1 Getting the code  
1.1 Prerequisites ......................................................... 3  
1.1.1 Python ......................................................... 3  
1.1.2 Macintosh ...................................................... 3  
1.1.3 Ubuntu ........................................................ 3  
1.2 Installing the GLAO_PSF code ........................................ 4  

2 Running the code example ............................................ 5  

3 Looking at results ..................................................... 7  

4 Setting up a configuration file ....................................... 9  
4.1 Multi-run specification ............................................. 9  
4.2 Entries in the configuration file ..................................... 10  
4.2.1 [constellation] Define a guide star constellation ............. 10  
4.2.2 [atmosphere] Define the seeing conditions ................. 10  
4.2.3 [AO] Adaptive optics system information .................... 11  
4.2.4 [image] Imaging properties .................................... 11  
4.2.5 [output] Set the file names for storing the calculated PSFs .... 12  

5 Code source, modifications and suggestions ......................... 13  

6 References ..................................................................... 15  
6.1 Ground Layer Adaptive Optics PSF Generator ..................... 15  
6.1.1 Getting the code .................................................. 15  
6.1.1.1 Prerequisites .................................................. 15  
6.1.1.1.1 Python .................................................... 15  
6.1.1.1.2 Macintosh ............................................... 15  
6.1.1.1.3 Ubuntu .................................................... 16  
6.1.1.2 Installing the GLAO_PSF code ............................. 16  
6.1.2 Running the code example .................................... 16  
6.1.3 Looking at results ............................................... 17  
6.1.4 Setting up a configuration file ................................. 17  
6.1.4.1 Multi-run specification .................................... 18  
6.1.4.2 Entries in the configuration file ........................... 18  
6.1.4.2.1 [constellation] Define a guide star constellation ... 18  
6.1.4.2.2 [atmosphere] Define the seeing conditions .......... 18
GLAO_PSF is a Ground-Layer Adaptive Optics Structure Function and average Point Spread Function calculator. Calculating the AO-corrected and uncorrected PSFs will allow comparisons of instrument sensitivity improvements expected with AO correction under various conditions\(^1\).

This code was created in support of GLAO design studies for

- Thirty Meter Telescope
- Keck Telescopes

1.1 Prerequisites

Prerequisites vary depending on host machine, OS, and so on. Here are some tips based on experience.

1.1.1 Python

The python version must be 2.7.x. Sorry, doesn’t work yet on python v3.x, but we’re working on it.

1.1.2 Macintosh

It’s generally useful to use XQuartz to emulate a Unix environment on the mac and get an xterminal to work on. You’ll need to install Xcode, the Xcode Command Line Tools, and a system installer such as Macports, Homebrew, etc.

Next, you’ll need the pip installer:

```
sudo port install python-pip
```

HDF files are used to store a database of $C^2_\nu$ profiles determined from site surveys. It is important to have the HDF5 C library installed in order to read these file. Macs don’t come with the HDF5 library installed, so use the installer to get HDF5 1.8.4 or newer:

```
port install hdf5
```

see [http://docs.h5py.org/en/latest/build.html#source-installation-on-osx-macos](http://docs.h5py.org/en/latest/build.html#source-installation-on-osx-macos) for more information.

1.1.3 Ubuntu

From a fresh install of Ubuntu 16.04 on a virtual (Parallels) machine, it all went pretty easy. Python 2.7 and hdf5 are already installed. Unfortunately pip and Tkinter are not. So:
1.2 Installing the GLAO_PSF code

GLAO_PSF is available from the Python Package Index (PyPI):

```
pip install --upgrade glao_psf
```

Code source is maintained in a Bitbucket repository: https://bitbucket.org/donald_gavel/glao_psf/src
Running the code example

The code starts by reading parameters from a configuration file that define the atmospheric seeing conditions, AO configuration (# of guidestars etc.), and image parameters (such as wavelength). Then it calculates both the AO-corrected PSF and the open-atmosphere (not AO corrected) PSF. Finally, it writes these PSFs to FITS files.

An example config file is provided with the package (example.cfg). The code will use this config file by default if no other arguments are given.

From within python:

```python
>>> from glao_psf import psf_fe
>>> db = psf_fe.run('example.cfg')
```

```
<table>
<thead>
<tr>
<th>NAME</th>
<th>R0</th>
<th>L0</th>
<th>RADIUS</th>
<th>FIELD_PO</th>
<th>ENA_R</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PSF</td>
<td>0.188209</td>
<td>200.0</td>
<td>2.0</td>
<td>(0.0, 0.0)</td>
</tr>
<tr>
<td>1</td>
<td>PSF</td>
<td>0.188209</td>
<td>200.0</td>
<td>2.0</td>
<td>(2.5, 0.0)</td>
</tr>
<tr>
<td>2</td>
<td>PSF</td>
<td>0.188209</td>
<td>200.0</td>
<td>2.0</td>
<td>(4.0, 0.0)</td>
</tr>
<tr>
<td>3</td>
<td>PSF</td>
<td>0.188209</td>
<td>200.0</td>
<td>2.0</td>
<td>(5.0, 0.0)</td>
</tr>
<tr>
<td>4</td>
<td>${\rm PSF}_{\rm seeing}$</td>
<td>0.188209</td>
<td>200.0</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>5</td>
<td>${\rm PSF}_{\rm seeing}$</td>
<td>0.188209</td>
<td>200.0</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>6</td>
<td>${\rm PSF}_{\rm seeing}$</td>
<td>0.188209</td>
<td>200.0</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>7</td>
<td>${\rm PSF}_{\rm seeing}$</td>
<td>0.188209</td>
<td>200.0</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>
```

We get a pandas DataFrame, `db` of the run results. This shows that ENA_R, the equivalent noise area, increases as the field angle, FIELD_PO increases.

Later on, you will want to write your own config file. Look for ‘keck.cfg’ and ‘tmt.cfg’ in the code distribution (or download from this bitbucket repository). Edit a copy of one of these and put it in the directory from which you are running python.
Looking at results

The results are saved in two FITS files (nominally psf.fits, and psf0.fits, but these file names can be overridden in the [output] section of the config file).

The return value `db` is a Pandas DataFrame containing a summary of results. This is mostly useful for multi-runs (see the section on multi-runs below)

To get a quick view of the guidestar constellation:

```python
>>> psf_fe.a.constellation.graph()
```

To view cross sections of the PSFs:

```python
>>> psf_fe.a.PSF.graph()
>>> psf_fe.a.PSF_seeing.graph(opath=True)
```

Note: PSFs are normalized so as to integrate to 1, that is \( \int PSF(\theta)d^2\theta = 1 \).

Equivalent Noise Area (ENA) is calculated for both PSFs. ENA has units of square arcseconds. There is also a notion of ENA radius, defined as the radius of the circle that has area equal to ENA, having units of arcseconds. ENA radius of a seeing PSF is very close to the full-width-half-max. ENA and ENA_R are keywords in the FITS headers of the PSF files. They can also be viewed using python:

```python
>>> psf_fe.a.PSF.ena
>>> psf_fe.a.PSF.ena_r
>>> psf_fe.a.PSF_seeing.ena
>>> psf_fe.a.PSF_seeing.ena_r
```

Finally, to view the entire PSF as an image:

```python
>>> psf_fe.a.PSF.show()
>>> psf_fe.a.PSF_seeing.show()
```

To get all the above graphs and ENA summary with a single command:
>>> psf_fe.summary()
Edit a new config file (using example.cfg as a guide). There are 5 sections in the config file:

- constellation
- atmosphere
- AO
- image
- output

The format of the config file is:

```
[section_name]
parameter= value
parameter= value
...
```

Lines that begin with # are comments.

Do not put quotes around any of the values. Some parameters are numerical values followed by optional units; some parameters are just strings. One parameter image/field_point is an XY pair surrounded by parentheses.

### 4.1 Multi-run specification

Starting with version 2, parameters can have multiple values, separated by commas, as in:

```
parameter= value1, value2, value3
```

The code will collect all the multi-value specifications and from them define a set of runs, one run for each combinations of the multiple parameters. For example if one parameter has 3 values and another 4 values, then 3x4 or 12 runs will happen, each generating a PSF with a different combination of the parameters.
The multi-run creates a database (Python Pandas DataFrame), which is returned by the call to `psf_fe.run(configFile)`. This DataFrame has the parameter settings and certain of the results such as ENA (equivalent noise area of the PSF), EE_50 (50% encircled energy radius), EE_80, etc. Within Python, the DataFrame can be queried or saved to disk.

The GLAO PSFs and the corresponding seeing-limited PSFs under the same parameters will be stored as fits files in a data directory (the directory’s name is specified by the `run_name` parameter). Each fits file has header information that defines the conditions of the run. All the fits file headers are summarized in a spreadsheet file `log.csv` in that same directory.

### 4.2 Entries in the configuration file

#### 4.2.1 [constellation] Define a guide star constellation

**geom**

Constellations are “circle” or “wheel.” A circle has all the guide stars at the radius, equally spaced in angle. A wheel is a circle with one additional guide star at the center.

**radius**

The radius is in arcminutes from the center of the field.

**ngs**

Set ngs to the number of guide stars.

**rot**

Ordinarily, the constellation geometry has the first guidestar at 0 degrees position angle and the remaining stars positioned around the circle equally spaced in angle. Use rot to rotate the entire constellation on the sky by this amount in degrees.

#### 4.2.2 [atmosphere] Define the seeing conditions

**profile**

Choose from a number of built-in $C_2^2$ profiles. These include statistical average profiles measured for the TMT site survey and the Gemini GLAO survey. For TMT use:

```
profile= Maunakea 13N median
```

Maunakea 13N 25% and Maunakea 13N 75% profiles are also available.

For Keck use:

```
profile= MK2009 50%
```

25% and 75% are also available.

**outer_scale**

The outer scale of turbulence (known as $L_0$). This is the distance at which the structure function flattens out, rather than continuing up on a $r^{5/3}$ power law. There is strong evidence that at Maunakea $L_0 \sim 30$ meters on average, but this can vary over the night. The often observed

---

open seeing at Maunakea, about 1/3 arcsec when $r_0 = 20$ cm, does seem consistent with the value of $L_0 ~ 30$ m. Seeing would be about 1/2 arcsec with an infinite outer scale.

Also, $L_0$ has an ambiguous meaning when talking about $C_n^2$ altitude profiles. Recall that the argument for an outer scale is based on surface topography. So does $L_0 = 30$ meters mean only at the ground layer, while at upper altitude layers well above the ground it is much larger? For simplicity, our code models the outer scale the same at all layers, although a future version of the code may allow for specifying a separate $L_0$ for each layer.

Specify the outer scale as a positive number, the distance in meters:

```
outer_scale= 30 meters
```

For an infinite outer scale (Kolmogorov $r_0^{5/3}$ turbulence for all $r$) use Infinity as the value:

```
outer_scale= Infinity
```

$r_0$

Fried seeing parameter.

**Note:** Specifying the $r_0$ parameter *scales the $C_n^2$ profile*, so don’t use it if you want to use a measured profile, which has its $r_0$ implicit.

### 4.2.3 [AO] Adaptive optics system information

**dm_conjugate**

The deformable mirror (DM) conjugate altitude. This is the (optical conjugate) altitude at which the wavefront is corrected. Both the Keck telescope and the present design for TMT are Richey-Cretien, where the secondary mirror conjugate is at a negative altitude. If the GLAO correction is to be done with an adaptive secondary, use these values:

- TMT: $dm_{\text{conjugate}} = -280$ meters
- Keck: $dm_{\text{conjugate}} = -126$ meters

**actuator_spacing**

A deformable mirror only has a finite number of actuators on the back in order to affect wavefront correction. We model this as a spatial-frequency cutoff to the correction. Actuator spacing is defined across the pupil (not across the secondary mirror), i.e. it is a sampling on the incoming beam to the telescope. Setting the actuator_spacing to zero is allowed, which means the model assumes correction across all spatial frequencies.

### 4.2.4 [image] Imaging properties

The PSF is the image of a point source at a given wavelength. We have not modeled a sky background, however the analysis by King\(^3\) shows that Equivalent Noise Area (ENA) can be calculated from just the PSF and will provide a factor that can be used to calculate the signal-to-noise given the sky background. (See also\(^4\))

---


The wavelength at which the PSFs are calculated, in microns.

This is a list (comma delimited) of positions in the field at which the AO corrected PSF is calculated. Each point is defined by an XY pair of numbers separated by a comma and surrounded by parentheses, as in (0.,0.),(1.,0.),... The values are distances from the center (0.,0.) of the guide star constellation, in arcminutes. Typically the field_points are located within the constellation radius, but don’t have to be. The calculation will be inaccurate beyond two constellation radii, but generally there is very little GLAO correction there, and the corrected PSF is essentially the same as the uncorrected PSF.

### 4.2.5 [output] Set the file names for storing the calculated PSFs

**NOTE** None of the [output] parameters are allowed to be multi-parameters.

**run_name**

The name for the run. The results will be stored in a directory with this name.

**filename**

The prefix name for the fits files with AO-corrected PSFs. For example filename psf will generate files psf_0000.fits, psf_0001.fits etc. one for each case defined by the configuration file.

**seeing_psf_filename**

The prefix name for the files with the open-seeing PSF.
Code source, modifications and suggestions

Code source is maintained in a Bitbucket repository:
https://bitbucket.org/donald_gavel/glao_psf/src

Feel free to modify your local copy to suit your needs. The code psf_fe.py (psf “front end”) drives psf.py. I would appreciate a note from you telling me about the changes you made so they can be possibly included in future release. Also, I’m happy to entertain suggestions for new features.

Code documentation is coming soon.
6.1 Ground Layer Adaptive Optics PSF Generator

GLAO_PSF is an Ground-Layer Adaptive Optics Structure Function and average Point Spread Function calculator. Calculating the AO-corrected and uncorrected PSFs will allow comparisons of instrument sensitivity improvements expected with AO correction under various conditions\textsuperscript{1}.

This code was created in support of GLAO design studies for

- Thirty Meter Telescope
- Keck Telescopes

6.1.1 Getting the code

6.1.1.1 Prerequisites

Prerequisites vary depending on host machine, OS, and so on. Here are some tips based on experience.

6.1.1.1.1 Python

The python version must be 2.7.x. Sorry, doesn’t work yet on python v3.x, but we’re working on it.

6.1.1.1.2 Macintosh

It’s generally useful to use XQuartz to emulate a Unix environment on the mac and get an xterminal to work on. You’ll need to install Xcode, the Xcode Command Line Tools, and a system installer such as Macports, Homebrew, etc.

Next, you’ll need the pip installer:

HDF files are used to store a database of $C_2^n$ profiles determined from site surveys. It is important to have the HDF5 C library installed in order to read these file. Macs don’t come with the HDF5 library installed, so use the installer to get HDF5 1.8.4 or newer:

```
port install hdf5
```

see http://docs.h5py.org/en/latest/build.html#source-installation-on-osx-macos for more information.

### 6.1.1.1.3 Ubuntu

From a fresh install of Ubuntu 16.04 on a virtual (Parallels) machine, it all went pretty easy. Python 2.7 and hdf5 are already installed. Unfortunately pip and Tkinter are not. So:

```
sudo apt-get install pip
sudo apt-get install python-tk
```

### 6.1.1.2 Installing the GLAO_PSF code

GLAO_PSF is available from the Python Package Index (PyPI):

```
pip install --upgrade glao_psf
```

Code source is maintained in a Bitbucket repository: https://bitbucket.org/donald_gavel/glao_psf/src

### 6.1.2 Running the code example

The code starts by reading parameters from a configuration file that define the atmospheric seeing conditions, AO configuration (# of guidestars etc.), and image parameters (such as wavelength). Then it calculates both the AO-corrected PSF and the open-atmosphere (not AO corrected) PSF. Finally, it writes these PSFs to FITS files.

An example config file is provided with the package (example.cfg). The code will use this config file by default if no other arguments are given.

From within python:

```python
>>> from glao_psf import psf_fe
>>> db = psf_fe.run('example.cfg')
>>> db[['NAME','R0','L0','RADIUS','FIELD_PO','ENA_R']]
NAME   R0    L0   RADIUS  FIELD_PO  ENA_R
0      PSF 0.188209 200.0 2.0 (0.0, 0.0) 0.337167
1      PSF 0.188209 200.0 2.0 (2.5, 0.0) 0.350140
2      PSF 0.188209 200.0 2.0 (4.0, 0.0) 0.364029
3      PSF 0.188209 200.0 2.0 (5.0, 0.0) 0.369688
4 ${\rm PSF}_{\rm seeing}$ 0.188209 200.0 NaN NaN 0.455189
5 ${\rm PSF}_{\rm seeing}$ 0.188209 200.0 NaN NaN 0.455189
6 ${\rm PSF}_{\rm seeing}$ 0.188209 200.0 NaN NaN 0.455189
7 ${\rm PSF}_{\rm seeing}$ 0.188209 200.0 NaN NaN 0.455189
```

We get a pandas DataFrame, db of the run results. This shows that ENA_R, the equivalent noise area, increases as the field angle, FIELD_PO increases.
Later on, you will want to write your own config file. Look for ‘keck.cfg’ and ‘tmt.cfg’ in the code distribution (or download from this bitbucket repository). Edit a copy of one of these and put it in the directory from which you are running python.

### 6.1.3 Looking at results

The results are saved in two FITS files (nominally psf.fits, and psf0.fits, but these file names can be overridden in the [output] section of the config file).

The return value `db` is a Pandas DataFrame containing a summary of results. This is mostly useful for multi-runs (see the section on multi-runs below)

To get a quick view of the guidestar constellation:

```python
>>> psf_fe.a.constellation.graph()
```

To view cross sections of the PSFs:

```python
>>> psf_fe.a.PSF.graph()
>>> psf_fe.a.PSF_seeing.graph(opath=True)
```

Note: PSFs are normalized so as to integrate to 1, that is $\int PSF(\theta)d^2\theta = 1$.

Equivalent Noise Area (ENA) is calculated for both PSFs. ENA has units of square arcseconds. There is also a notion of ENA radius, defined as the radius of the circle that has area equal to ENA, having units of arcseconds. ENA radius of a seeing PSF is very close to the full-width-half-max. ENA and ENA_R are keywords in the FITS headers of the PSF files. They can also be viewed using python:

```python
>>> psf_fe.a.PSF.ena
>>> psf_fe.a.PSF.ena_r
>>> psf_fe.a.PSF_seeing.ena
>>> psf_fe.a.PSF_seeing.ena_r
```

Finally, to view the entire PSF as an image:

```python
>>> psf_fe.a.PSF.show()
>>> psf_fe.a.PSF_seeing.show()
```

To get all the above graphs and ENA summary with a single command:

```python
>>> psf_fe.summary()
```

### 6.1.4 Setting up a configuration file

Edit a new config file (using example.cfg as a guide). There are 5 sections in the config file:

- constellation
- atmosphere
- AO
- image
- output

The format of the config file is:
Lines that begin with # are comments.

Do not put quotes around any of the values. Some parameters are numerical values followed by optional units; some parameters are just strings. One parameter image/field_point is an XY pair surrounded by parentheses.

6.1.4.1 Multi-run specification

Starting with version 2, parameters can have multiple values, separated by commas, as in:

```
parameter= value1, value2, value3
```

The code will collect all the multi-value specifications and from them define a set of runs, one run for each combinations of the multiple parameters. For example if one parameter has 3 values and another 4 values, then 3x4 or 12 runs will happen, each generating a PSF with a different combination of the parameters.

The multi-run creates a database (Python Pandas DataFrame), which is returned by the call to `psf_fe.run(configFile)`. This DataFrame has the parameter settings and certain of the results such as ENA (equivalent noise area of the PSF), EE_50 (50% encircled energy radius), EE_80, etc. Within Python, the DataFrame can be queried or saved to disk.

The GLAO PSFs and the corresponding seeing-limited PSFs under the same parameters will be stored as fits files in a data directory (the directory’s name is specified by the `run_name` parameter). Each fits file has header information that defines the conditions of the run. All the fits file headers are summarized in a spreadsheet file `log.csv` in that same directory.

6.1.4.2 Entries in the configuration file

6.1.4.2.1 [constellation] Define a guide star constellation

```
geom
```

Constellations are “circle” or “wheel.” A circle has all the guide stars at the radius, equally spaced in angle. A wheel is a circle with one additional guide star at the center.

```
radius
```

The radius is in arcminutes from the center of the field.

```
gns
```

Set gns to the number of guide stars.

```
rot
```

Ordinarily, the constellation geometry has the first guidestar at 0 degrees position angle and the remaining stars positioned around the circle equally spaced in angle. Use rot to rotate the entire constellation on the sky by this amount in degrees.

6.1.4.2.2 [atmosphere] Define the seeing conditions

```
profile
```
Chose from a number of built-in $C_2^2$ profiles. These include statistical average profiles measured for the TMT site survey and the Gemini GLAO survey.\(^2\) For TMT use:

```
profile= Maunakea 13N median
```

Maunakea 13N 25% and Maunakea 13N 75% profiles are also available.

For Keck use:

```
profile= MK2009 50%
```

25% and 75% are also available.

**outer_scale**

The outer scale of turbulence (known as $L_0$). This is the distance at which the structure function flattens out, rather than continuing up on a $r^{5/3}$ power law. There is strong evidence that at Maunakea $L_0 \sim 30$ meters on average, but this can vary over the night. The often observed open seeing at Maunakea, about 1/3 arcsec when $r_0 = 20$ cm, does seem consistent with the value of $L_0 \sim 30$ m. Seeing would be about 1/2 arcsec with an infinite outer scale.

Also, $L_0$ has an ambiguous meaning when talking about $C_2^2$ altitude profiles. Recall that the argument for an outer scale is based on surface topography. So does $L_0 \sim 30$ meters mean only at the ground layer, while at upper altitude layers well above the ground it is much larger? For simplicity, our code models the outer scale the same at all layers, although a future version of the code may allow for specifying a separate $L_0$ for each layer.

Specify the outer scale as a positive number, the distance in meters:

```
outer_scale= 30 meters
```

For an infinite outer scale (Kolmogorov $r^{5/3}$ turbulence for all $r$) use Infinity as the value:

```
outer_scale= Infinity
```

**r0**

Fried seeing parameter.

---

**Note:** Specifying the r0 parameter scales the $C_2^2$ profile, so don’t use it if you want to use a measured profile, which has its $r_0$ implicit.

6.1.4.2.3 [AO] Adaptive optics system information

**dm_conjugate**

The deformable mirror (DM) conjugate altitude. This is the (optical conjugate) altitude at which the wavefront is corrected. Both the Keck telescope and the present design for TMT are Richey-Cretien, where the secondary mirror conjugate is at a negative altitude. If the GLAO correction is to be done with an adaptive secondary, use these values:

- TMT: `dm_conjugate= -280 meters`
- Keck: `dm_conjugate= -126 meters`

---

actuator_spacing

A deformable mirror only has a finite number of actuators on the back in order to affect wavefront correction. We model this as a spatial-frequency cutoff to the correction. Actuator spacing is defined across the pupil (not across the secondary mirror), i.e. it is a sampling on the incoming beam to the telescope. Setting the actuator_spacing to zero is allowed, which means the model assumes correction across all spatial frequencies.

6.1.4.2.4 [image] Imaging properties

The PSF is the image of a point source at a given wavelength. We have not modeled a sky background, however the analysis by King\(^3\) shows that Equivalent Noise Area (ENA) can be calculated from just the PSF and will provide a factor that can be used to calculate the signal-to-noise given the sky background. (See also\(^4\))

wavelength

The wavelength at which the PSFs are calculated, in microns.

field_points

This is a list (comma delimited) of positions in the field at which the AO corrected PSF is calculated. Each point is defined by an XY pair of numbers separated by a comma and surrounded by parentheses, as in (0.,0.),(1.,0.),... The values are distances from the center (0.,0.) of the guide star constellation, in arcminutes. Typically the field_points are located within the constellation radius, but don’t have to be. The calculation will be inaccurate beyond two constellation radii, but generally there is very little GLAO correction there, and the corrected PSF is essentially the same as the uncorrected PSF.

6.1.4.2.5 [output] Set the file names for storing the calculated PSFs

NOTE  None of the [output] parameters are allowed to be multi-parameters.

run_name

The name for the run. The results will be stored in a directory with this name.

filename

The prefix name for the fits files with AO-corrected PSFs. For example filename psf will generate files psf_0000.fits, psf_0001.fits etc. one for each case defined by the configuration file.

seeing_psf_filename

The prefix name for the files with the open-seeing PSF.

6.1.5 Code source, modifications and suggestions

Code source is maintained in a Bitbucket repository:

https://bitbucket.org/donald_gavel/glao_psf/src


Feel free to modify your local copy to suit your needs. The code psf_fe.py (psf “front end”) drives psf.py. I would appreciate a note from you telling me about the changes you made so they can be possibly included in future release. Also, I’m happy to entertain suggestions for new features.

Code documentation is coming soon.

6.1.6 References

6.2 API

Programmer’s guide - work in progress.

6.2.1 Main modules

6.2.1.1 Test_API

The dummy “testapi” module  A test of API documents generation on Readthedocs.

class GeneralObject
    A generic object
    a_method ()
    This is one of the methods

Some text after the automodule command.

6.2.1.2 psf_fe

PSF generator Front End

The front end acts to translate a run config file, or a run specification dictionary, into a form that can drive a ‘run’. The run involves calling the PSF generator multiple times and saving each resulting PSF into a FITS file. A pandas database is returned, and a csv file is written, that summarizes the results of the run.

Contents

- psf_fe
  - Functions
    * read config
    * write config
    * run
    * summary
  - Classes
    * Coordinator
6.2.1.2.1 Functions

6.2.1.2.1.1 read config

read_config(configFile='example.cfg', format='native', verbose=True)
Read the config file and generate a dictionary that can be sent as an argument to run()

Parameters
- configFile (str) – the configuration file name
- format (str) – file format (‘xml’, ‘json’, or ‘native’ where ‘native’ is the format described in the user docs)
- verbose (bool) – be chatty about it

The configFile can also be a file-like object. To send a string as if it were a file, use StringIO:

```python
>>> d = read_config(StringIO(s),...)
```

6.2.1.2.1.2 write config

write_config(d, configFile=None, format='native', clobber=False)
Write a config file given the dictionary of configs

Parameters
- d (dict) – dictionary with configuration parameters
- configFile (str) – the config file name to write to. If None then return a string of text that would have been written to the file
- format (str) – file format (‘xml’, ‘json’, or ‘native’ where ‘native’ is the format described in the user docs)
- clobber (bool) – whether or not to over write an existing file. The default is False for safety

If configFile is None then return the string that would have been written to the file

6.2.1.2.1.3 run

run(config=None, save_result=True, verbose=True)
Generate a PSF or set of PSFs using either the config file or a dictionary of args. The configuration typically specifies a set of PSF calculation, varying one parameter at a time. The results are saved in a sequence of FITS files stored in a directory named by the configuration’s run_name. A run-summarizing spreadsheet “log.csv” is also written to this directory.

Parameters
- config (str, or, dict) –
  - If str, the name of the config file.
  - If dict, a dictionary describing the configuration, such as that created by read_config(), which is used instead of a config file

If config = None, then the default config file, distributed with the package, is used.)
• \texttt{save\_result} (bool) – whether or not to save the resulting PSF calculations in FITS files. This would only be turned off in, say, a single-case test run

• \texttt{verbose} (bool) – be chatty about it

\textbf{Returns} a \texttt{pandas.DataFrame} that summarises the run results.

As a side effect, the last computed result \texttt{psf.Cn2\_Profile} is stored in the global \texttt{psf\_fe.a}.

\subsection{summary}

\texttt{summary}(a=None, pdfFile=None)

Print and plot results from a PSF calculation.

\textbf{Parameters}

\begin{itemize}
  \item \texttt{a} (\texttt{psf.Cn2\_Profile}) – the result from a PSF calculation run. See \texttt{run()}. If \texttt{a} is \texttt{None} then use the global \texttt{psf.Cn2\_Profile} object \texttt{psf\_fe.a} that resulted from the most recent PSF calculation.
\end{itemize}

\subsection{Classes}

\subsubsection{Coordinator}

\texttt{Coordinator}(run\_name, psf\_basename='psf', seeing\_psf\_basename='psf0', ext='fits')

A Coordinator object orchestrates the setup, running, and storage of results for glao_psf runs. It manages a directory of FITS files and its database.

\textbf{Parameters}

\begin{itemize}
  \item \texttt{run\_name} (str) – name of the run
  \item \texttt{psf\_basename} (str) – prefix, before the sequence number, of the PSF FITS file names
  \item \texttt{seeing\_psf\_basename} (str) – prefix, before the sequence number, of the Seeing PSF FITS file names
  \item \texttt{ext} (str) – extension for the FITS file names (like ‘fits’, with no dot)
\end{itemize}

\textbf{filelist}

Returns a list of the names of the PSF files saved so far.

\texttt{save}(a, verbose=True)

Save one of the PSF results into a FITS file.

\textbf{Parameters}

\begin{itemize}
  \item \texttt{a} (\texttt{psf.Cn2\_Profile}) – the \texttt{Cn2\_Profile} containing the PSF results
\end{itemize}

\subsection{psf}

The Point Spread Function generator module. Structure function and average PSF calculator for astronomical imaging
6.2.1.3.1 Functions

6.2.1.3.1.1 available

6.2.1.3.1.2 fwhm

\[ fwhm(f) \]

determine full-width-half-max of a lineout

Parameters

\[ f (\text{numpy.ndarray}) \] – a 1-D array of data

Returns

FWHM in pixels

6.2.1.3.1.3 fwhm2

\[ fwhm2(f, \text{method=`area'})) \]

find the full-width-half-maxima of a 2d blob

Parameters

- \[ f (\text{numpy.ndarray}) \] – a 2-D array of data
- \[ \text{method (str)} \] – ‘area’ or ‘slice’ or ‘default’

Returns

a 2-tuple with the major and minor diameters of the fit to the FWHM contour, in pixels

6.2.1.3.2 Classes

6.2.1.3.2.1 Cn2_Profile

\[ \text{class Cn2_Profile (profile=`Simple 1', **kwargs)} \]

The Profile class defines the atmospheric profile

Instances contain

- Cn2 profile vs altitude
- the normalized Cn2 (normalized to sum to one)
- normalized altitude (normalized to mean height)
- computed r0, theta0
Several valid profiles are pre-defined, but the caller can also specify with a name, Cn2 and h arguments.

‘HV’ profiles require keyword arguments r0 and theta0 and dh. Other profiles can optionally contain an r0 argument, in which case the profile is normalized (Cn2 is changed) to compute to that r0.

```graph
(which='structure', type='line', over=False, linestyle=None, color=None, linewidth=None, label='", dots=False, normalized_r=False, pdfFile=None)
```

graph one of the characteristic functions, Structure function or Correlation function.

**Parameters**

- **which**, *(str)* – ‘structure’ or ‘correlation’
- **type**, *(str)* – the type of graph, either ‘line’ (lineout) or ‘grey’ (2-d image greyscale)
- **over**, *(bool)* – for lineout, this allows multiple overplots. First call should be False, then True for subsequent overplots
- **label** *(str)* – if multiple overplots, the text for a legend (call plt.legend() after the last overplot)

**make_PSF** *(w, constellation=None, **kwargs)*

Make the PSF (point spread function), and incidentally, the MTF (modulation transfer function) given a set of weights on the guide star measurements.

**Parameters**

- **w**, *(list)* – guide star weight vector (size equal to number of guidestars). The weights must sum to one.
- **constellation**, *(Constellation)* – guide star constellation
- **kwargs** – keyword arguments to the Constellation constructor and to the S matrix generator [make_S](#)

**make_PSF0** *(n=512, r_lim='calc')*

Make the diffraction-limited and seeing limited PSFs and MTFs.

**make_S** *(constellation, r_lim='calc', n=512, dm_conjugate=0.0, wfs_conjugate=0.0, act_spacing=None)*

Make the S matrix given the guide star constellation (see reference¹ for the definition of the S matrix).

**Parameters**

- **constellation**, *(Constellation)* – a constellation object
- **r_lim**, *(float)* – the upper limit to the shift, in meters. Default is to calculate this internally based on the guide star separation and telescope size
- **n**, *(int)* – the fine grid (number of fine pixels from -r_lim to +r_lim in the structure functions)
- **dm_conjugate**, *(float)* – conjugate altitude of the DM, in meters
- **wfs_conjugate**, *(float)* – conjugate altitude of the wavefront sensor, in meters
- **act_spacing**, *(float)* – the actuator spacing, in meters (optional if modeling the DM Nyquist cutoff).

**make_profile** *(profile, **kwargs)*

Make a Cn2(h) profile by reading it from a database or using a precomputed internal one.

**Parameters**

- **profile**, *(str)* – one of the valid profiles

---

See available()

profile_fromDatabase(profile, dbFile='Cn2_profiles.hd5')
Read in a profile from a database of profiles argument:

Parameters

• profile (str) – string of the form `<site> <tile>`. For example ‘Armazones median’
or ‘Maunakea 13N 75%’
• dbFile (str) – the name of the HDF5 file that contains seeing profiles

profile_plot(altscale='log', cn2scale='log', drawstyle='steps', pdfFile=None)
plot the profile and cumulative Cn2 profile

Parameters

• altscale (str) – altitude scale; can be ‘linear’ or ‘log’
• cn2scale (str) – Cn2 axis scale; can be ‘linear’ or ‘log’
• drawstyle (str) – draw style for the cumulative graph: can be ‘default’ or ‘steps’ (Cn2
draw style is bar, with dot at end)

r0_calc()
Compute r0 from the profile

showPSF(scale='log', which='both', pdfFile=None)
Display PSF, and optionally, PSF seeing, with (optionally) the Equivavlent Noise Area circled

Parameters

• scale (str) – ‘linear’ or ‘log’
• which (str) – ‘AO’, ‘seeing’, or ‘both’

theta0_calc()
Compute theta0 from the profile

6.2.1.3.2.2 Constellation

class Constellation(ngs=1, radius=10, geometry='wheel', rot=0.0, field_point=[0.0, 0.0])
Defines the locations of the guide stars

Parameters

• ngs (int) – number of guide stars
• radius (float) – radius of guide star constellation, arcmin
• geometry (str) – ‘circle’, ‘wheel’, ‘grid’ (‘wheel’ is a circle with one guidestar in the
  middle) default is ‘wheel’
• rot (float) – rotation of the constellation on-sky, degrees
• field_point (float) – position of the science field position, arcmin

graph(field_points=None, colors={}, pdfFile=None)
Graph the constellation of guidestars and the field point

Parameters Field_points (list) – a list of [x,y] field points (of type float, in arcsec)
to graph instead of the one associated with the constellation
6.2.2 Supporting modules

6.2.2.1 info_array

Contents

- info_array
  - Functions
    * load
    * show
    * whatsin
  - Classes
    * InfoArray
    * InfoMatrix

6.2.2.1.1 Functions

6.2.2.1.1.1 load

load(filename, name=None, group='/', file_type='hdf5')
Read in an InfoArray from a disk file

Parameters

- filename (str) – the name of the file, including the extension
- name (str) – in the case of HDF files, the internal object name
- group (str) – in the case of HDF files, the group name
- file_type (str) – the default file type to use if the extension is not recognized

Type load(‘?’) to get a list of recognized extensions and supported file types

See also:
info_array.InfoArray.save()

6.2.2.1.1.2 show

show(arg, nmax=4)
Display a list or grid of InfoArrays all on one figure.

argument: arg = and InfoArray, list of InfoArrays, or list of lists of InfoArrays (2d grid)
keyword argument: nmax = the maximum number of plots on a figure in each direction
6.2.2.1.3 whatsin

whatsin (filename)
List the directory of an HDF file.

6.2.2.1 Classes

6.2.2.1.2 InfoArray

class InfoArray
A sub class of numpy.ndarray, with useful additional attributes like name, dx, and units.

(See numpy array subclassing)
to use:

```python
a = InfoArray(np.zeros((2,2)),name='blah')
a.dx = ...
```

InfoArrays are containers in that they can contain any number of attributes. Useful ones are:

- **name** (str) The name of the array
- **units** (str) The units of the values in the array
- **dx** (float) The physical size of one pixel in the array
- **dx_units** (str) The units of the pixel size
- **axis_names** (list) A list of strings containing the axis names. The first and second are the horizontal and vertical axes correspondingly. An optionall third is the radial (for radial average plots).
- **x0y0** (list) Physical position of the [0,0] pixel (defaults to -shape*dx/2)

`calc_ee` (p)
Calculate the radius containing a given encircled energy (assuming the data represent a point spread function)

p = the encircled energy fraction, a number between 0 and 1

`calc_ena` ()
Calculate the Equivalent Noise Area, assuming the data represent a point spread function (PSF). Reference: King, PASP, v95, 1983

**Formula:** \( ENA = \frac{dA}{\sum_i f_i^2} \)

where dA is the area of a pixel (dx^2), and f is the psf normalized to integrate to unity. The PSF is assumed real.

`copy` ()
Return a deep copy of the InfoArray

**Return type** InfoArray

`crop` (shape, p0=None)
Crop the image to a different size than the present image. This different size can be larger or smaller than the original image. p0 is an optional lower left hand corner for the crop

`minmax` ()
Calculate the minimum and maximum of the array

**Returns** a list of [minimumm, maximum]
mtf()
Compute the MTF (Modulation Transfer Function) assuming that the data are the PSF (Point Spread Function). It is assumed that the PSF is real

oft (upsamp=1)
Performs the Optical Fourier Transform on the array and returns a new InfoArray object. The object data are treated as optical path distances (OPD). An InfoArray object representing the aperture throughput can be included as an instance object, otherwise a default unit disk is used.

The formula is \( \text{PSF} = F \{ \text{ap} \exp \{ i \left( \frac{2 \pi}{\text{wavelength}} \right) \text{OPD} \} \} \)

OFT assumes that the object has the following instance variables, and if it does not, then default ones are set up:
- \( \text{wavelength} \) = wavelength of the light, in microns (default: 0.5)
- \( \text{ap} \) = an InfoArray object that defines the aperture (default: a circular disk that assures Nyquist sampling)
- \( \text{units} \) = the units of the wavefront (default: ‘meters’)
- \( \text{dx_units} \) = the units of \( dx \) (default: ‘meters’)

plot (kind='image', fontsize=14, grid=False, oplot=False, scale='lin', fuzz=1e-09, **kwargs)
Display the InfoArray data as an image or lineout plot

Parameters
- \textbf{kind} (str) – plot kind: ‘image’, ‘lineout’, or ‘radial’
- \textbf{fontsize} (int) – for the titles, axis labels, and tick labels
- \textbf{grid} (boolean) – draw axis grid
- \textbf{oplot} (boolean) – whether or not to plot over a previous graph (only for line plots)
- \textbf{fuzz} (float) – a minimum value for log scale to keep the image contrast reasonable (used only for ‘image’, scale=’log’)
- \textbf{**kwargs} – is sent on to dimg.show in the kind=’image’ case

pprint (full=False)
Pretty-print the object. This consists of reporting on the object’s attributes.

Parameters \textbf{full} (boolean) – if True, \texttt{pprint} also prints out the contents of the array. Default is to print only the “header” (attributes) information.

psf()
Compute the PSF (Point Spread Function) assuming the data are the MTF (Modulation Transfer Function). It is assumed that the MTF is symmetric so that the PSF is real

resample (dx)
Resample the image so as to have a different \( dx \). This will increase or decrease the dimensions of the data.

resize (shape)
Resize the image by resampling. This will change the sampling \( dx \). It is prevented from changing the aspect ratio.

save (filename=None, group=None, file_type=None, clobber=False, verbose=False)
Write an InfoArray to a disk file. Only hdf5 and FITS file types are presently supported. The file type is derived from the filename extension (*.hdf5, *.fits); the keyword parameter file_type overrides.

Parameters
- \textbf{filename} (str) – name of the file
- \textbf{group} (str) – in the case of HDF file, the group name
• **file_type** *(str)* – one of the valid filetypes

• **clobber** *(bool)* – if the file already exists, overwrite it (default is False)

• **verbose** *(bool)* – talk about it

**HDF5:**

Create a group within the hdf5 file (creating a new file if necessary) and store the data and metadata (InfoArray attributes) as datasets in the group, named ‘data’ and ‘metadata’ respectively. The group is named the same as the name of the InfoArray. Name uniqueness is checked within an existing group and will not overwrite an existing dataset of the same name.

Storing into a subgroup is possible using the tree structure syntax of the group parameter.

**FITS:**

For FITS files, create the header from the InfoArray attributes and store the data as a single primary HDU. The filename is an optional parameter. The filename is based on the InfoArray’s name (<name>.fits) if filename is not specified.

See also:

```
info_array.load()
```

### 6.2.2.1.2.2 InfoMatrix

**class InfoMatrix**

A sub class of `numpy.matrix`, with useful additional attributes like `name` and `units`.

(See numpy array subclassing)

to use:

```
a = InfoMatrix(np.zeros((2,2)), name='blah')
a.units = ...
```

InfoArrays are containers in that they can contain any number of attributes. Useful ones are:

• **name** *(str)* The name of the array

• **units** *(str)* The units of the values in the array

• **dx** *(float)* The physical size of one pixel in the array

• **dx_units** *(str)* The units of the pixel size

• **axis_names** *(list)* A list of strings containing the axis names. The first and second are the horizontal and vertical axes correspondingly. An optionall third is the radial (for radial average plots).

• **x0y0** *(list)* Physical position of the [0,0] pixel (defaults to -shape*dx/2)

**copy** ()

Make a deep copy of the InfoMatrix

**plot** *(kind=’image’, fontsize=14, grid=True, oplot=False, scale=’lin’, fuzz=1e-09, **kwargs)*

Display the InfoMatrix data as an image or lineout plot

Parameters

• **kind** *(str)* – plot kind: ‘image’, ‘lineout’, or ‘radial’

• **fontsize** *(int)* – for the titles, axis labels, and tick labels

• **grid** *(boolean)* – draw axis grid
• **oplot** (*boolean*) – whether or not to plot over a previous graph (only for line plots)


• **fuzz** (*float*) – a minimum value for log scale to keep the image contrast reasonable
  (used only for ‘image’, scale=’log’)

• ****kwargs – is sent on to dimg.show in the kind=’image’ case

**pprint** (*full=False*)
Pretty-print the InfoMatrix. This consists of reporting on the object’s attributes.

**Parameters**

**full** (*boolean*) – if True, **pprint** also prints out the contents of the array. Default is to print only the “header” (attributes) information.
g

glao_psf.info_array, 27
glao_psf.psf, 23
glao_psf.psf_fe, 21
glao_psf.testapi, 21
Index

<table>
<thead>
<tr>
<th>Letters</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>a_method() (GeneralObject method), 21 Actuator spacing, 11, 19</td>
</tr>
<tr>
<td>C</td>
<td>calc_ee() (InfoArray method), 28 calc_ena() (InfoArray method), 28 Cn2 profile, 10, 18 Cn2_Profile (class in glao_psf.psf), 24 Code source, 12, 20 Configuration file, 8, 17 Constellation (class in glao_psf.psf), 26 Coordinator (class in glao_psf.psf_fe), 23 copy() (InfoArray method), 28 copy() (InfoMatrix method), 30 crop() (InfoArray method), 28</td>
</tr>
<tr>
<td>D</td>
<td>DM conjugate, 11, 19</td>
</tr>
<tr>
<td>E</td>
<td>ENA Equivalent Noise Area, 11, 20 Example run, 4, 16</td>
</tr>
<tr>
<td>F</td>
<td>filelist (Coordinator attribute), 23 fwhm() (in module glao_psf.psf), 24 fwhm2() (in module glao_psf.psf), 24</td>
</tr>
<tr>
<td>G</td>
<td>GeneralObject (class in glao_psf.testapi), 21 Getting the code, 1, 15 glao_psf.info_array (module), 27 glao_psf.psf (module), 23 glao_psf.psf_fe (module), 21 glao_psf.testapi (module), 21 graph() (Cn2_Profile method), 25 graph() (Constellation method), 26</td>
</tr>
<tr>
<td>H</td>
<td>HDF files, 3, 16</td>
</tr>
<tr>
<td>I</td>
<td>InfoArray (class in glao_psf.info_array), 28 InfoMatrix (class in glao_psf.info_array), 30 Installation, 4, 16</td>
</tr>
<tr>
<td>L</td>
<td>load() (in module glao_psf.info_array), 27</td>
</tr>
<tr>
<td>M</td>
<td>make_profile() (Cn2_Profile method), 25 make_PSF() (Cn2_Profile method), 25 make_PSFO() (Cn2_Profile method), 25 make_S() (Cn2_Profile method), 25 minmax() (InfoArray method), 28 mtf() (InfoArray method), 29</td>
</tr>
<tr>
<td>O</td>
<td>ooft() (InfoArray method), 29 Outer scale, 10, 18</td>
</tr>
<tr>
<td>P</td>
<td>plot() (InfoArray method), 29 plot() (InfoMatrix method), 30 pprint() (InfoArray method), 29 pprint() (InfoMatrix method), 31 profile_fromDatabase() (Cn2_Profile method), 26 profile_plot() (Cn2_Profile method), 26 psf() (InfoArray method), 29</td>
</tr>
<tr>
<td>R</td>
<td>r0_calc() (Cn2_Profile method), 26 read_config() (in module glao_psf.psf_fe), 22 References, 13, 21 resample() (InfoArray method), 29 resize() (InfoArray method), 29 run() (in module glao_psf.psf_fe), 22</td>
</tr>
</tbody>
</table>
S
save() (Coordinator method), 23
save() (InfoArray method), 29
Seeing, 10, 18
show() (in module glao_psf.info_array), 27
showPSF() (Cn2_Profile method), 26
Source code, 12, 20
summary() (in module glao_psf.psf_fe), 23

T
theta0_calc() (Cn2_Profile method), 26

W
whatsin() (in module glao_psf.info_array), 28
write_config() (in module glao_psf.psf_fe), 22