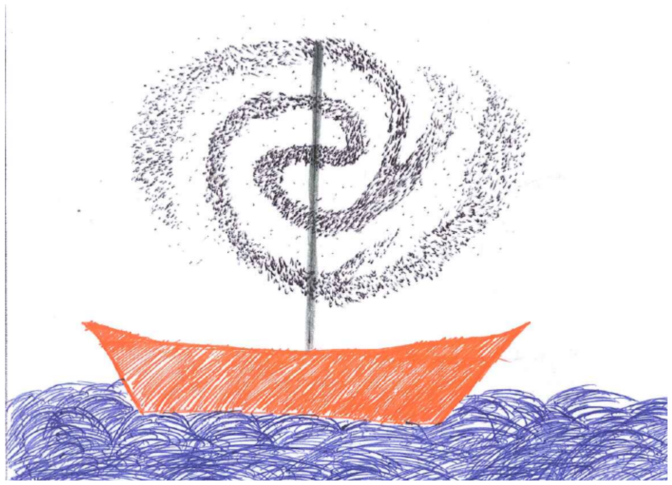
# ABISM

Adaptive Background Interactive

Strehl Meter



BY

Julien Girard, Antoine Mérand, Martin Tourneboeuf

November, 2013
Santiago, Chile

# Contents

# Chapter 1

# Introduction

There are many tools to measure the Strehl ratio yet. Why a new one ?
ABISM claims to be the most user friendly Strehl meter, no configuration is needed:

- The 4 needed parameters (wavelength, pixel scale, diameter and obstruction) are read out from the fits header.
- The apertures for background, photometry is automatically assessed from a fit.
- Correction is made for bad pixels[1].
- Some labels and check plots are displayed for the user to estimate the validity of the result.

ABISM is a GUI written in Python.

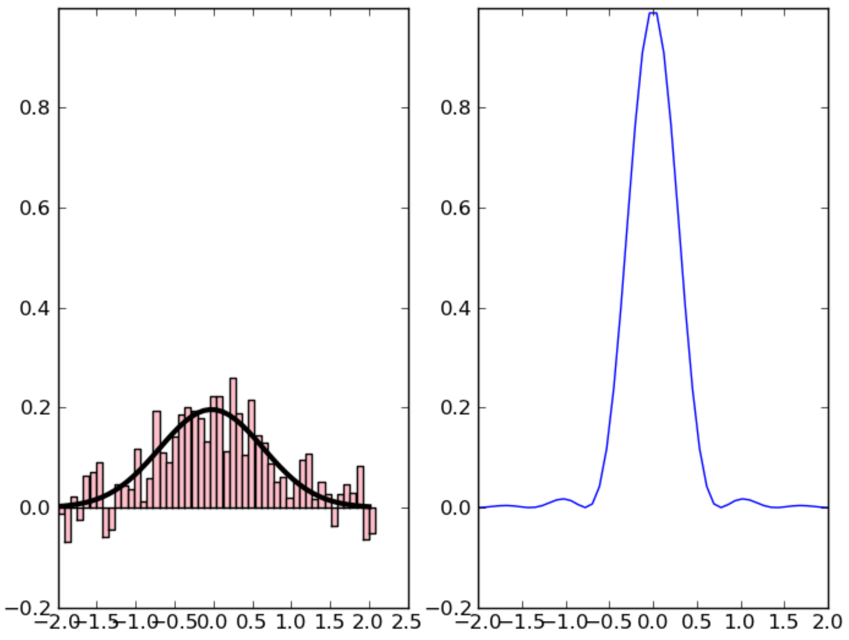
## 1.1 What is the Strehl ratio ?

The Strehl ratio (here after SR) is defined as the peak intensity of a measured PSF to the peak intensity of a perfect diffraction limited PSF for the same optical system (aperture + obstruction):

$$Sr = \frac{I(x=0)}{P(x=0)} \tag{1.1}$$

Where x stands for the position vector on the image on a PSF centred on 0; I(x = 0) for the maximum intensity of the measured PSF and P(x = 0) for the maximum of the perfect diffraction limited PSF. In Figure 1.1, it's the ratio between the maximum of left curve and the maximum right curve.

[1]..in the background, not implemented yet in the photometry where the deviation from the fit should be the criterium of the badness of a pixel



## 1.2 What ABISM does ?

A fit (Moffat by default) is performed over the selected area. The peak of the PSF is measured by the fit. An elliptical aperture is also deduced from the fit in order to get 99% of the flux[2]. The photometry is performed in this aperture and the background is computed in an annulus between 1.3 and 1.6 times the photometric aperture and with a 3 sigma kappa clipping. From the photometry, peak, wavelength, pixel scale, diameter of the primary mirror, and the central obstruction, the Strehl is calculated. No Fourier transform is performed, but an analytical expression[3] links the photometric of the object to the peak it would have in a diffraction limited system.

### 1.2.1 What you have to do

- Open ABISM (bash Abism.sh [image.fits] or >python Abism.py [image.fits] [params]
- Open ONE image.fits in command line following the script or from the GUI:file→open
- DRAW a rectangle around your star. You can make a color cut [in the View menu] if you cannot distinguish your star very well.

[2]For a divergent Moffat (b<1), the aperture was setted by experience comparing with the aperture given by a Gaussian fit. Note that for the same PSF, the best aperture should depend on the S/N. But it doesn't in Abism so low S/N sources have a high uncertainty because the aperture may be "too" large.

[3]in Strehl.py and in Appendix.

- Et voilà ! It work ? If not, the software doesn't know the parameters (Wavelength, diameter...), give it in image parameters. Don't forget to enter <Return> after giving your numbers or to click on ImageParameters button once more.

### 1.2.2 What you can do

You can modify the way of measuring photometry, background, and the fit type. More information is provided in Chapter

### 1.2.3 What you don't have to do

As ABISM reads automatically the header, so you don't have to enter the:

1. Wavelength [$\mu m$]
2. Diameter [m]
3. Central Obstruction [%](In % of size: for example a 1m m2 on a 10m m1 gives a percentage of 10%. all previous parameters are employed in the diffraction pattern calculation)
4. Pixel Scale [arcsec/pixel] (utilized to transform FWHM in arcsec)

Its are the only parameters to calculate the Strehl ratio, fortunately its are always in the Header. If the image is not from ESO, you will have to enter it manualy clicking on Image Parameters or permanently changing the Python module ReadHeader.py.

# Chapter 2

# Install

## 2.1 Packages

The following python packages are necessary to run ABISM:

| | | |
|---|---|---|
| 1 | **matplotlib** | for the image display and interaction |
| 2 | **tkinter** | for the GUI |
| 3 | **pyfits** | to open fits images, convert its to an array and a header |
| 4 | **scipy** | for the bessel J1 function |
| 5 | **pywcs** | to get the WCS projection from the header of the image |

And you can install all following these commands for a rpm based Linux (Fedora):
yum install python-matplotlib
yum install tkinter
yum install python-matplotlib-tk # for matplotlib.backends.backend_tkagg
yum install pyfits
yum install scipy
yum install pywcs deb: apt-get install python-matplotlib

For a deb based Linux (Ubuntu):
apt-get install python-matplotlib
apt-get install python-tk
apt-get install python-pyfits
apt-get install python-scipy
apt-get install python-pywcs

# Chapter 3

# Presentation

## 3.1 Modules

ABISM is called with the following command : "python Abism.py [image.fits]".
Initially, the GUI was written in a single class called MyWindow. The independent function were written in ImageFunction.py (this did not change). However the class, written in a single text module became larger and larger: it was difficult to maintain. The advantages of a class is that it shares its variables and function. Sharing all the function resulted not so useful: each module import the module it needs and the independent functions are written in lower modules which can be called by every upper module. Meanwhile, in one hand, the variables needed to be modified by the GUI and read by the function and in the other hand, the output of the mathematical functions needed to be displayed on the GUI.
The solution used to share the variables was to create two lowest modules: WorkVariable.py and GuiVariable.py. The modules of variables are empty. Every single module which aims to communicate with the whole software will import the modules of variables and modify their variables. An example of such a process is given in Table 3.1.

The dependencies are presented in a diagram in Table 3.1. Note that there is no plural in the name of the modules or functions.

- **Abism.py** just calls MyGui.py, it is to help the user to know how to call the program.
- **MyGui.py** is the most important part of the code, it is the main caller. There are some closed importation loops between for example MyGui.py and Pick.py, this should be cancelled in the future. The only "necessary" importation loop with MyGui.py is InitGui.py. Actually, in the futur, MyGui.py should be rename EventGui.py or CallerGui.py and the function presents should just import some other modules. In that way, ABISM will not need to import all the modules at the beginning. This will permit ABISM to open faster and to consume less RAM.
- **InitGui.py** creates the GUI and modify its on demand when a button is creating

```
“Gui.py”
import Var as V
import Function as F
from Tkinter import *
tk = Tk()
V.button = Button(tk,text=“Change color”,background=“red”,command=F.Command)
tk.mainloop()


“Function.py”
import Var as V
def Command():
    if V.button[“bg”]==“blue”: V.Button[“bg”=“green”]
    else : V.Button[“bg”=“blue”]


“Var.py”
#There is nothing written in this module
```

Table 3.1: Peace of code showing the way of sharing variables through the common importation of an empty module. This is a work around to avoid writing all the GUI in a single class.

a new frame (ex: “moreoption”).

- **Pick.py** defines the interaction between the user and the image (draw a rectangle...). It calls EventArtist.py modules created to hide the heavy matplotlib event handling classes.
- **Strehl** currently composed of 2 modules: a caller a a worker. These two modules are gathering informations to estimate the Strehl (with some conditions) but never work directly on the image array, that is why it is calling ImageFunction.py. It’s result is directly displayed in AnswerReturn.py.
- **ImageFunction.py** is the first module I created, it aims to hide some heavy calculation with lots of conditions. It may be decompose into several modules but why.
- **AnswerReturn.py** updates the GUI and the terminal to display the outputs.
- **HeaderRead.py** reads the header to give some useful parameters. It is called when an image is opened

- **FitFunction.py** performs a least square fit

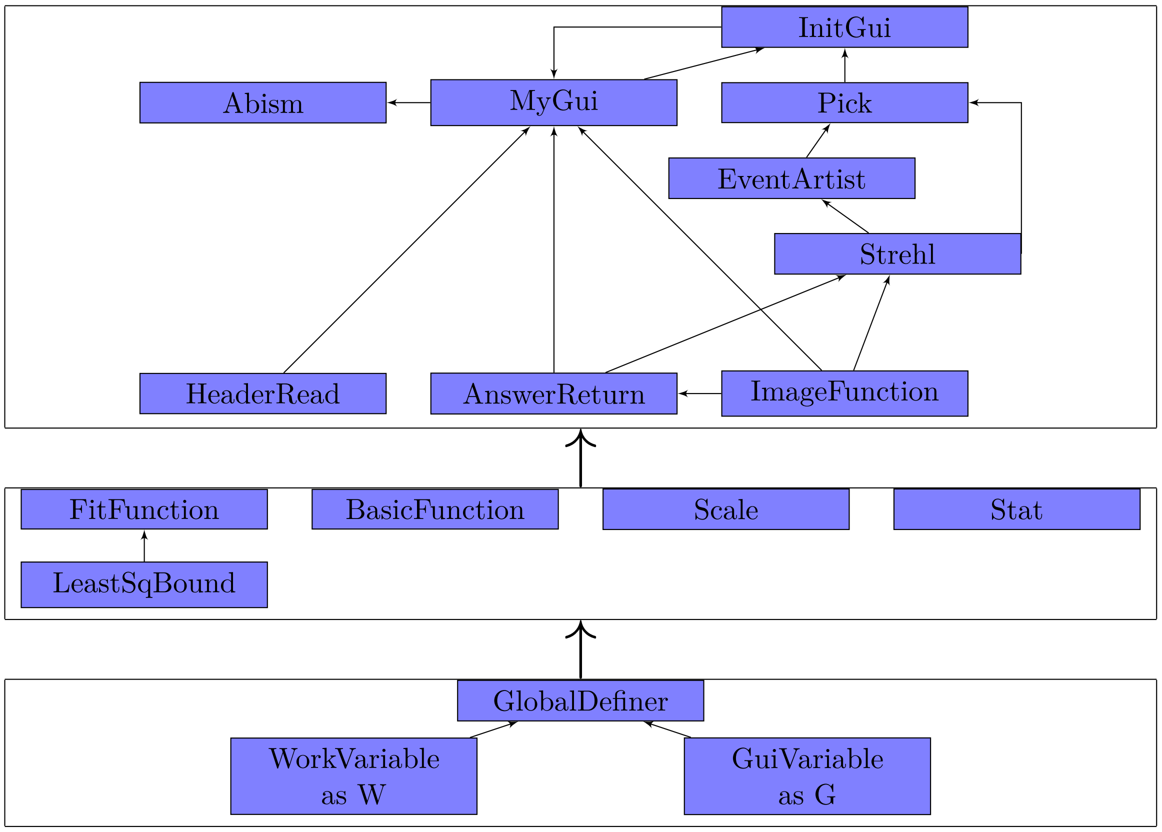


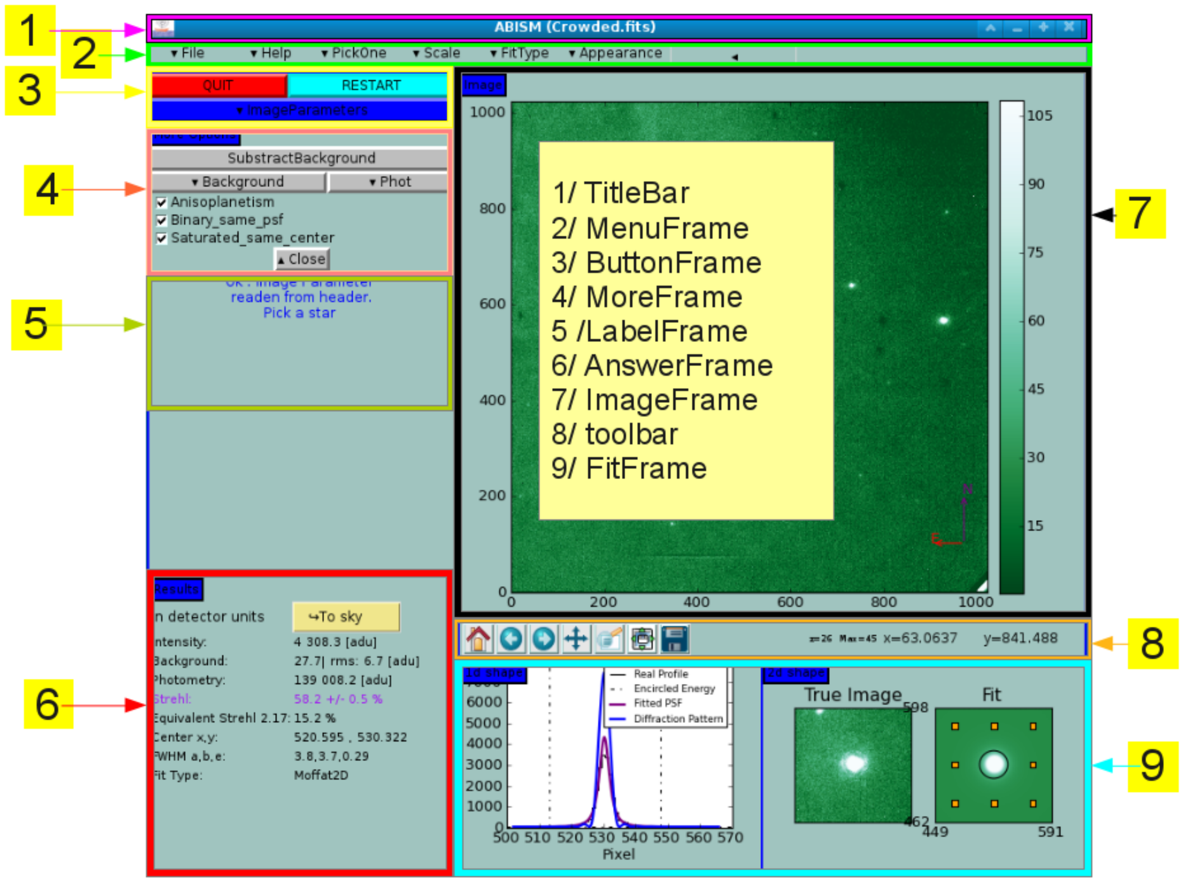
Table 3.2: Importation diagram of ABISM

- **BasicFunction.py** parametric function aimed to fit the date
- **Scale.py** To rescale the image (should be merged with Stat.py)
- **Stat.py** Some statistical functions on numpy array including a basic sky estimation.
- **GlobalDefiner.py** Define the global variables reads sys.argv (the terminal input line)

## 3.2 GUI

GUI stands for "Graphical User Interface". ABISM is written in tkinter[1], a python library. Tkinter library calls Tcl-Tk[2] which is currently present on all platforms (win-

[1]http://effbot.org/tkinterbook/tkinter-index.htm
[2]Tcl: Tool Command Language (http://www2.tcl.tk/445) is an interface, a common language created in Berkeley University.
Tk: Tool Kit (http://www2.tcl.tk/477) is an implementation of the scripted interface permitting GUI.



```
ok : Image Parameter
readen from header.
Pick a star
```

```
n detector units
ntensity:                    4 308.3 [adu]
Background:                  27.7| rms: 6.7 [adu]
Photometry:                  139 008.2 [adu]
Strehl:                      58.2 +/- 0.5 %
Equivalent Strehl 2.17:      15.2 %
Center x,y:                  520.595 , 530.322
FWHM a,b,e:                  3.8,3.7,0.29
Fit Type:                    Moffat2D
```

```
x=63.0637    y=841.488
```

Figure 3.1: Screen shot of ABISM in October 2013: version 1.00

dows, Mac, Python).
Let's describe ABISM GUI presents in Figure 3.2.

The GUI created by InitGui.py first packs the menu bar frame (MenuBar), then the left frame (TextPaned) and the right frame (ImagePaned)[3]. The elements presents are:

1. **Title bar** with the title and the icon.
2. **Menu bar** with some cascade buttons. [G.MenuBar is a frame]
   - **File** gives some info on the image.
   - **Help** helps.

[3]I differ between a Frame and a Paned Window because a child is packed in a Frame and append in a paned window, the paned window has sash so it can be resized.

- **Pick type** chooses the way of picking objects.
- **Scale** Rescale the image, change its color.
- **Fit type** chooses the parametric function to fit.
- **Appearance** change the background of ABISM (foreground may be implemented)

3. **Button frame** Some useful buttons, “ImageParameters” creates a frame with entry for the user to fill the image parameters if not present in the header. Pushing “entry“ or cliking again on ImageParameters save the parameters. The necessary parameters are [wavelength, diameter[4], obstruction[5], pixel scale].

4. **More Options frame** Some additional option for photometry and background estimation. This new frame is created by Menu> more option.

5. **Label frame** Presents what ABISM learned with the header.

6. **Answer frame** Displays the results, Strehl.

7. **Image frame** Shows the image.

8. **Toolbar frame** Gathers some toolkits, x,y,z value of the image under the cursor and z_max is the maximum value of z 10 pixels away from the pixel.

9. **Fit paned window** Some check plots to see if the fit is good. Also, these image canvas can be used to show other thing (histogram, statistics....).

## 3.3 Usefull info

### 3.3.1 Preferences

Some parameters ( see in GlobalDefiner.TerminalVar() ) can be stored when pushing the button reset. Then the resetting is calling ABISM with a certain command line displayed in the terminal. You have then two option, or you make an alias, utilising this command line for abism or you write GlobalDefiner.PreferenceDefined and put their your command line as a string variable called preference[“whatever”]. Also change in the same module, the default variable of the function Preference: the line ’def Preference(string=“whatever”)’ to call your preferences.

[4]in meter of primary mirror

[5]in area % of the secondary mirror on the primary

# Chapter 4

# Use

1.2.2

## 4.1 Quick example

You have a fits image in the file : "image_folder/image.fits" and ABISM modules are stored in abism_folder. The command:

```
python abism_folder/abism.py image_folder/image.fits
```

Will open ABISM GUI. If you cannot detect a star, the menu button "Scale" may help. Draw with the left mouse button a rectangle around the star et voilà !
Note that the fit will be performed in the aperture you've just drawn so try to include 1/2 of noise and 1/2 of signal because both are fitted. You want more info ? Press Help->PythonConsole and type "print W.strehl" and then press run. It is ugly, I defined a function called PD (Print Dictionary) so type "PD(W.strehl)" and then press run. The sum is the photometry before the background subtraction.

## 4.2 Photometry and background

### 4.2.1 Mesure background

After multiple tests, it was noticed that the background estimation is the main source of error, uncertainties. The background type variable is called W.type["back"]. For annulus and 8Rects background, the area where the sky is estimated is displayed in the 2d shape figure. If you go to File -> MoreOption -> Background, you have some different ways to estimate the background:

- **Annulus:** (default behaviour) is assessing the background in an elliptical annulus of the same shape of the photometric ellipse and between axes of 1.3 and 1.6 times the axes of the photometric ellipse. A 3-sigma kappa clipping is performed in the annulus area and the mean of the remaining pixels is the sky background estimation.

- **Fit:** It is assessing the background from the fit. It is though very dependant on the size of the bow you draw around your star: if the background is varies a lot, better draw a small box (if you checked this mode).
- **8Rects:** It is drawing 8 small rectangles around the star and take the average value of the photometry in these rectanges. This function was initially designed to detect a non-homogeneous background comparing the small rectangles between themselves.
- **Manual and None:** You can choose the background level, and None sets it to zero.

### 4.2.2 Wiew: set color, stretch and cuts

**Color:** The color map can be changed with a click on the view menu button, nmore color can be utilized with a cascade menu button ("More color") button.
The shortcuts up and down arrow key can loop throw the (matplotlib) color map list and left and right arrow keys invert the colors. The key shortcuts can only be used if the cursor is on Abism GUI.

**Stretch:** The stretch, scale, can only be called in the menu button view.

**Cut:** The cuts can be chosen with an algorithm in the view menu button, manually with the last menu button "Manual cut" of with the mouse on the color bar like for ds9 (Feature from DraggableColorbar.py). The algorithm are:

- percentage, the cut is performed to keep only a certain (99) percentage of the pixels, excluding the brightest AND the faintest.
- RMS, Scale from -1 to 5 sigma around the median.
- None, scale from the faintest to the brightest pixel.

**Contour:** Instead of giving colors, only draw the contour of the objects. Contours are drawn for 1 and 3 sigma higher than the median.

### 4.2.3 Measure photometry

The photometric type variable is called W.type["phot"], change it in File -> MoreOption -> Photometry.

- **Elliptical Aperture:** from the fit we determine the aperture to get 99% of the flux of the parametric function. A basic sum is performed in this area.
- **Fit:** The photometry is performed from the fit. The theoretical integral of the parametric function is computed (see Appendix).

- **Rectangle Aperture:** Like the elliptical aperture but with a projection of the major and minor axes on x and y axes to get a rectangular aperture.
- **Manual:** The photometry is performed in the box you've drawn. It is a basic sum of all the pixels presents in the picking rectangle.

## 4.3 Constrain the fit

If you draw a rectangle (pick one), the fit will be performed in this rectangle. If you choose binary fit, the fit will be performed in an aperture including both the binary stars with 5 times their FWHM. The parametric functions are Gaussian, Moffat and Bessel1. Also two Gaussians with the same center (Gaussian_hole) can mimic some saturated stars but it not recommended to use this fit. It is possible to change the parametric function in the Analysis menu button. In File -> More Option, one can also get some check buttons which aims to constrain some parameters of the fits:

- **Anisomorphisme:** Fit a PSF with 2 major axes if checked and a circular symmetric PSF if unchecked
- **Binary__same__psf:** Is only read out in case you are performing a binary fit. IS constrains the PSF

Each pixel are considered with the same uncertainties. The uncertainty is the root mean square of the image minus it median filter with a box of $3 \times 3$. Before the fit, a supposed PSF center is estimated as the maximum of the median filtered subimage selected (by the user drawn rectangle). This procedure permits to get read of the bad pixels and also it is more accurate than an iterative gravity center which may fall in a high noise level instead of on the true signal. Meanwhile, due to this estimation of the maximum, it is impossible to get the SR with negative PSF. After having guessed the center (and its intensity), the FWHM is quickly estimated by going down the PSF in the four cardinal directions. An average of the 4 values is considered as the first guessed FWHM. The supposed background is 0. The fit is performed with the python routine scipy.optimize.leastsq() with some boundaries :

## 4.4 Error

$$\begin{aligned} Sr &= \frac{I_{psf}}{I_{diff}} \\ dSr &= \frac{dI_{psf}}{I_{diff}} + I_{psf} \times \frac{dI_{diff}}{-I_{diff}^2} \\ &= \frac{1}{I_{diff}} \times (dI_{psf} + Sr\ dI_{diff}) \end{aligned}$$

We change the - in + and in each term we take the absolute value because errors can only be added (at least in our case [trust me]). The $dI_{psf}$ is given by the error in the best fit square. It is actually well constrained (1,2 %). The main source of error is $dI_{diff}$. As we said previously :

$$I_{diff} = bessel\_integer \times photometry \tag{4.1}$$

So we need to calculate the error on the photometry :

$$dPhot = dBack \times \sqrt{N} \tag{4.2}$$

Where dBack is the error on the background, the RMS and N is the number count, the number of pixels in the aperture where the photometry were performed.

# Chapter 5

# Develop ABISM

## 5.1 Add fit type

ABISM is most of the time fitting the PSF, you may want to had more parametric functions. The fit type is stored in W.fit_type variable and mostly read by StrehlImage.py module.

- ABIMS has:
  - Gaussian
  - Moffat
  - Bessel
  - Gaussian_hole (2 gaussians with same center for saturated stars
- ABISM misses:
  - Cut fit with a threshold (for saturation)
  - Bessel with central obstruction (same as the telescope)

The fit is performed by FitFunction.py calling leastsqbound.py[1]. If you want to add a fit, you need to:

[1]The first written by Antoine Mérand aims to call easier the python function scipy.optimize.leastsq. Some examples of fitting procedure are given at the end of FitFunction.py.
The second written by F. James and M. Winkler aims to bound the fit returning a junk function when the parameters are out of the bound.

1. Add a menu button in the GUI in InitGui.py in the function FitMenu: just add in the list of fit types, a small list with two strings. The first string is the displayed value in the GUI (“My Fit”) and the second is value taken by W.type[“fit”] variable when the user is pressing your new menu button (“my_fit_type”). It is recommended to use the same string value my_fit_type as the parametric function you will define in BasicFunction.py module. In done so, you can call this function with vars(BF)[W.type[“fit”]]. Otherwise, you will have to add condition in AnswerReturn.py module which needs to call the parametric function in order to display it.

2. Add a condition in StrehlImage.py: if W.type[“fit”]==“my_fit_type”. You then need to determine the supposed parameters and you can constrain the parameters with some boundaries. so call:

   FitFunction.leastsqFit(BasicFunction.Moffat2pt,points,param,
   image,err=err_image,doNotFit=doNotFit,bounds=James)

   Where:

   - **image** is a 2d array of the intensity in the region you want to fit
   - **err_image** is a 2d array of the error in the region you want to fit
   - **doNotFit** is a list of string of the parameters you don’t want to fit, because you know they have the good value. For example if you want to fit a binary star, you can fit the first one, assume it is good, and then fit the second one, then assume the second is good, fit the first, iteratively instead of subtracting the fit to the image (it is mathematically equivalent but cleaner).
   - **James** is a dictionary with key some parameters and value a list of 2 values : the lower bound and the upper bound (in this order).

   You need also to determine the aperture of the ellipse in ImageFunction.py module in EncircledEnergy. That is why it is recommended to use the same strings for the parameters:

   - center_x, center_y , spread_x, background, intensity
   - spread_y , theta
   - exponent

3. Add a parametric function in BasicFunction.py off the form:

   def ParametricFunc(points,param) : return grid

   Where points[0] and points[1] are arrays with the same shape as the image and containing the index of the rows and columns respectively (created with np.meshgrid); param is a dictionary countaining the parameters of the parametric function and their value.

## 5.2 Add pick type (matplotlib interaction)

The matplotlib (image) interaction type is stored in W.type[“pick”] and mostly read by Pick.py module. Note that AnswerReturn module also reads it to anticipate which parameters it will know.

• ABIMS has:

- Pick One (draw a rectangle)
- Pick Many (draw Many rectangle)
- Stat (draw a rectangle)
- Binary (Make 2 clicks)
- Profile (draw a line)
- Ellipse (an ellipse canvas is following the cursor using a blit mode to go faster )
- Annulus (an annulus canvas is following the cursor with blit)
- NoPick (To disable the matplotlib connections connection)

• ABISM misses:

- Custom selection (enable the creation of a custom polygon to get a customize photometry for a very asymmetric PSF)
- Other event happening with some current selection method, for example pick many objects and store them.

Note that the current conections event are really enougth for most of the work in astronomy. One may better, change the way of working with the selected area rather than changing way connections methods.

1. Add a GUI menu button including a string representing the label displayed on the menu button (“My Pick”) in the list in ConnectMenu function in InitGui.py module.
2. Add an element in RefreshPick list. Every pick button in the pick type menu button is calling RefreshPisk function in Pick.py module with its label (the name displayed on the button) as argument. In the list, add a sublist with: 1/The Label displayed on the menu button (“My Pick”); 2/ The value W.pick_type will take (“my_pick”); 3/ The Function to call (“MyPick”)

3. Add a function in Pick.py module which will be called when the user click on your menu button. This function must include a matplotlib connection when the function is called with no argument and a disconnection when the function is called as MyPick(disconnect==True). See the other pick functions or http://matplotlib.org/users/event_handling.html for more information on matplotlib event connection. If the connection function is complicated, please store it as a class in EventArtist.py as the function Profile().

## 5.3 Add instrument

The instruments classes are stored in ReadHeader.py. All these classes inherit from the Header class. The Header class contains the ___init___ function which is automatically launch when the class is called: ___init___ is the python name for the constructor function. The function ___init___ only call few functions which aims to store header informations on class variables. W.head.header is the full header dictionary and the header class is called W.head. For example the wavelength can be called in any module as W.head.wavelength.
The instrument is read out by the function CallHeaderClass in ReadHeader.py module as 'INSTRUM', 'INSTRUME' or 'INSTRUMENT' key.
So to add an instrument:

- Make a condition in the CallHeaderClass function (basically to call your class when you want): "if 'my_instru_string' in instru : W.head = MyInstru(header)" where my_instru_string is string contained in the INSTRUMENT keyword and MyInstu() is the class you will generate; finally, instru, W.head and header are variables I defined before, just call its like that.

- Create a class inheriting from Header class :"class MyInstru(Header)" and define the variables you want in its. There are several functions called by ___init___ (i. e., when calling your class). Note that you if a function is missing in your class, the function with the same name will be called in its parent class: Header. Therefore, you don't need to create all of these functions if you don't want to modify the the header reading default of the corresponding keys. Let's enumerate the called functions, if its are marked with a "*", this means that you should create a function with such a name to create your instrument specific variables.

  1. **InitKey:** To create the defaults keys, in case some keys are not defined [you should not create such a function and rely on the inherited function I created ]
  2. **StrehlKey*:** To define the necessary variables to determine a Strehl
     - self.wavelength
     - self.diameter
     - self.obstruction

  - self.pixel_scale

  If you create a function called StrehlKey, make sure you defined all these variables and that they are floats.

3. **ObservationKey*:** To know some conditions of observation, telescope, inst, ESO. It is a good check to know what your header reading performed.
4. **WCSKey:** To know the WCS projection [ if you're using reduced image, I advice you create such a function ]
5. **MoreKey:** To put

# Appendix A

# Ellipse drawing

## A.1 Rotate referential, coordinates behave the opposite way as vectors

An intuitive example is scaling : if you increase the size of the base vector, for the same point, you need to decrease the coordinate.

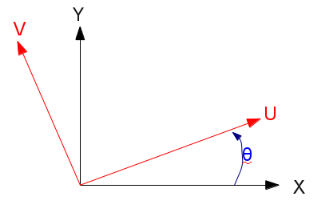
We consider a orthonormal referential with base vector $(\vec{U}, \vec{V})$ rotated by $\theta$ from the base $(\vec{X}, \vec{Y})$ like Figure A.1.

We define the rotational matrix:
$\begin{pmatrix} \vec{U} \\ \vec{V} \end{pmatrix} = \begin{bmatrix} Cos(\theta) & Sin(\theta) \\ -Sin(\theta) & Cos(\theta) \end{bmatrix} \begin{pmatrix} \vec{X} \\ \vec{Y} \end{pmatrix}$

But coordinates behave in the opposite way:
$\begin{pmatrix} u & v \end{pmatrix} = \begin{pmatrix} x & y \end{pmatrix} \begin{bmatrix} Cos(\theta) & Sin(\theta) \\ -Sin(\theta) & Cos(\theta) \end{bmatrix}$

That can be written as :
$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{bmatrix} Cos(\theta) & -Sin(\theta) \\ Sin(\theta) & Cos(\theta) \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$

Figure A.1: Rotation of a 2D base of vectors.



Why ?
If:

$$\vec{Vec} = (x, y) \begin{pmatrix} \vec{X} \\ \vec{Y} \end{pmatrix} = (u, v) \begin{pmatrix} \vec{U} \\ \vec{V} \end{pmatrix} \tag{A.1}$$

And:

$$\begin{pmatrix} \vec{U} \\ \vec{V} \end{pmatrix} = M \begin{pmatrix} \vec{X} \\ \vec{Y} \end{pmatrix} \tag{A.2}$$

So:

$$(u,v) \times M \begin{pmatrix} \vec{X} \\ \vec{Y} \end{pmatrix} = (x,y) \times \begin{pmatrix} \vec{X} \\ \vec{Y} \end{pmatrix} \tag{A.3}$$

And by identification $\boxed{(u,v) = (x,y)M}$ or $\begin{pmatrix} u \\ v \end{pmatrix} = M^{-1} \begin{pmatrix} x \\ y \end{pmatrix}$

## A.2 Ellipse equation

We have an ellipse of axes (ru,rv) along $(\vec{U},\vec{V})$ vectors and centred on (0,0).

$$\begin{aligned} & \left(\tfrac{u}{ru}\right)^2 + \left(\tfrac{v}{rv}\right)^2 \\ = & \left(\tfrac{xCos(\theta)-ySin(\theta)}{ru}\right)^2 + \left(\tfrac{xSin(\theta)+yCos(\theta)}{rv}\right)^2 \\ = & \boxed{\begin{array}{c} x^2\left(\left(\tfrac{Cos(\theta)}{ru}\right)^2 + \left(\tfrac{Sin(\theta)}{rv}\right)^2\right) \\ +y^2\left(\left(\tfrac{Sin(\theta)}{ru}\right)^2 + \left(\tfrac{Cos(\theta)}{rv}\right)^2\right) \\ +x \times y\left(Sin(2\theta)(1/rv^2 - 1/ru^2)\right) \end{array}} \end{aligned} \tag{A.4}$$

# Appendix B

# Encircled

## B.1 The PSFs

We forget the two useless parameters x0 and y0 and consider the PSF to be centred on 0.

### B.1.1 Gaussian

$$PSF(I, \alpha) = I \times e^{(R/\alpha)^2}$$

### B.1.2 Moffat

$$PSF(I, \alpha, b) = I \times \left(1 + \frac{R^2}{\alpha^2}\right)^{-b}$$

### B.1.3 Encircled Energy

We integer a typical PSF up to radius R. How much flux do we have. We also add, I and $\alpha$, some standard scale parameters such as $PSF_{scaled} = I * PSF(R/\alpha)$. $\alpha$ has units of pixels and I of flux.
Now if we want $\epsilon$% of the flux, up to which R ($R/\alpha$), do we need to integrate the PSF.

### B.1.4 Changing Variable

In the next section we will calculate the integral of $PSF(I, \alpha ...)$ wich can be related to the integral of $PSF(I, 1...)$ (considering $\alpha = 1$ is easier).

$$
\begin{aligned}
PSF(I,\alpha ...) &= 2\pi \times \int f\left(\frac{r}{\alpha}\right) rdr \\
&= 2\pi \times \int f\left(u\right)(u \times \alpha)(du \times \alpha) \\
&= \alpha^2 \times 2\pi \times \int f\left(u\right) udu \\
&= \alpha^2 \times PSF(I, 1...)
\end{aligned}
$$

WARNING, this is an obvious situation, but you have 2 things to remember :

- Multiply by $\alpha^2$ (as described above)
- Integrate up to $R/\alpha$ (and not up to R any more because this R you calculated before was calculated for a $\alpha = 1$, So to be at the "same" place on the PSF, scale also the cuts)

We utilised

$$
\begin{aligned}
u &= r/\alpha \\
du &= dr/\alpha
\end{aligned}
$$

## B.2 Gaussian

### B.2.1 FWHM

$$
\begin{aligned}
e^{-\frac{FWHM^2}{\alpha^2}} &= \frac{1}{2} \\
\frac{FWHM^2}{\alpha^2} &= ln(2) \\
FWHM &= \sqrt{ln(2)} \times \alpha
\end{aligned}
$$

### B.2.2 Encircled Energy

$$
\begin{aligned}
2\pi \int_0^R e^{-r^2} r\ dr &= 2\pi [\frac{e^{-r^2}}{-2}]_0^R \\
&= \pi(1 - e^{-R^2}) \\
\implies 2\pi \int_0^R e^{-(r/\alpha)^2} r\ dr &= I \times \alpha^2 \times \pi(1 - e^{-R^2/\alpha^2}) \\
&\lim_{R \to +\infty} I\alpha^2\pi
\end{aligned}
$$

Note that we integrate up to R and not up to $R/\alpha$. The factor $\alpha^2$, comes from a change of units and the term $\alpha$ in the exponential comes from the fact that the PSF is different.

The percentage $\epsilon$ can be calculated : $A = \epsilon \times Total\ energy$, considering I=$\alpha$=1, (I has no influence, R is prop to $\alpha$).

$$\begin{aligned}
\pi(1 - e^{-R^2}) &= A \\
e^{-R^2} &= 1 - \frac{A}{\pi} \\
R &= \sqrt{-ln(1 - \frac{A}{\pi})} \\
R &= \sqrt{-ln\left(1 - \frac{\epsilon\pi}{\pi}\right)} \\
\implies R &= \alpha\sqrt{-ln(1-\epsilon)}
\end{aligned}$$

### B.2.3 2d Integral

$$\begin{aligned}
& & 2\pi \times \int_0^\infty e^{-r^2} r dr \\
&= & 2\pi \times \left[\frac{e^{-r^2}}{-2}\right]_0^\infty \\
&= & 2\pi \times \left[0 - \frac{1}{-2}\right] \\
&= & \pi \\
& & \boxed{\int_{2d} I \times e^{-x^2/\alpha^2} = I\alpha^2\pi}
\end{aligned}$$

## B.3 Moffat

### B.3.1 FWHM

$$\begin{aligned}
\left(1 + \frac{FWHM^2}{\alpha^2}\right)^{-b} &= \frac{1}{2} \\
\frac{FWHM^2}{\alpha^2} &= \left(\frac{1}{2}\right)^{-1/b} - 1 \\
FWHM &= \alpha \times \sqrt{\left(\frac{1}{2}\right)^{-1/b} - 1}
\end{aligned}$$

### B.3.2 Encircled Energy

$$\begin{aligned}
2\pi \times \int_0^R \left(1+r^2\right)^{-b} rdr &= 2\pi \times \left[\frac{(1+r^2)^{-b+1}}{2(-b+1)}\right]_0^R \\
&= \frac{\pi}{b-1}\left(1-(1+R^2)^{-b+1}\right) \\
\implies 2\pi \times \int_0^R I\left(1+(r/\alpha)^2\right)^{-b} rdr &= \frac{I\alpha^2\pi}{b-1}\left(1-(1+(R/\alpha)^2)^{-b+1}\right) \\
& \lim_{R\to+\infty}\frac{I\alpha^2\pi}{b-1}
\end{aligned}$$

The percentage ($A = \frac{pi}{b-1} \times \epsilon$: $\alpha$ and I wound vanish on the first line) of encircled energy over the total energy :

$$\begin{aligned}
\frac{\pi}{b-1}\left(1-(1+R^2)^{-b+1}\right) &= A \\
\left(1-(1+R^2)^{-b+1}\right) &= A\frac{b-1}{\pi} \\
(1+R^2) &= \left(1-A\frac{b-1}{\pi}\right)^{1/(1-b)} \\
R &= \sqrt{\left(1-A\frac{b-1}{\pi}\right)^{1/(1-b)}-1} \\
\implies R &= \alpha\sqrt{(1-\epsilon)^{1/(1-b)}-1}
\end{aligned}$$

### B.3.3 2d integral

$$\begin{aligned}
& 2\pi \times \int_0^\infty \left(1+r^2\right)^{-b} rdr \\
= \quad & 2\pi \times \left[\frac{(1+r^2)^{-b+1}}{2(-b+1)}\right]_0^\infty \\
= \quad & 2\pi \times \left[0-\frac{1}{2(-b+1)}\right] \\
= \quad & \frac{\pi}{b-1} \\
& \boxed{\int_{2d} I \times [1+x^2/\alpha^2]^{-b} = \frac{\pi \times I\alpha^2}{b-1}}
\end{aligned}$$

## B.4 Bessel

### B.4.1 FWHM

$$FHWM = 1.028\lambda/D \tag{B.1}$$

### B.4.2 2d integral

$$\begin{aligned} & \int_{2d} \left(\frac{2J_1(x)}{x}\right)^2 \\ = & \quad 2\pi \int_0^\infty 4\frac{J_1(r)^2}{r^2} r \times dr \\ = & \quad 8\pi/2 = 4\pi \end{aligned}$$

Because :

$$\int_0^i nfty \frac{J_\nu(t)}{dt} = \frac{1}{2\nu} \tag{B.2}$$

## B.5 Bessel with obstruction ($\epsilon$)

2d integral
The fourier transform of an circular aperture with a pourcentage of central obstruction $\epsilon$ is :

$$I(u) = \frac{1}{(1-\epsilon^2)^2}\left[\frac{2J_1(u)}{u} - \epsilon^2 \frac{2J_1(\epsilon u)}{\epsilon u}\right]^2 \tag{B.3}$$

With $u = \frac{\pi D \theta}{\lambda}$

$$\begin{aligned} I(u) = & \quad \frac{4}{(1-\epsilon^2)^2}\left[\left(\frac{J_1(u)}{u}\right)^2 - 2\epsilon^2\left(\frac{J_1(\epsilon u)\times J_1(u)}{\epsilon u^2}\right) + \epsilon^4\left(\frac{J_1(\epsilon u)}{\epsilon u}\right)^2\right] \\ \int_{2d} I(u) = & \quad \frac{4}{(1-\epsilon^2)^2}\left[\pi - 2\epsilon \times (\pi\epsilon) + \epsilon^4 \times \frac{\pi}{\epsilon^2}\right] \\ \int_{2d} I(u) = & \quad \frac{4\pi}{(1-\epsilon^2)^2}\left[1-\epsilon^2\right] \\ \int_{2d} I(u) = & \quad \frac{4\pi}{1-\epsilon^2} \end{aligned}$$

We could have found it noticing that with a central apertur, the flux will decrease like $1-\epsilon^2$ but the central intensity will decrease like $(1-\epsilon^2)^2$. So :

$$\frac{I_{max}}{\int I(u)du} = \frac{1-\epsilon^2}{4\pi} \tag{B.4}$$

Now we have to change variable u to be able to integrate over the pixels. We can do that in 2 steps : from u to $\theta$ and from $\theta$ to pixels. We have :

$$\begin{aligned} \frac{\theta}{u} = & \quad \frac{\lambda}{\pi D} \\ \frac{Pixel}{\theta} = & \quad 1/FPS \\ \frac{Pixel}{u} = & \quad \frac{\lambda}{\pi D \times FPS} \end{aligned}$$

| | 99.9% | 99.5% | 99% | 95% | 90% | 50% |
|---|---|---|---|---|---|---|
| Gaussian | 2.63 | 2.30 | 2.15 | 1.73 | 1.52 | 0.83 |
| Moffat (b=1.5) | 999 | 199 | 100 | 20 | 10 | 1.75 |
| Moffat (b=2) | 31 | 14. | 10 | 4.35 | 3. | 1. |

Where FPS = Focal Plane Scale and is proportional to the inverse of the telescope focal lenght. And then :

$$\begin{aligned}
\int_{2d} I(u) dpixel = & \left(\frac{\lambda}{FPS\pi D}\right)^2 \int_{2d} I(u) du \\
\int_{2d} I(u) dpixel = & \left(\frac{\lambda}{FPS\pi D}\right)^2 \times \frac{4\pi}{1-\epsilon^2} \times I_{max}
\end{aligned}$$

We know the integral, we will find $I_{max}$ to calculate the strehl.

## B.6 Numerical Results

So, if you want percent % of the flux, how far must you go ? The results are given in units of $\alpha$, link $\alpha$ to the FWHM with Section **??**.

bad for moffat