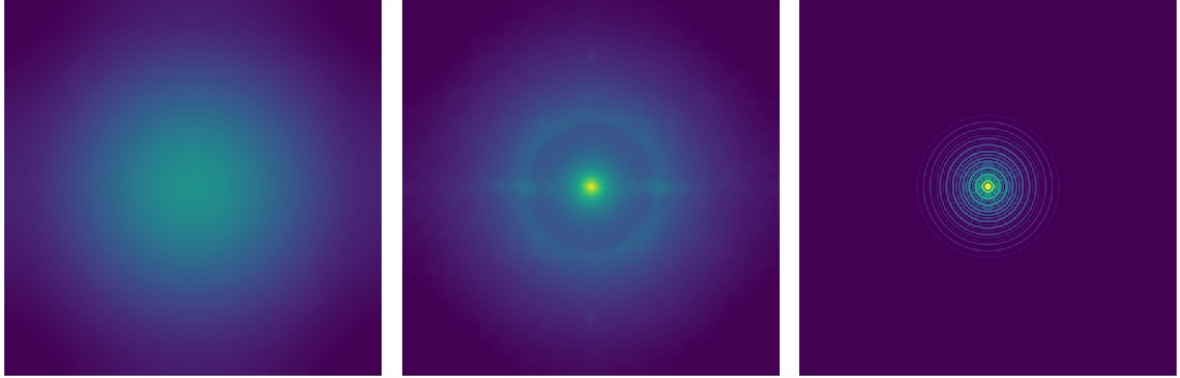


Figure 1: Simulation of three PSF using Maoppy from seeing limited (left) to diffraction limited (right) and typical AO corrected (center)

Contents

I	Installation	3
I.1	Requirements	3
I.2	Installation using PIP	3
I.3	Manual installation	3
I.3.1	Download the project folder	3
I.3.2	Add Maoppy to the Pythonpath	3
II	Understanding the Psfao model	5
II.1	Where does the Psfao model come from?	5
II.2	Main characteristics	5
II.3	Review of the Psfao parameters	5
II.4	Notes on the numerical implementation	6
II.4.1	Even arrays	6
II.4.2	Energy normalisation at infinity	7
II.4.3	Pixel centering	7
III	How to use MAOPPY	9
III.1	The "Instrument" class	9
III.2	Generate a PSF	9
III.3	Fit a PSF	10
	References	11

I Installation

I.1 Requirements

MAOPPY has been written in Python 3. It requires the libraries `numpy`, `scipy` and `astropy`. The `matplotlib` library is recommended to run examples and plot results, although it is not mandatory to run the core functions.

I.2 Installation using PIP

Make sure you have previously installed PIP and then run the PIP command `pip install git+https://gitlab.lam.fr/lam-grd-public/maoppy.git@master`. This command should have correctly installed the MAOPPY core package, however the documentation, the "how-to" examples, the readme and the license files are discarded. You may visit the website listed below if you want to have a look at them.

I.3 Manual installation

You may choose the manual installation if you don't want to use PIP.

I.3.1 Download the project folder

The library is available on the official gitlab repository of the Laboratoire d'Astrophysique de Marseille, at the following url

<https://gitlab.lam.fr/lam-grd-public/maoppy>

A zip folder can be manually downloaded, or the project can be cloned using the following `GIT` commands

```
cd my/folder/to/put/the/library/
```

```
git clone https://gitlab.lam.fr/lam-grd-public/maoppy
```

Specific releases of the code can be downloaded through the dedicated `Releases` tab. It is recommended to download the latest release. By default the version of the latest modification (`commit`) is downloaded, and not the latest release (`tag X.Y.Z`). They can slightly differ if some work has been performed on the project since the latest release. I will try to keep the latest release as close as possible to the latest modification.

I.3.2 Add Maoppy to the Pythonpath

Once the library has been downloaded, you need to add it to your `PYTHONPATH` in order to be found by Python. Different methods exist according to your operating system and development environment.

Spyder environment (any OS)

1. Go to `tools > manage PYTHONPATH > add path`
2. Add the full path to your `maoppy/` folder. Take care to add the path up to `maoppy/` included, but not to `maoppy/maoppy/`
3. Restart Spyder to take into account the new path

Windows

1. Go to Windows `Parameters`
2. Launch a new search for `variable`
3. Select `Modify environment variables > Environment variables`
4. In the list select `PYTHONPATH` and then `modify`
5. Add the full path to the `maoppy/` folder

Linux

1. Open the `.bashrc`, for example with gedit `gedit ~/.bashrc`
2. At the end of the bashrc file add the line
`export PYTHONPATH=$PYTHONPATH:$path/to/maoppy`
3. Save and close the bashrc
4. From the terminal run `source ~/.bashrc` to take modifications into account

Temporary add to path (not recommended)

1. Open your Python interpreter
2. Run the python code `import sys` and `sys.path.append("path/to/maoppy")`

II Understanding the Psfao model

II.1 Where does the Psfao model come from?

The adaptive optics corrected PSF model, called **Psfao**, is the main model of this library. It has been developed in order to describe accurately the long-exposure PSF of adaptive optics systems for astronomy. A full mathematical description of the model can be found in [Fétick, R. et al., 2019]. This paper also proves the adaptation of the PSF model to the VLT instruments SPHERE/ZIMPOL and MUSE (Narrow Field Mode). Other instruments have been successfully tested in [Beltramo-Martin et al., subm], such as Keck AO, SOUL at LBT, CANARY at the WHT and GEMS/GSAOI at GEMINI. The open access ArXiv version of these papers is available online.

II.2 Main characteristics

The main characteristics of the **Psfao** model are the following:

- telescope pupil diffraction taken into account (primary mirror aperture and secondary mirror occultation)
- telescope static phase aberrations taken into account (if they have been previously measured and provided to the PSF model)
- the two PSF zones, AO corrected and turbulent halo, are well described
- an elliptic asymmetry (such as the Moffat) is possible
- the halo can be estimated out of the field of view, in order to get a robust photometric accuracy, even on cropped or incomplete data
- manage undersampled PSF

II.3 Review of the Psfao parameters

The **Psfao** model is made of 7 parameters. The two main parameters of the model are:

- r_0 : the Fried parameter, scaling the strength of the turbulence. Higher Fried parameter means lower turbulence. Must be strictly positive.
- A : the variance of the AO system correction, measuring the quality of the correction. Higher variance means lower quality. Must be positive (partial AO correction) or null (perfect AO correction).

The shape refining parameters are:

- C : the AO area constant. A typical value of 10^{-2} has been found on SPHERE Zimpol and MUSE instruments. Must be positive (partial AO correction) or null (perfect AO correction).
- α : the Moffat frequency transition. It has a minor impact of the PSF shape and can be fixed for many application cases. Typical values range from 10^{-3} to 1. A typical value of 5×10^{-2} has been found on SPHERE Zimpol and MUSE instruments. Must be strictly positive.
- β : the Moffat power law defining the decrease of the phase PSD. Typical values range from 1.1 to 2.0. A typical value of 1.6 has been found on SPHERE Zimpol and MUSE instruments. Must be strictly greater than 1.

The asymmetry parameters are:

- r : the squared-root ratio between the major and the minor axis of the Moffat, such as $\alpha_x = \alpha r$ and $\alpha_y = \alpha/r$. Equivalently $\alpha^2 = \alpha_x \alpha_y$ and $r^2 = \alpha_x / \alpha_y$. Fixing $r = 1$ makes the PSF symmetric. Must be strictly positive.
- θ : the angle of the Moffat major axis with the horizontal axis. It has no importance if the PSF is symmetric ($r = 1$). This parameter is not bounded.

Category	Symbol	Definition	Bounds	Typical	Unit
MAIN	r_0	Fried parameter	> 0	$5 - 30$	cm
	A	AO variance	≥ 0	$0 - 10$	rad ²
REFINE	C	AO area constant	≥ 0	$0 - 10^{-2}$	rad ² m ²
	α	Moffat transition	> 0	$10^{-3} - 1$	m ⁻¹
	β	Moffat power law	> 1	$1.1 - 2.0$	—
ASYM	r	axis ratio	> 0	$0.5 - 2$	—
	θ	axis angle	no	$0 - 2\pi$	rad

Table 1: Summary of the 7 parameters of the `Psfao` model. Bounds must always be satisfied. Typical values are only an indication and can vary depending on the instrument or the atmospheric conditions. Remember that the Moffat parameters are defined in the electromagnetic phase power spectral density plane, and not in the usual PSF focal plane, so their usage might be slightly counter-intuitive at first glance.

II.4 Notes on the numerical implementation

II.4.1 Even arrays

The sizes of PSF arrays must be an even number of pixels.

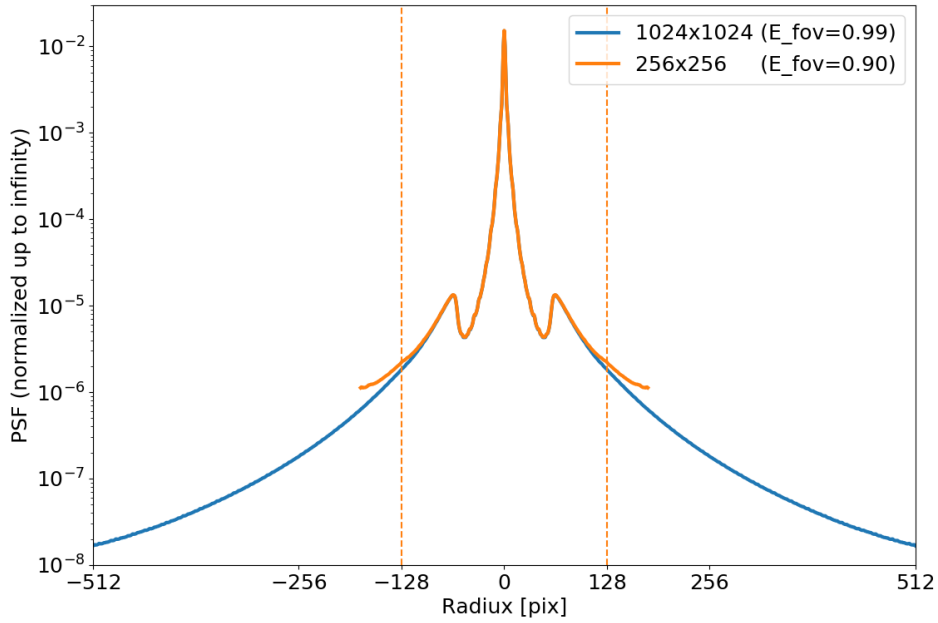


Figure 2: PSF energy normalisation at infinity. The same PSF is represented, computed on a 1024×1024 pixels array (blue) or on a reduced 256×256 pixels array (orange). The energy in the numerical field of view is $E = 0.99$ for the big array and $E = 0.90$ for the small array. The two PSF match on their common field of view, the only difference is visible at the borders of the small one due to numerical precision.

II.4.2 Energy normalisation at infinity

The PSF is normalised in energy at infinity. It means that its sum is 1 for an infinite size array. In practice, for finite size arrays, the energy is strictly smaller than one and depends on the array size. However the value of the PSF at each pixel does *not* depend on the size of the array. Note that minor differences might appear at the edges of smaller arrays due to numerical precision. See figure 2.

II.4.3 Pixel centering

For PSF sampled at least at the Shannon-Nyquist frequency ($s \geq 2$) the center position is directly $N/2$ with N the length of the array in pixels.

For PSF sampled below the Shannon-Nyquist frequency ($s < 2$), MAOPPY performs an internal oversampling in order to safely compute Fourier transforms. The array size is thus multiplied by an integer k . This integer is available through the function `oversample` of the module `maoppy.psfmodel`. It can be called such as

```

1 from maoppy.psfmodel import oversample
2 s = 0.4 # define here your sampling
3 # s = 2.3 # works also with oversampled PSF (then k=1)
4 ks,k = oversample(s)

```

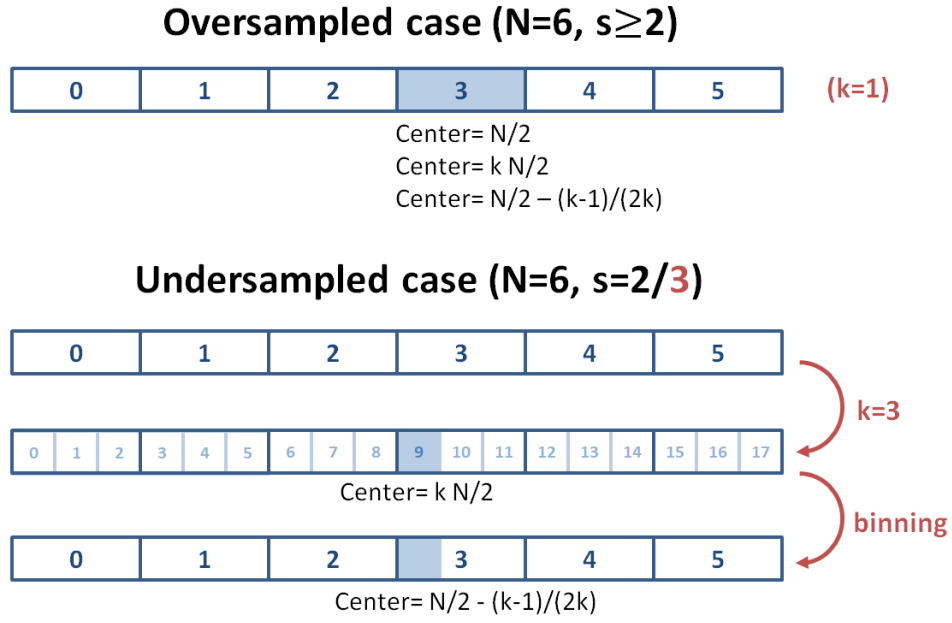


Figure 3: Definition of the numerical center of the PSF, here in the case of $N = 6$ pixels large array. The oversampling factor $k > 1$ holds for PSF sampled under the Shannon-Nyquist frequency ($s < 2$). The intermediate array is computed internally by `Psfao` and is not visible to the user. For oversampled PSFs, all centering definitions are equivalent since $k = 1$.

It returns k times the sampling, and k . Once k has been retrieved, the actual center of the PSF given by `Psfao` is

$$c = \frac{N}{2} - \frac{k-1}{2k} \quad (1)$$

Note that this formula is still valid for oversampled PSF, since $k = 1$ leads to the previous formula $c = N/2$. The detailed explanation of internal oversampling is given on Fig. 3.

III How to use MAOPPY

The `example/` folder of MAOPPY gives example scripts to start to use the library. You may want to have a look at it. The section below provides some details about these scripts.

III.1 The "Instrument" class

In order to work, the `Psfa0` model requires to be linked to an `Instrument` instance. The `Instrument` class contains all the information about the telescope (diameter `D`, central occultation `occ`), the AO system (number of actuators `Nact`) and the instrument (`resolution_rad`, `resolution_mas`, `filters`). All these attributes can be accessed by the user.

So far only the instruments `zimpol` and `muse_nfm` have been implemented, but other ones can be defined by the user. Please contact me if you want me to add a new instrument to the library. The example below shows the main characteristics of the instruments `zimpol` and `muse_nfm`.

```
1 from maoppy.instrument import muse_nfm, zimpol
2
3 print(muse_nfm) # shows info on muse_nfm
4 print(zimpol) # shows info on zimpol
```

The method `samp()` gives the sampling of the instrument at a given wavelength

```
1 wvl = 600*1e-9 # wavelength [m]
2 samp = muse_nfm.samp(wvl) # sampling for the given wvl
```

The instruments can allow spatial `binning` to adapt to binned images

```
1 zimpol.binning = 2
2 print(zimpol) # the actual resolution has been modified
```

III.2 Generate a PSF

The class `Psfa0` and the instance `muse_nfm` (of the class `Instrument`) are imported from MAOPPY. We also define the array length in pixels.

```
1 from maoppy.psfmodel import Psfa0
2 from maoppy.instrument import muse_nfm
3
4 Npix = 128 # pixel size of PSF
```

We define the sampling at which we want to generate the PSF. For a given instrument, the sampling is given by the observing wavelength. We use the `muse_nfm.samp()` function to get the sampling of MUSE NFM at the observation wavelength. Here we choose a wavelength of 600 nm.

```

1 wvl = 600*1e-9 # wavelength [m]
2 samp = muse_nfm.samp(wvl) # sampling for the given wvl

```

Then, the `Psfao` instance is created. It requires the tuple `(Npix,Npix)` to define its (X,Y) shape. The instrument is given through the mandatory keyword `system`. And the sampling is given through the mandatory keyword `samp`. The instance of `Psfao` is called `Pmodel` in this example.

```

1 Pmodel = Psfao((Npix,Npix),system=muse_nfm,samp=samp)

```

The instance of `Psfao` has been initialised and is now ready to be used. A PSF can then be generated for a given set of parameters. Let's define a set of these parameters.

```

1 r0 = 0.15 # Fried parameter [m]
2 bck = 1e-5 # background [rad2 m2]
3 amp = 5.0 # Moffat amplitude [rad2]
4 alpha = 0.1 # Moffat alpha [1/m]
5 ratio = 1.2 # alpha_x/alpha_y
6 theta = np.pi/4 # rotation angle
7 beta = 1.6 # Moffat beta power law
8
9 param = [r0,bck,amp,alpha,ratio,theta,beta]

```

The vector of parameters can be used to generate the PSF (frequent). Similarly you may want the OTF (less frequent) or the phase PSD (rare). Note that the OTF is not available for undersampled (`samp<2`) data, it is the case in this example. The `dx` and `dy` keywords are optional floats to shift the PSF and the OTF (in pixels).

```

1 psf = Pmodel(param,dx=0,dy=0) # the PSF
2 #otf = Pmodel.otf(param,dx=0,dy=0) # the OTF (not available if samp<2)
3 #psd,integral = Pmodel.psd(param) # the PSD and its integral up to infinity

```

The PSF can be plotted with `matplotlib`.

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 print('PSF energy is %.3f'%np.sum(psf))
5 ## returns 0.976
6 ## the PSF energy is normalised at infinity (for photometric robustness)
7 ## increase Npix to retrieve more energy in the field of view
8
9 plt.figure(1)
10 plt.clf()
11 plt.pcolormesh(np.log(psf))
12 plt.axis('image')

```

III.3 Fit a PSF

The MAOPPY library also provides a PSF fitting method, called `psffit`. It fits a star in an image with a PSF model, and returns the PSF parameters, the estimated flux and

background of the image.

```

1 import numpy as np
2 from maoppy.psfmodel import Psfao
3 from maoppy.instrument import muse_nfm
4 from maoppy.psffit import psffit
5
6 # Read your PSF image
7 image = ...
8
9 # Define sampling at the observing wavelength
10 wvl = 600*1e-9
11 samp = muse_nfm.samp(wvl)
12
13 # Define pixel weights for fitting
14 w = np.ones_like(image) # homogeneous weighting
15
16 var_photon = image*(image>0) # if image counts in electron
17 var_ron = muse_nfm.ron**2
18 w = 1/np.sqrt(var_photon+var_ron) # SNR weighting
19
20 w[123,456] = 0 # possibly remove some aberrant pixels
21
22 # Define the guess parameters for fitting
23 guess = [...] # see 'param' in the previous example
24
25 # Perform fitting
26 psfmodel = Psfao(image.shape, system=muse_nfm, samp=samp)
27 out = psffit(image, psfmodel, guess, weights=w)
28
29 # Get flux, background and fitted PSF
30 flux_fit, bck_fit = out.flux_bck
31 fitao = flux_fit*out.psf + bck_fit

```

References

- [Beltramo-Martin et al., subm] Beltramo-Martin, O., Fétick, R., Neichel, B., and Fusco, T. (subm). Exhaustive demonstration of an analytical point spread function model for adaptive-optics assisted optical and near infrared instruments. *Astronomy and Astrophysics*.
- [Fétick, R. et al., 2019] Fétick, R., Fusco, T., Neichel, B., Mugnier, L. M., Beltramo-Martin, O., Bonnefois, A., Petit, C., Milli, J., Vernet, J., Oberti, S., and Bacon, R. (2019). Physics-based model of the adaptive-optics-corrected point spread function - applications to the sphere/zimpol and muse instruments. *A&A*, 628:A99.