astwro Documentation

Release 0.7.3

Mikolaj Kaluszynski

Sep 12, 2018

Contents:

1 Installation **3**
- 1.1 Installation through PyPI . . . 3
- 1.2 Dependencies . . . 3
- 1.3 github Installation . . . 3

2 astwro.pydaophot **5**
- 2.1 Configuration . . . 5
- 2.2 Files and directories . . . 6
- 2.3 Operation modes - batch and parallel execution . . . 8
- 2.4 Setting image and options . . . 9
- 2.5 Logging . . . 9

3 gapick - Finding optimal set of PSF stars **11**
- 3.1 Overview . . . 11
- 3.2 Parameters . . . 11

4 astwro.tools command line tools **15**

5 Deriving a Point-Spread Function in a Crowded Field **17**
- 5.1 following Appendix III of Peter Stetson's *User's Manual for DAOPHOT II* . . . 17
- 5.2 Using `pydaophot` form `astwro` python package . . . 17

6 *(1) Run FIND on your frame* **19**

7 *(2) Run PHOTOMETRY on your frame* **21**

8 *(3) SORT the output from PHOTOMETRY* **23**

9 *(4) PICK to generate a set of likely PSF stars* **25**

10 *(5) Run PSF* **27**

11 *(6) Run GROUP and NSTAR or ALLSTAR on your NEI file* **29**

12 *(8) EXIT from DAOPHOT and send this new picture to the image display* **31**

13 *(9) Back in DAOPHOT II ATTACH the original picture and run SUBSTAR* **33**

14 (10) ATTACH the new star subtracted frame and repeat step (5) to derive a new point spread function 35

15 (11+. . .) Run GROUP NSTAR or ALLSTAR 37

16 API Reference 39
16.1 API Reference 39

17 Indices and tables 57

18 Contact 59

Python Module Index 61

astwro is the set of modules developed in Astronomical Institute of Wroclaw University.

It contains wrappers for *daophot* package, star lists (as *pandas* DataFrames) manipulation routines with export/import to *daophot* and *ds9* formats, genetic algorithm for the search for optimal PSF stars and some other stuff.

CHAPTER 1

Installation

1.1 Installation through PyPI

Use standard pip installation:

```
$ pip install astwro
```

This also installs *astwro.tools* command line scripts.

1.2 Dependencies

Developed of *astwro* have been switched to Python 3. Most of the code still works with Python 2, but support for this version will be dropped.

The different submodules have different requirements, the common requirements are:

- *pandas*
- *astropy*
- *scipy*

pydaophot module, and tools that use it, requires the installation of modern Peter B. Stetson's *DAOPHOT* package. However, there is no guarantee that yours version will work with *pydaophot*.

The optimization of PSF stars set using genetic algorithm (*astwro.tools.gapick.py* tool) uses *deap* GA package and *bitarray*.

1.3 github Installation

One can also install unreleased version from github

CHAPTER 2

astwro.pydaophot

The `astwro.pydaophot` module provides an interface to the command line tools of Peter B. Stetson *daophot* and *allstar*

2.1 Configuration

2.1.1 *pydaophot.cnf* configuration file

The `astwro.pydaophot` Module uses configuration file *pydaophot.cfg*.

On import, pydaophot is looking for *pydaophot.cfg* in the following directories:

```
/etc/pydaophot/
~/.config/pydaophot/
./
```

and reads found files in that order, overwriting repeat parameters.

The default configuration file is included in the module: *[astwro path]/pydaophot/config/pydaophot.cfg* and can be used as template for creating user's own ones.

Default configuration – default *pydaophot.cfg* file content:

```
# Patches (optional) and names of executables
[executables]
daophot = sdaophot
allstar = sallstar

# Location of standard config files
[files]
# daophot.opt =
# allstar.opt =
# photo.opt =
```

2.1.2 Daphot/Allstar *opt*-configuration files

The module provides various options to indicate the location of following *daophot* configuration files:

```
daophot.opt
allstar.opt
photo.opt
```

Routines searches the following locations in the order provided:

- parameter – constructors of `Daophot` and `Allstar` objects and some routines, have parameters to indicate the location of *opt*-files (e.g. the *daophotopt* parameter of the `Daophot` constructor). Also script's command line parameters (e.g. `gapick --photo-opt`) are passed as arguments to appropriate routines.
- working directory – if working directory of script process (not to be confused with `Daophot` object's working directory - *runner directory*!) contains *opt* file, this file will be used.
- *pydaophot.cfg* – the module configuration file contains section *[files]* where location of the *opt* files can be specified.
- default files – if module cannot locate *opt* file in locations below, uses the default file located in *[astwro path]/pydaophot/config*.

Note, that the presented order of searching means, that e.g. the working directory *daophot.opt* file have priority over another one provided in *pydaophot.cfg*.

2.2 Files and directories

2.2.1 Paths

To distinguish between the working directory of the python program and the working directory of the underlying *daophot* process, the following naming convention is used:

- *runner directory* is the working directory of the underlying *daophot* process. This directory accessible by the `Daophot[Allstar].dir.path` property.
- *working directory* is current working directory of python program as obtained by `os.getcwd()`.

2.2.2 Runner directory

Each `Daophot`[1] object maintains it's own *runner directory*. If directory is not specified in constructor, the temporary directory is created.

The *runner directory* is accessible by the `Daophot[Allstar].dir.path` property.

`Daophot`'s *runner directory* is the working directory of *daophot* program.

2.2.3 Specifying the file patch

For all command methods (`FInd()`, `PHotometry()`, . . .) parameters that refer to files follow the rules described below. Understanding those rules is especially important to distinguish whether the file in *runner directory* or another directory is addressed.

1. All filenames without *path* prefix, addresses the *runner directory* files.

[1] All information below applies to `Allstar` as well

2. Files with absolute path prefix (that is, starting with /), are… absolute addressed files as expected.
3. Files with relative (but not empty) path prefix, are relative to *working directory*.
4. Files with patch prefix starting with ~ (tilde) are relative to the user's home directory.

In other words, file pathnames are fairly standard and reltive to script *working directory*, with exception than lack of path prefix indicates file in *runner directory*.

During operation, all files has representation in *runner directory*, and underlying *daophot* processes only works on the files in that directory. It's implemented by creating symbolic link in *runner directory* for the input files and copying the output files from *runner directory* into destination directories if such external output file is requested.

2.2.4 Runner directory file names

To avoid filename conflicts, the name of link/file in the *runner directory* created for external file consists of:

- hash of absolute pathname and
- original filename.

2.2.5 Input files

Due to the limited length of directory paths maintained by the *daophot* program, for all filepaths provided to `Daophot` object, the symbolic link is created in *runner directory*, and this link is given to the *daophot* process instead of the original filename. Existing symbolic links of the same name are overwritten (because name is generated from absolute patch it's not a problem at all).

2.2.6 Output files

For output files, when the filename contains path component, *daophot* is instructed to output into the *runner directory* file, then, after *daophot* terminates, this file(s) are copied to the path specified by the user.

Warning: The output files, existing in *runner directory*, are deleted on queuing command. This can lead to unexpected behaviour in `"batch"` mode, when mixing input/output files. Consider following example:

```
d.mode = 'batch'
d.GRoup(psf_file='i.psf')  # preexisting i.psf (input)
d.PSf(psf_file='i.psf')    # deletes i.psf  (output)
d.run()                    # GROUP will miss i.psf and fail
```

Note: In the batch mode, the copying occurs after execution of all commands in queue. This can have consequences when using the external file as an output of one command and input of further one. Usually everything should be fine, since the filenames generated for *runner directory* are deterministic as described above.

In the following example

```
from astwro.pydaophot import Daophot
from astwro.sampledata import fits_image

d = Daophot(image=fits_image())
```

(continues on next page)

(continued from previous page)

```
d.mode = 'batch'
d.FInd(starlist_file='~/my.coo')
d.PHotometry(stars_file='~/my.coo')
d.run()
```

`FInd()` command instruct daophot to output into file *1b7afb3.my.coo* in *runner directory*. `PHotometry()` command will read file *1b7afb3.my.coo* from *runner directory*. After all *1b7afb3.my.coo* will be copied to *~/my.coo*. Sometimes it's easier to work explicitly on the files inside the *runner directory* :

```
from astwro.pydaophot import Daophot
from astwro.sampledata import fits_image

d = Daophot(image=fits_image(), batch=True)
d.FInd()         # equiv: d.FInd(starlist_file='i.coo')
d.PHotometry()   # equiv: d.PHotometry(starlist_file='i.coo')
d.run()
d.copy_from_runner_dir('i.coo', '~/my.coo')
```

User can also get patch to this file without copying

```
d.file_from_runner_dir('i.coo')
```

or, without specifying names at all

```
d.FInd_result.starlist_file
```

2.3 Operation modes - batch and parallel execution

The execution regime of *daophot* commands depends on `Daophot`'s operation mode (this applies to any runner subclassing the `Runner` class).

2.3.1 Operation modes

Property `Daophot.mode` (type: *str*) indicates operation mode:

- `"normal"` (default) - Every command method (`FInd()`, `PHotometry()`, ...) blocks until the underlying *daophot* process completes processing. That is intuitive behaviour. Every command is executed by brand new *daophot* process, which terminates once the command execution is finished.

  The `ATtach()` and `OPtions()` commands are not available in `"normal"` mode. Instead use `set_image()` and `set_options()` methods that enqueue the appropriate *daophot* commands for execution before any other command.

- `"bath"` - The command methods does not trigger the underlying *daophot* process. Instead, commands are stored in the internal commands queue and are send to *daophot* for execution together on explicitly called `run()` method. All commands are executed one by one in a single *daophot* process, which terminates after completion of the last command.

2.3.2 Asynchronous execution

The `"bath"` operation mode allows asynchronous execution by passing `wait=False` to the `run(wait=False)` method. In that case, the `run()` method returns immediately after passing the commands to the underlying *daophot* process. Further execution of the Python program runs in parallel to the *daophot* process.

The user can check if *daophot* is still processing commands by testing the `Daophot.running` property.

2.4 Setting image and options

The *daophot options and the attached image are the parameters that persist in ‘daophot* session. In `"normal"` mode each command is executed in a separate *daophot* process which terminates after execution of the command, so the configuration options and the attached image must be set before each command execution.

The `ATtach()` and `OPtions()` methods enqueues *ATTACH* and *OPTION* commands like any other command methods and are useless in *“normal”*. The `set_image()` and `set_options()` methods should be used instead, which enqueue the appropriate *daophot* commands for execution before every command.

2.5 Logging

The `astwro.pydaophot` uses the logger (from the `logging` module) named `"pydaophot"` and it’s child loggers.

CHAPTER 3

gapick - Finding optimal set of PSF stars

3.1 Overview

gapick - [g]enetic [a]lgorithm PICK (after daophot PICK), is command line and python module for finding PSF stars set which minimizes goal function: mean of *allstar* errors for all good stars in field. The "good stars" are those which passes filtering against magnitude (brighter than minimal threshold `--max-ph-mag`) and against aperture photometry error (error lower than threshold `--max-ph-err`).

The mineralization process is implemented as genetic algorithm, where individual is some subset of initial PSF stars candidates.

gapick command line tool is automatically installed with *pip* installation of *astwro* package. Python function interface is available as follows:

```
from astwro.tools import gapick
gapick.main(arguments)
```

Arguments of `astwro.tools.gapick.main()` corresponds to long version of command line arguments, eg command line:

```
$ gapick gapick --overwite --out_dir results i.fits
```

is equivalent to:

```
from astwro.tools import gapick
gapick.main(overwrite=True, out_dir='results', image='i.fits')
```

`image` is only positional argument of commandline.

3.2 Parameters

Commandline parameters help is displayed with `--help` option:

```
$ gapick --help
usage: gapick [-h] [--all-stars-file FILE] [--psf-stars-file FILE]
              [--frames-av n] [--frames-sum n] [--photo-opt FILE]
              [--photo-is r] [--photo-os r] [--photo-ap r [r ...]]
              [--stars-to-pick n] [--faintest-to-pick MAG] [--fine]
              [--max-psf-err-mult x] [--max-ph-err x] [--max-ph-mag m]
              [--parallel n] [--out_dir output_dir] [--overwrite]
              [--ga_init_prob x] [--ga_max_iter n] [--ga_pop n]
              [--ga_cross_prob x] [--ga_mut_prob x] [--ga_mut_str x]
              [--loglevel level] [--no_stdout] [--no_progress] [--version]
              [image]

Find best PSF stars using GA to minimize mean error 2nd version of PSF fit
applied to all stars by allstar. Minimized function is the mean of allstar's
chi value calculated on sigma-clipped (sigma=4.0) list of all stars. Results
will be stored in --dir directory if provided. List of stars will be output to
stdout until suppressed by -no_stdout

positional arguments:
  image                 FITS image file (default: astwro sample image for
                        tests)

optional arguments:
  -h, --help            show this help message and exit
  --all-stars-file FILE, -c FILE
                        all stars input file in one of daophot's formats
                        (default: obtained by daophot FIND)
  --psf-stars-file FILE, -l FILE
                        PSF candidates input file in one of daophot's formats,
                        the result of algorithm is a subset of those stars
                        (default: obtained by daophot PICK)
  --frames-av n         frames ave - parameter of daophot FIND when --all-
                        stars-file not provided (default: 1)
  --frames-sum n        frames summed - parameter of daophot FIND when --all-
                        stars-file not provided (default: 1)
  --photo-opt FILE, -O FILE
                        photo.opt file for aperture photometry (default: none)
  --photo-is r          PHOTOMETRY inner sky radius, overwrites photo.opt,
                        (default: from --photo-opt or 35)
  --photo-os r          PHOTOMETRY outher sky radius, overwrites photo.opt,
                        (default: from --photo-opt or 50)
  --photo-ap r [r ...]  PHOTOMETRY apertures radius (up to 12), overwrites
                        photo.opt, (default: from --photo-opt or 8)
  --stars-to-pick n, -P n
                        number of stars to PICK as candidates when --stars-to-
                        pick not provided (default: 100)
  --faintest-to-pick MAG
                        faintest magnitude to PICK as candidates when --stars-
                        to-pick not provided (default: 20)
  --fine, -f            fine tuned PSF calculation (3 iter) for crowded
                        fields, without this option no neighbourssubtraction
                        will be performed
  --max-psf-err-mult x  threshold for PSF errors of candidates - multipler of
                        average error; candidates with PSF error greater than
                        x*av_err will be rejected (default 3.0)
  --max-ph-err x        threshold for photometry error of stars for processing
                        by allstar; stars for which aperture photometry
```

(continues on next page)

(continued from previous page)

```
                        (daophot PHOTO) error is greater than x will be
                        excluded form allstar run and have no effect on
                        quality measurment (default 0.1)
  --max-ph-mag m        threshold for photometry magnitude of stars for
                        processing by allstar; stars for which aperture
                        photometry (daophot PHOTO) magnitude is greater than m
                        (fainter than m) will be excluded form allstar run and
                        have no effect on quality measurement (default 20)
  --parallel n, -p n    how many parallel processes can be forked; n=1 avoids
                        parallelism (default: 8)
  --out_dir output_dir, -d output_dir
                        output directory; directory will be created and result
                        files will be stored there; directory should not exist
                        or --overwrite flag should be set (default: do not
                        produce output files)
  --overwrite, -o       if directory specified by --out_dir parameter exists,
                        then ALL its content WILL BE DELETED
  --ga_init_prob x, -I x
                        what portion of candidates is used to initialize GA
                        individuals; e.g. if there is 100 candidates, each of
                        them will be chosen to initialize individual genome
                        with probability x; in other words if x=0.3 first
                        population in GA will contain individuals with around
                        30 stars each; try to make size of first population
                        stars similar to expected number of resulting PDF
                        stars (default: 0.3)
  --ga_max_iter n, -i n
                        maximum number of iterations of generic algorithm -
                        generations (default: 50)
  --ga_pop n, -n n      population size of GA (default: 80)
  --ga_cross_prob x     crossover probability of GA (default: 0.5)
  --ga_mut_prob x       mutation probability of GA - probability to became a
                        mutant (default: 0.2)
  --ga_mut_str x        mutation strength of GA - probability of every bit
                        flip in mutant (default: 0.05)
  --loglevel level, -L level
                        logging level: debug, info, warning, error, critical
                        (default: info)
  --no_stdout, -t       suppress printing result (list of best choice of PSF
                        stars) to stdout at finish
  --no_progress, -b     suppress showing progress bar
  --version, -v         show version and exit
```

Note: Run `gapick --help` for actual set of parameters which can slightly differ from above.

CHAPTER 4

astwro.tools command line tools

Astwro tools are python scripts, ready to use from command line.

Each script has `--help` option to display usage.

All scripts can be also used as python modules (this is the reason they have `py` extension). Each script exports `main(**kwargs)` function which exposes it's functionality. Also scripts exports `info()` function which returns usage string – convenient way to find out script purpose and `main()` parameters for ones working with python interactively.

Some of the scipts are installed in system by `pip install astwro`:

- *`gapick`* see: *gapick - Finding optimal set of PSF stars*
- *`grepfitshdr`*

CHAPTER 5

Deriving a Point-Spread Function in a Crowded Field

5.1 following Appendix III of Peter Stetson's *User's Manual for DAOPHOT II*

5.2 Using `pydaophot` form `astwro` python package

All *italic* text here have been taken from Stetson's manual.

The only input file for this procedure is a FITS file containing reference frame image. Here we use sample FITS form astwro package (NGC6871 I filter 20s frame). Below we get filepath for this image, as well as create instances of `Daophot` and `Allstar` classes - wrappers around `daophot` and `allstar` respectively.

One should also provide `daophot.opt`, `photo.opt` and `allstar.opt` in apropiriete constructors. Here default, build in, sample, `opt` files are used.

```
from astwro.sampledata import fits_image
frame = fits_image()
```

`Daophot` object creates temporary working directory (*runner directory*), which is passed to `Allstar` constructor to share.

```
from astwro.pydaophot import Daophot, Allstar
dp = Daophot(image=frame)
al = Allstar(dir=dp.dir)
```

Daophot got FITS file in construction, which will be automatically **ATTACH**ed.

CHAPTER 6

(1) Run FIND on your frame

Daophot `FIND` parameters `Number of frames averaged, summed` are defaulted to `1,1`, below are provided for clarity.

```
res = dp.FInd(frames_av=1, frames_sum=1)
```

Check some results returned by `FIND`, every method for `daophot` command returns results object.

```
print ("{} pixels analysed, sky estimate {}, {} stars found.".format(res.pixels, res.
↪sky, res.stars))
```

```
9640 pixels analysed, sky estimate 12.665, 4166 stars found.
```

Also, take a look into *runner directory*

```
!ls $dp.dir
```

```
63d38b_NGC6871.fits   allstar.opt      daophot.opt    i.coo
```

We see symlinks to input image and `opt` files, and `i.coo` - result of `FIND`

CHAPTER 7

(2) Run PHOTOMETRY on your frame

Below we run photometry, providing explicitly radius of aperture A1 and IS, OS sky radiuses.

```
res = dp.PHotometry(apertures=[8], IS=35, OS=50)
```

List of stars generated by daophot commands, can be easily get as astwro.starlist.Starlist being essentially pandas.DataFrame:

```
stars = res.photometry_starlist
```

Let's check 10 stars with least A1 error (mag_err column). (pandas style)

```
stars.sort_values('mag_err').iloc[:10]
```

CHAPTER 8

(3) SORT the output from PHOTOMETRY

in order of increasing apparent magnitude decreasing stellar brightness with the renumbering feature. This step is optional but it can be more convenient than not.

`SORT` command of `daophor` is not implemented (yet) in `pydaohot`. But we do sorting by ourself.

```
sorted_stars = stars.sort_values('mag')
sorted_stars.renumber()
```

Here we write sorted list back info photometry file at default name (overwriting existing one), because it's convenient to use default files in next commands.

```
dp.write_starlist(sorted_stars, 'i.ap')
```

```
'i.ap'
```

CHAPTER 9

(4) PICK to generate a set of likely PSF stars

How many stars you want to use is a function of the degree of variation you expect and the frequency with which stars are contaminated by cosmic rays or neighbor stars. [. . .]

```
pick_res = dp.PIck(faintest_mag=20, number_of_stars_to_pick=40)
```

If no error reported, symlink to image file (renamed to `i.fits`), and all daophot output files (`i.*`) are in the working directory of runner:

```
ls $dp.dir
```

```
63d38b_NGC6871.fits       daophot.opt         i.coo
allstar.opt               i.ap                 i.lst
```

One may examine and improve `i.lst` list of PSF stars. Or use `astwro.tools.gapick.py` to obtain list of PSF stars optimised by genetic algorithm.

CHAPTER 10

(5) Run PSF

tell it the name of your complete (sorted renumbered) aperture photometry file, the name of the file with the list of PSF stars, and the name of the disk file you want the point spread function stored in (the default should be fine) [. . .]

If the frame is crowded it is probably worth your while to generate the first PSF with the "VARIABLE PSF" option set to -1 — pure analytic PSF. That way, the companions will not generate ghosts in the model PSF that will come back to haunt you later. You should also have specified a reasonably generous fitting radius — these stars have been preselected to be as isolated as possible and you want the best fits you can get. But remember to avoid letting neighbor stars intrude within one fitting radius of the center of any PSF star.

For illustration we will set `VARIABLE PSF` option, before `PSf()`

```
dp.set_options('VARIABLE PSF', 2)
psf_res = dp.PSf()
```

CHAPTER 11

(6) Run GROUP and NSTAR or ALLSTAR on your NEI file

If your PSF stars have many neighbors this may take some minutes of real time. Please be patient or submit it as a batch job and perform steps on your next frame while you wait.

We use `allstar`. (`GROUP` and `NSTAR` command are not implemented in current version of `pydaophot`). We use prepared above `Allstar` object: `al` operating on the same runner dir that `dp`.

As parameter we set input image (we haven't do that on constructor), and `nei` file produced by `PSf()`. We do not remember name `i.psf` so use `psf_res.nei_file` property.

Finally we order `allstar` to produce subtracted FITS .

```
alls_res = al.ALlstar(image_file=frame, stars=psf_res.nei_file, subtracted_image_file=
↪'is.fits')
```

All `result` objects, has `get_buffer()` method, useful to lookup unparsed `daophot` or `allstar` output:

```
print (alls_res.get_buffer())
```

```
    63d38b_NGC6871...


                                    Picture size:   1250  1150


   File with the PSF (default 63d38b_NGC6871.psf):             Input file (default␣
↪63d38b_NGC6871.ap):                   File for results (default i.als):          ␣
↪Name for subtracted image (default is):
    915 stars.  <<


I = iteration number

R = number of stars that remain

D = number of stars that disappeared
```

(continues on next page)

(continued from previous page)

```

C = number of stars that converged



     I       R       D       C
     1     915       0       0  <<
     2     915       0       0  <<
     3     915       0       0  <<
     4     724       0     191  <<
     5     385       0     530  <<
     6     211       0     704  <<
     7     110       0     805  <<
     8      67       0     848  <<
     9      40       0     875  <<
    10       0       0     915

    Finished i


Good bye.
```

CHAPTER 12

(8) EXIT from DAOPHOT and send this new picture to the image display

Examine each of the PSF stars and its environs. Have all of the PSF stars subtracted out more or less cleanly, or should some of them be rejected from further use as PSF stars? (If so use a text editor to delete these stars from the LST file.) Have the neighbors mostly disappeared, or have they left behind big zits? Have you uncovered any faint companions that FIND missed?[. . .]

The absolute path to subtracted file (like for most output files) is available as result's property:

```
sub_img = alls_res.subtracted_image_file
```

We can also generate region file for psf stars:

```
from astwro.starlist.ds9 import write_ds9_regions
reg_file_path = dp.file_from_runner_dir('lst.reg')
write_ds9_regions(pick_res.picked_starlist, reg_file_path)
```

```
# One can run ds9 directly from notebook:
!ds9 $sub_img -regions $reg_file_path
```

CHAPTER 13

(9) Back in DAOPHOT II ATTACH the original picture and run SUBSTAR

specifying the file created in step (6) or in step (8f) as the stars to subtract, and the stars in the LST file as the stars to keep.

Lookup into runner dir:

```
ls $al.dir
```

```
63d38b_NGC6871.fits       i.ap                  i.nei
allstar.opt               i.coo                 i.psf
daophot.opt               i.err                 is.fits
i.als                     i.lst                 lst.reg
```

```
sub_res = dp.SUbstar(subtract=alls_res.profile_photometry_file, leave_in=pick_res.
↪picked_stars_file)
```

You have now created a new picture which has the PSF stars still in it but from which the known neighbors of these PSF stars have been mostly removed

CHAPTER 14

(10) ATTACH the new star subtracted frame and repeat step (5) to derive a new point spread function

CHAPTER 15

(11+. . .) Run GROUP NSTAR or ALLSTAR

```
for i in range(3):
    print ("Iteration {}: Allstar chi: {}".format(i, alls_res.als_stars.chi.mean()))
    dp.image = 'is.fits'
    dp.PSf()
    alls_res = al.ALlstar(image_file=frame, stars='i.nei')
    dp.image = frame
    dp.SUbstar(subtract='i.als', leave_in='i.lst')
print ("Final:       Allstar chi: {}".format(alls_res.als_stars.chi.mean()))
```

```
Iteration 0: Allstar chi: 1.14670601093
Iteration 1: Allstar chi: 1.13409726776
Iteration 2: Allstar chi: 1.1332852459
Final:       Allstar chi: 1.13326229508
```

Check last image with subtracted PSF stars neighbours.

```
!ds9 $dp.SUbstar_result.subtracted_image_file -regions $reg_file_path
```

Once you have produced a frame in which the PSF stars and their neighbors all subtract out cleanly, one more time through PSF should produce a point-spread function you can be proud of.

```
dp.image = 'is.fits'
psf_res = dp.PSf()
print ("PSF file: {}".format(psf_res.psf_file))
```

```
PSF file: /var/folders/kt/1jqvm3s51jd4qbxns7dc43rw0000gq/T/pydaophot_tmpBVHrtR/i.psf
```

CHAPTER 16

API Reference

16.1 API Reference

16.1.1 `astwro.pydaophot` Module

See also:

astwro.pydaophot

Runner classes

`Daophot` and *`Allstar`* are wrappers of *daophot* and *allstar* tools.

class `astwro.pydaophot.`**`Daophot`**(*dir=None*, *image=None*, *daophotopt=None*, *options=None*, *batch=False*)

daophot runner

Object of this class maintains single process of **daophot** and it's working directory.

Methods of this class corresponds to **daophot**'s commands, each of those methods returns result object providing access to daophot screen output as well as easy access to files generated by **daophot** command.

Instance attributes:

Variables

- **`daophotopt`** (*`str`*) – daophotopt.opt file to be copied
- **`OPtion_result`** (`DPOP_OPtion`) – results of command OPtion or initial options reported by *daophot*
- **`ATtach_result`** (`DPOP_ATtach`) – results of command ATtach
- **`SKy_result`** (`DpOp_SKy`) – results of command SKy
- **`FInd_result`** (`DpOp_FInd`) – results of command FInd

- **PHotometry_result** (`DpOp_PHotometry`) – results of command PHotometry
- **PIck_result** (`DpOp_PIck`) – results of command PIck
- **PSf_result** (`DpOp_PSf`) – results of command PSf
- **SOrt_result** (*DpOp_SOrt*) – results of command SOrt (not implemented)
- **SUbstar_result** (`DpOp_SUbstar`) – results of command SUbstar
- **image** (*str*) – image which will be automatically ATTACHed before every run
- ***options*** – options which will be automatically added as OPTION command before every run, can be either:

  dictionary:

  ```
  >>> dp = Daophot()
  >>> dp.options = {'GAIN': 9, 'FI': '6.0'}
  ```

  iterable of tuples:

  ```
  >>> dp.options = [('GA', 9.0), ('FITTING RADIUS', '6.0')]
  ```

  filename string of daophot.opt-formatted file:

  ```
  >>> dp.options = 'config/pydaophot.opt'
  ```

Parameters

- **dir** (*str*) – pathname or TmpDir object - working directory for daophot, if None temp dir will be used and deleted on *Daophot.close()*
- **image** (*str*) – if provided this file will be automatically attached (AT) as first daophot command setting image property has same effect
- **daophotopt** (*str*) – daophot.opt file, if None build in default file will be used, can be added later by *Runner.copy_to_runner_dir(file, 'daophot.opt')*
- **options** (*list or str*) – if provided OPTION command will be automatically attached setting options property has same effect; list of tuples or dict. Do not set WATCH PROGRESS to sth else than -2
- **batch** (*bool*) – whether Daophot have to work in batch mode.

dir

Runner's directory, object of *astwro.utils.TmpDir*, call `Daophot.dir.path` for string path to directory

set_options (*options, value=None*)

Set option(s) before run.

Options can be either:

dictionary: `dp.set_options({'GAIN': 9, 'FI': '6.0'})`

iterable of tuples: `dp.set_options([('GA', 9.0), ('FITTING RADIUS', '6.0')])`

option key, followed by value in 'value' parameter: `dp.set_options('GA', 9.0)`

filename string of allstar.opt-formatted file (file will be symlinked as *allstar.opt*): `dp.set_options('opts/newallstar.opt')`

Warning: Do not set *WATCH PROGRESS* to something else than -2

Parameters

- **options** – described above
- **value** – value if `options` is just single key

Returns results object also accessible as `Daophot.OPtion_result` property

Return type *DPOP_OPtion*

ATtach(*image_file*)

Add daophot ATTACH command to execution queue. Available only in "batch" mode.

If image_file parameter is provided in constructor or by set_image method, ATtach is enqueued automatically (preferred method until multiple ATTACH commands needed in "batch" mode).

Parameters **image_file** (*str*) – image to attach file will be symlinked to work dir as `"i.fits"`, if `None`, *i.fits* (file or symlink) is expected in working dir

Returns results object also accessible as `ATtach_result` property

Return type *DPOP_ATtach*

OPtions(*options*, *value=None*)

Adds daophot OPTION command to execution queue. Available only in "batch" mode.

Use *set_options()* for options which are set after daophot process start

Parameter *options* can be either:

dictionary:

```
>>> dp = Daophot(mode = "batch")
>>> dp.OPtions({'GAIN': 9, 'FI': '6.0'})
```

iterable of tuples:

```
>>> dp.OPtions([('GA', 9.0), ('FITTING RADIUS', '6.0')])
```

option key, followed by value in 'value' parameter:

```
>>> dp.OPtions('GA', 9.0)
```

filename string of daophot.opt-formatted file:

```
>>> dp.OPtions('config/pydaophot.opt')
```

Parameters

- **options** – described above
- **value** – value if *options* is just single key

Returns results object also accessible as `Daophot.OPtion_result` property

Return type *DPOP_OPtion*

SKy()

Runs (or adds to execution queue in batch mode) daophot SKY command.

Returns results object also accessible as `Daophot.SKy_result` property

Return type *DpOp_SKy*

`FInd`(*frames_av=1*, *frames_sum=1*, *starlist_file='i.coo'*)
Runs (or adds to execution queue in batch mode) daophot FIND command.

Parameters

- **`frames_av`** (`int`) – averaged frames in image, default: 1
- **`frames_sum`** (`int`) – summed frames in image, default: 1
- **`starlist_file`** (`str`) – output coo file, default: `"i.coo"`

Returns results object also accessible as `Daophot.FInd_result` property

Return type *DpOp_FInd*

`PHotometry`(*photoopt=None*, *IS=0*, *OS=0*, *apertures=None*, *stars='i.coo'*, *photometry_file='i.ap'*)
Runs (or adds to execution queue in batch mode) daophot PHOTOMETRY command.

Either `photoopt` or `IS`, `OS` and `apertures` have to be set. eg.: `IS=35, OS=50, apertures=[8]`

Parameters

- **`photoopt`** (`str`) – photo.opt file to be used, default: None (provide `IS`, `OS` and `apertures`)
- **`IS`** (`float`) – inner sky radius, overwrites `photoopt` file value IS
- **`OS`** (`float`) – outer sky radius, overwrites `photoopt` file value OS
- **`apertures`** (`list`) – apertures radius, up to 12, overwrites `photoopt` file values A1, A2, . . .
- **`stars`** (`str or StarList`) – input list of stars, default: `"i.coo"`
- **`photometry_file`** (`str`) – output magnitudes file

Returns results object also accessible as *Daophot.PHotometry_result* property

Return type *DpOp_PHotometry*

See also:

`NEda()`

`PIck`(*number_of_stars_to_pick=50*, *faintest_mag=20.0*, *photometry='i.ap'*, *picked_stars_file='i.lst'*)
Runs (or adds to execution queue in batch mode) daophot PICK command.

Parameters

- **`int`** – number_of_stars_to_pick
- **`faintest_mag`** (`float`) – instrumental magnitude for the faintest star considered to be picked
- **`photometry`** (`str or StarList`) – input magnitudes file or *`StarList`*, usually from aperture photometry done by *`PHotometry()`*.
- **`picked_stars_file`** (`str`) – output list of picked stars, default: i.lst

Returns results object also accessible as **:var:‘Daophot.PIck_result‘** property

Return type *DpOp_PIck*

PSf (*photometry='i.ap'*, *psf_stars='i.lst'*, *psf_file='i.psf'*)
Runs (or adds to execution queue in batch mode) daophot PHOTOMETRY command.

Parameters

- **or sl.StarList photometry** (`str`) – input magnitudes file or Starlist, e.g. from aperture photometry by `PHotometry()`.
- **or sl.StarList psf_stars** (`str`) – input list of PSF stars, default: i.coo
- **psf_file** (`str`) – output PSF file, default: i.psf

Returns results object also accessible as *Daophot.PSf_result* property

Return type *DpOp_PSf*

SOrt (*file*, *by*, *decreasing=None*)
Adds daophot SORT command to execution stack. NOT IMPLEMENTED sorry

Use sorting capabilities of *StarList* and *StarList.renumber()* :param str file: fname.COO_FILE etc… any fname.*_FILE to sort :param by: 1-based column number, negative for descending order - daophot standard, or

one of 'id', 'x', 'y', 'mag'

Parameters decreasing (`bool`) – in not None, forces sort order

Returns results object, also accessible as *Daophot.SOrt_result* property

Return type DpOp_Sort

SUbstar (*subtract*, *leave_in=None*, *subtracted_image='is.fits'*, *psf_file='i.psf'*)
Adds daophot SUBSTAR command to execution stack.

Parameters

- **subtract** – relative to work dir pathname of stars to subtract file
- **leave_in** – relative to work dir pathname of stars to be kept file (default: None)
- **psf_file** – relative to work dir pathname of file with PSF (default i.psf)
- **subtracted_image** – relative to work dir pathname of output fits file (default is.fits)

Returns results object, also accessible as *Daophot.SUbstar_result* property

GRoup (*photometry='i.ap'*, *psf_file='i.psf'*, *critical_overlap=0.1*, *groups_file='i.grp'*)
Runs (or adds to execution queue in batch mode) daophot GROUP command to execution stack.

Parameters

- **photometry** (`str`, `StarList`) – stars to be grouped
- **psf_file** (`str`) – file with PSF
- **critical_overlap** (`float`) – relative to work dir pathname of file with PSF
- **groups_file** (`str`) – output gouped stars file

Returns results object, also accessible as *GRoup_result* property

NEda (*photoopt=None*, *IS=0*, *OS=0*, *apertures=None*, *psf_file='i.psf'*, *psf_photometry='i.als'*, *stars_id='i.als'*, *neda_photometry_file='i.nap'*)
Runs (or adds to execution queue in batch mode) daophot NEDA command.

Performing aperture photometry with neighbours SPF profiles subtraction. Either :param photoopt or :param photo_is, :param OS and :param photo_ap have to be set. :param [str] photoopt: photo.opt file to be used, default: none

(provide :param photo_is, :param OS and :param photo_ap)

Parameters

- **IS** (*float*) – inner sky radius, overwrites :param photoopt file value IS
- **OS** (*float*) – outer sky radius, overwrites :param photoopt file value OS
- **apertures** (*[list]*) – apertures radius, up to 12, overwrites *photoopt* file values A1, A2, …
- **psf_file** (*str*) – file with PSF for profile subtraction
- **sl.StarList] psf_photometry** (*[str,*) – stars with PSF photometry for profile subtraction, default: i.als
- **sl.StarList] stars_id** (*[str,*) – input list of stars to be measured, default: i.als
- **neda_photometry_file** (*str*) – output neda aperture photometry file

Returns results object, also accessible as *NEda_result* property

Return type DpOp_NEda

exception ExitError(*message*, *runner*, *code*)
Exceptions raised when underlying process returns error code on exit

exception NoFileError(*message*, *runner*, *filename*)

exception RunnerException(*message*, *runner*)
Exceptions raised by *Runner* and subclasses

exception RunnerTypeError

exception RunnerValueError

absolute_path(*path*)
Returns absolute path for filepath parameter, if :arg:path contain filename only, runner dir is added

apertures_file_create(*apertures*, *IS*, *OS*)
Creates photo.opt in daophot working dir from list :param list apertures: list of apertures A1,A2… e.g. [6.0,8.0,12.0] :param float IS: inner radius of sky annulus :param float OS: outer radius of sky annulus :rtype: None

apertures_file_pull(*dst_path='.'*)
Extracts current aperture file photo.opt from working dir. :param dst_path: destination :rtype: None

apertures_file_push(*src_path*)
Copies aperture file photo.opt into working dir. File will be used by daophot :param str src_path: patch to src file :rtype: None

clone()
Clones runner

If *runner directory* was provided in constructor, clone will share the same dir, else, if *runner directory* is temp dir created implicitly by runner, clone will create it's own one, and content of *runner directory* will be copied from source to clone.

close()
Cleans things up.

copy_from_runner_dir (*filename*, *dest='./'*)
: Copies file: filename from runner dir. Overwrites existing file.

copy_to_runner_dir (*source*, *filename=None*)
: Copies source file to runner dir under name filename or the same as original if filename is None. Overwrites existing file.

exists_in_runner_dir (*filename*)
: Checks for filename existence in runner dir

static expand_path (*path*)
: Expand user ~ directory and finds absolute path.

file_from_runner_dir (*filename*)
: Simply adds runner dir path into filename

has_finished_run ()
: Returns True if process has finished and output is available

  Returns bool

is_ready_to_run ()
: Returns True if there are some commands waiting for run but process was not started yet

  Returns bool

link_from_runner_dir (*filename*, *dest='./'*)
: Creates symlink in dest of file from runner dir. dest can be either file path for new symlink or directory. In second case name of symlink will be filename. Overwrites existing file.

link_to_runner_dir (*source*, *link_filename=None*)
: Creates symlink in runner dir under name filename or the same as original if filename is None. Overwrites existing link. :param source: file patch :param link_filename: worker dir link name, default: same as filename part of source

mode
: Either "normal" or "batch". In batch mode, commands are not executed but collected on execution queue, then run together, in single process, one by one, triggered by `run()` method

read_starlist (*filepath*, ***kwargs*)
: Returns *StarList* object with stars extracted from daophot files :param [str] filepath: source file for starlist, if filename without path is provided, runner directory is assumed. :param kwargs: additional parameters for extra processing in subclasses e.g. add_psf_errors=True :rtype: starlist.StarList

rm_from_runner_dir (*filename*)
: Removes (if exists) file filename from runner dir

run (*wait=True*)
: Execute commands queue.

  In the "normal" `mode` there is no need to call `run()`, because all commands are executed immediately. In "batch" `mode`, commands execution is queued and postponed until `run()`

  Parameters **wait** (`bool`) – If false, `run()` exits without waiting for finishing commands executions (asynchronous processing). Call `wait_for_results()` before accessing results.

  Returns None

running
: Whether if runner is running

  `True` If executable was started in async mode `run(wait=False)`, and no output collected yet.

Note: Even if executable has finished, output will not be collected and `running` will return `True` until user asks for results or call `wait_for_results()`

Returns bool

`wait_for_results``()`
In the "batch" mode, waits for commands completion if `run(wait=False)` was called

`write_starlist``(`*stars*, *filename=None*, *dao_file_type=None*`)`
Writes *StarList* object to file in runner directory :param sl.StarList stars: star list to be written :param filename: name of file in runner directory, default: random name with extension '.stars' :return name of file in runner directory

class `astwro.pydaophot.`**`Allstar`**`(`*dir=None*, *image=None*, *allstaropt=None*, *options=None*, *batch=False*`)`
daophot runner

Object of this class maintains single process of *allstar* and it's working directory.

Instance attributes: :var str allstaropt: allstar.opt file to be copied into runner dir :var DPOP_OPtion OPtion_result: initial options reported by *allstar* :var DPOP_ATtach ATtach_result: results of command ATtach :var str image: image which will be automatically used if not provided in *ALlstars* command :var dict options: options which will be automatically set as OPTION command before every run,

can be either:

dictionary:

```
>>> dp = Allstar()
>>> dp.options = {'PROFILE ERROR': 5, 'FI': '6.0'}
```

iterable of tuples:

```
>>> dp.options = [('PR', 5.0), ('FITTING RADIUS', 6.0)]
```

Parameters

- **`dir`** (`[str]`) – pathname or TmpDir object - working directory for daophot, if None temp dir will be used and deleted on *Allstar.close()*
- **`image`** (`[str]`) – if provided this file will be used if not provided in *ALlstars* command setting image property has same effect
- **`allstaropt`** (`[str]`) – allstar.opt file, if None build in default file will be used, can be added later by *Runner.copy_to_runner_dir(file, 'allstar.opt')*
- **`options`** (`[list, dict]`) – if provided options will be set on beginning of each process list of tuples or dict
- **`batch`** (`bool`) – whether Allstar have to work in batch mode.

`set_options``(`*options*, *value=None*`)`
set option(s) before run.

Options can be either: dictionary: *dp.OPtion({'GAIN': 9, 'FI': '6.0'})* iterable of tuples: *dp.OPtion([('GA', 9.0), ('FITTING RADIUS', '6.0')])* option key, followed by value in 'value' parameter:

dp.OPtion('GA', 9.0)

filename string of allstar.opt-formatted file (file will be symlinked as *allstar.opt*):
dp.OPtion('opts/newallstar.opt')

Once set, options will stay set in next runs, set option to *None* to unset

ALlstar(*image_file=None*, *psf_file='i.psf'*, *stars='i.ap'*, *profile_photometry_file='i.als'*, *subtracted_image_file=None*)
Runs (or adds to execution queue in batch mode) daophot PICK command. :param [str] image_file: input image filepath, if None, one set in constructor or 'i.fits' will be used :param str psf_file: input file with psf from daophot PSF command :param str stars: input magnitudes file, e.g. from aperture photometry done by `Daophot.PHotometry()`. :param str profile_photometry_file: output file with aperture photometry results, default: i.als :param str subtracted_image_file: output file with subtracted FITS image, default: do not generate image :return: results object also accessible as **:var:'Allstar.ALlstars_result'** property :rtype: AsOp_result

exception ExitError(*message*, *runner*, *code*)
Exceptions raised when underlying process returns error code on exit

exception NoFileError(*message*, *runner*, *filename*)

exception RunnerException(*message*, *runner*)
Exceptions raised by *Runner* and subclasses

exception RunnerTypeError

exception RunnerValueError

absolute_path(*path*)
Returns absolute path for filepath parameter, if :arg:path contain filename only, runner dir is added

apertures_file_create(*apertures*, *IS*, *OS*)
Creates photo.opt in daophot working dir from list :param list apertures: list of apertures A1,A2... e.g. [6.0,8.0,12.0] :param float IS: inner radius of sky annulus :param float OS: outer radius of sky annulus :rtype: None

apertures_file_pull(*dst_path='.'*)
Extracts current aperture file photo.opt from working dir. :param dst_path: destination :rtype: None

apertures_file_push(*src_path*)
Copies aperture file photo.opt into working dir. File will be used by daophot :param str src_path: patch to src file :rtype: None

clone()
Clones runner

If *runner directory* was provided in constructor, clone will share the same dir, else, if *runner directory* is temp dir created implicitly by runner, clone will create it's own one, and content of *runner directory* will be copied from source to clone.

close()
Cleans things up.

copy_from_runner_dir(*filename*, *dest='./'*)
Copies file: filename from runner dir. Overwrites existing file.

copy_to_runner_dir(*source*, *filename=None*)
Copies source file to runner dir under name filename or the same as original if filename is None. Overwrites existing file.

exists_in_runner_dir(*filename*)
Checks for filename existence in runner dir

static expand_path(*path*)
Expand user ~ directory and finds absolute path.

file_from_runner_dir(*filename*)
Simply adds runner dir path into filename

has_finished_run()
Returns True if process has finished and output is available

Returns bool

is_ready_to_run()
Returns True if there are some commands waiting for run but process was not started yet

Returns bool

link_from_runner_dir(*filename*, *dest='./'*)
Creates symlink in dest of file from runner dir. dest can be either file path for new symlink or directory. In second case name of symlink will be filename. Overwrites existing file.

link_to_runner_dir(*source*, *link_filename=None*)
Creates symlink in runner dir under name filename or the same as original if filename is None. Overwrites existing link. :param source: file patch :param link_filename: worker dir link name, default: same as filename part of source

mode
Either "normal" or "batch". In batch mode, commands are not executed but collected on execution queue, then run together, in single process, one by one, triggered by `run()` method

read_starlist(*filepath*, ***kwargs*)
Returns *StarList* object with stars extracted from daophot files :param [str] filepath: source file for starlist, if filename without path is provided, runner directory is assumed. :param kwargs: additional parameters for extra processing in subclasses e.g. add_psf_errors=True :rtype: starlist.StarList

rm_from_runner_dir(*filename*)
Removes (if exists) file filename from runner dir

run(*wait=True*)
Execute commands queue.

In the "normal" `mode` there is no need to call `run()`, because all commands are executed immediately. In "batch" `mode`, commands execution is queued and postponed until `run()`

Parameters **wait** (`bool`) – If false, `run()` exits without waiting for finishing commands executions (asynchronous processing). Call `wait_for_results()` before accessing results.

Returns None

running
Whether if runner is running

`True` If executable was started in async mode `run(wait=False)`, and no output collected yet.

Note: Even if executable has finished, output will not be collected and `running` will return `True` until user asks for results or call `wait_for_results()`

Returns bool

wait_for_results()
In the "batch" mode, waits for commands completion if `run(wait=False)` was called

write_starlist(*stars*, *filename=None*, *dao_file_type=None*)
Writes *StarList* object to file in runner directory :param sl.StarList stars: star list to be written :param filename: name of file in runner directory, default: random name with extension '.stars' :return name of file in runner directory

Command Results

Results of *daophot* and *allstar* commands execution are available as *Output Providers* objects

class astwro.pydaophot.OutputProviders.**DPOP_ATtach**(*prev_in_chain=None*)
Results of *ATTACH* daophot command

picture_size
tuple with (x,y) size of pic returned by 'ATTACH'

raise_if_error()
Should raise exception if not properly processed: - command did not run - buffer analysis indicates error - no output value found… User can call it to check if command was successful To be overridden

class astwro.pydaophot.OutputProviders.**DPOP_OPtion**(*prev_in_chain=None*)
Results of *OPTION* daophot command, or initial daophot options

options
Dictionary of options: XX: 'nnn.dd' keys are two letter option names values are strings

get_option(*key*)
single option

raise_if_error()
Should raise exception if not properly processed: - command did not run - buffer analysis indicates error - no output value found… User can call it to check if command was successful To be overridden

class astwro.pydaophot.OutputProviders.**DpOp_SKy**(*prev_in_chain=None*)
Results of *SKY* daophot command

sky
Sky estimation

skydev
Standart deviation of :var sky

mean
Mean of image

median
Median of image

pixels
Number of analyzed pixels

class astwro.pydaophot.OutputProviders.**DpOp_FInd**(*prev_in_chain=None*, *starlist_file=None*)
Results of *FIND* daophot command (extends SKY)

starlist_file = None
Patch to output file with found stars

found_starlist
StarList with found stars

err
Error estimation

stars
Number of found stars

class astwro.pydaophot.OutputProviders.**DpOp_PHotometry**(*prev_in_chain=None*, *photometry_file=None*)

Results of *PHOTOMETRY* daophot command

photometry_file = None
Patch to output file with aperture photometry

photometry_starlist
StarList with photometry

Return type *astwro.starlist.StarList*

class astwro.pydaophot.OutputProviders.**DpOp_PIck**(*prev_in_chain=None*, *picked_stars_file=None*)

Results of *PICK* daophot command

picked_stars_file = None
Patch to output file with picked stars

stars
Number of picked stars

picked_starlist
StarList with picked stars

class astwro.pydaophot.OutputProviders.**DpOp_PSf**(*prev_in_chain=None*, *psf_file=None*, *nei_file=None*, *err_file=None*)

Results of *PSF* daophot command

psf_file = None
Patch to output file with PSF function

nei_file = None
Patch to output neighbours file

err_file = None
Patch to output errors file

nei_starlist
StarList with neighbours stars

converged
False if daophot PSF routine does not produced result, e.g. 'Failed to converge'

errors
StarList (pandas.DataFrame) of PSF stars with errors and flags ('?', '*' or ' ')

This information is identical to i.err file, but obtained directly from daophot output

chi
Chi error estimation

hwhm_xy
tuple (x,y) of halfwidth's of PSF function

raise_if_error()
Should raise exception if not properly processed: - command did not run - buffer analysis indicates error - no output value found… User can call it to check if command was successful To be overridden

class astwro.pydaophot.OutputProviders.**DpOp_SUbstar**(*prev_in_chain=None*, *subtracted_image_file=None*)

Results of *SUBSTAR* daophot command

subtracted_image_file = None
Patch to output fits image

class astwro.pydaophot.OutputProviders.**DpOp_GRoup**(*prev_in_chain=None*, *groups_file=None*)
Results of *GROUP* daophot command

groups_file = None
Patch to output file with groups

groups_histogram
List of tuples: (size_of_group, number_of_groups)

stars
Number of grouped stars reported by daophot *GROUP* command

groups
Number of groups

16.1.2 `astwro.starlist` Module

Star lists, are objects of class *StarList* which inherits directly from :class:pandas.DataFrame.

StarList class

class astwro.starlist.**StarList**(*data=None*, *index=None*, *columns=None*, *dtype=None*, *copy=False*)
Bases: pandas.core.frame.DataFrame

static new()
Returns empty StarList instance with columns id,x,y

DAO_hdr
DAO file header dict if any

DAO_type
DAO file header dict if any

import_metadata(*src*)
Copies metdata (dao type, dao hdr) from src

Parameters src (StarList) – source of metadata

stars_number()
returns number of stars in list

renumber(*start=1*)
Renumbers starlist (in place), updating *id* column and index to range start.. start+stars_number

refresh_id()
Cheks for id column existence and renumber if needed then recreate index basin on id

If *id* column exist but cannot be casted to *int* (by *pandas.Series.to_numeric*), *ValueError* exception is raised

to_table()
Return a astropy.table.Table instance

classmethod from_table(*table*)
Create a *StarList* from a astropy.table.Table instance

table [astropy.table.Table] The astropy astropy.table.Table instance

sl [*StarList*] A ‘StarList‘instance

classmethod from_skycoord(*coo*)
Create a *StarList* from a `astropy.coordinates.SkyCoord` instance

coo [`astropy.coordinates.SkyCoord`] Source

sl [*StarList*] A ‘StarList‘instance

IO of daophot/allstar files

`astwro.starlist.daofiles.`**convert_dao_type**(*starlist*, *new_daotype*, *update_daotype=True*)
Converts from one daotype to another

Only subset of conversions supported. You can also provide own map (directory) of column names

`astwro.starlist.daofiles.`**read_dao_file**(*file*, *dao_type=None*)
Construct StarList from daophot output file. The header lines in file may be missing. :rtype: StarList :param file: open stream or filename, if stream dao_type must be specified :param dao_type: file format, one of DAO.XXX_FILE constants:

- DAO.COO_FILE
- DAO.AP_FILE
- DAO.LST_FILE
- DAO.NEI_FILE
- DAO.ALS_FILE

If missing filename extension will be used to determine file type if file is provided as filename

Returns StarList instance

`astwro.starlist.daofiles.`**write_dao_file**(*starlist*, *file*, *dao_type=None*, *with_header=None*)
Write StarList object into daophot file. :param starlist: StarList instance to be writen :param file: writable stream or filename, if stream dao_type must be specified :param dao_type: file format, one of DAO.XXX_FILE constants:

- DAO.COO_FILE
- DAO.AP_FILE
- DAO.LST_FILE
- DAO.NEI_FILE
- DAO.ALS_FILE

If missing extension of file will be used to determine file type if file is provided as filename

Parameters **with_header** – True, False or None. If None header will be written if not None in starlist

Return type None

`astwro.starlist.daofiles.`**dump_dao_hdr**(*hdr*, *line_prefix=”*)
returns two line string representation of header dictionary :param dict hdr: dao header dictionary like StarList.DAO_hdr :param str line_prefix: add this prefix at beginning of every line (e.g. comment char) :rtype:str

`astwro.starlist.daofiles.`**`write_dao_header`**(*hdr*, *stream*, *line_prefix=''*)

writes two lines of dao header :param dict hdr: dao header dictionary like StarList.DAO_hdr :param file stream: to write :param str line_prefix: add this prefix at beginning of every line (e.g. comment char)

`astwro.starlist.daofiles.`**`read_dao_header`**(*stream*, *line_prefix=''*)

tries to read dao header, if fails returns already read characters :param file stream: open input file :param line_prefix: additional prefix expected on the beginning of line :return: tuple (header dict, stolen chars)

> if header is detected, reads 2 lines of stream and returns (dict, None) else reads couple of chars and return (None, couple-of-chars)

`astwro.starlist.daofiles.`**`parse_dao_hdr`**(*hdr*, *val*, *line_prefix=''*)

creates dao header dict form two lines of file header :param str hdr: first line :param str val: second line :param line_prefix: expected line prefix :return: dict with dao header compatible with StarList.DAO_header

IO of ds9 files

`astwro.starlist.ds9.`**`read_ds9_regions`**(*file*)

Reads ds9 region :param file: filename or open input stream :return: StarList object

Returned object has columns id, x, y, auto_id

Boolean column auto_id indicates weather id for item is read from file (#id=xxx comment) or generated by function.

`astwro.starlist.ds9.`**`write_ds9_regions`**(*starlist*, *filename*, *color='green'*, *width=1*, *size=None*, *font=None*, *label='{id:.0f}'*, *exclude=None*, *indexes=None*, *colors=None*, *sizes=None*, *labels=None*, *color_column=None*, *size_column=None*, *comment=None*, *add_global=None*, *WCS=False*)

Writes ds9 region file. Some regions can be visually distinguish by providing additional indexes to select those regions with specific attributes :param StarList starlist: StarList object to dump :param str filename: output filename or stream open for writing :param str color: default color :param int width: default line width :param int size: default radius (default 8px or 2") :param str font: ds9 font specification e.g. "times 12 bold italic" :param str label: format expression for label, use col names :param pd.Index exclude: index of disabled regions, if None all are enabled :param [pd.Index] indexes: additional indexes to include specific color and size attributes :param [str] colors: specific colors for indexes :param [int] sizes: specific sizes for indexes :param [str] labels: specific labels for indexes :param str color_column: column of starlist with color values :param str size_column: column of starlist with size values :param str add_global: content of additional 'global' if not None :param str comment: content of additional comment line if not None :param bool or str WCS: If true, columns *ra* and *dec* will be used and coord system set to ICRS

> If nonepmpty string, string will be used as system description If None, False or '', columns 'x','y' will be used and system set to IMAGE

Example: write_ds9_regions(sl, 'i.reg', color='blue',

> indexes=[saturated, psf], colours=['yellow', 'red'], sizes=[12, None], labels=[None, 'PDF:{id}'], exclude=faint)

Generates regions file i.reg of blue circles, radius 8, objects present in index saturated will have larger yellow circles objects present in index psf will be red and labeled with prefix PSF: objects present in index faint will be disabled by '-' sign and not displayed by ds9, but can be parsed back

16.1.3 `astwro.tools` Module

Astwro tools are python scripts, ready to use from command line.

gapick

Find best PSF stars using GA to minimize mean error.

See also:

gapick - Finding optimal set of PSF stars

`astwro.tools.gapick.`**`main`**`(`*`**kwargs`*`)`
Entry point for python script calls. Parameters identical to command line

`astwro.tools.gapick.`**`info`**`()`
Prints commandline help message

grepfitshdr

Grep-like tool for FITS headers

Call commandline: `grepfitshdr --help` for parameters info.

`astwro.tools.grepfitshdr.`**`iter_fields`**`(`*hdr*, *onlyvalues=False*, *fields=None*`)`
splits header into lines if onlyvalues does not return field names if fields returns only specified fields (forces onlyvalues)

`astwro.tools.grepfitshdr.`**`main`**`(`*pattern*, *file*, *`**kwargs`*`)`
Entry point for python script calls. Parameters identical to command line

`astwro.tools.grepfitshdr.`**`info`**`()`
Prints commandline help message

16.1.4 `astwro.utils` Module

`astwro.utils.`**`tmpdir`**`(`*use_existing=None*, *prefix='astwro_tmp_'*, *base_dir=None*`)`
Creates instance of TmpDir which creates and keeps lifetime of temporary directory :param str use_existing: If provided, instance will point to that directory and not delete it on destruct :param str prefix: Prefix for temporary dir :param str base_dir: Where to crate tem dir, in None system default is used :rtype: TmpDir

`astwro.utils.`**`cyclefile`**`(`*path*, *basename*, *extension=''*, *create_symlinks=True*, *symlink_suffix='_last'*, *auto_close=True*`)`
Creates CycleFile which can create series of files with names containing counter, with symlink to newest one :param str path: patch to the file, absolute or relative, can be empty string '' for current directory :param str basename: first part of file name before counter :param str extension: part of filename after counter :param str create_symlinks: whether to create symlink to newest file :param str symlink_suffix: part of suffix filename in place of counter value :param bool auto_close: close prev files on next_file nad destruction :rtype: CycleFile

`astwro.utils.`**`progressbar`**`(`*total=100*, *prefix=''*, *suffix=''*, *decimals=1*, *bar_length=20*, *step=0*`)`
Creates Text console progress bar object, print progress using print_progress() method :param int total: total iterations (Int) :param str prefix: prefix string (Str) :param str suffix: suffix string (Str) :param int decimals: positive number of decimals in percent complete (Int) :param int bar_length: character length of bar (Int) :param int step: allows automatic progress increasing on parameter-less print_progress call :rtype:ProgressBar

Astwro tools are python scripts, ready to use from command line.

TmpDir

`class` `astwro.utils.TmpDir.`**`TmpDir`**`(`*use_existing=None*, *prefix='astwro_tmp_'*, *base_dir=None*`)`
instances of TmpDir keeps track and lifetime of temporary directory

Parameters

- **use_existing** (*str*) – If provided, instance will point to that directory and not delete it on destruct
- **prefix** (*str*) – Prefix for temporary dir
- **base_dir** (*str*) – Where to crate tem dir, in None system default is used

ProgressBar

Greenstick's code form http://stackoverflow.com/questions/3173320/text-progress-bar-in-the-console encapsulated into class

CycleFile

class astwro.utils.CycleFile.**CycleFile**(*path*, *basename*, *extension=''*, *create_symlinks=True*, *symlink_suffix='_last'*, *auto_close=True*)

Creates series of files with names containing counter, with symlink to newest one

Parameters

- **auto_close** (*bool*) –
- **path** (*str*) – patch to the file, absolute or relative, can be empty string '' for current directory
- **basename** (*str*) – first part of file name before counter
- **extension** (*str*) – part of filename after counter
- **create_symlinks** (*str*) – whether to create symlink to newest file
- **symlink_suffix** (*str*) – part of suffix filename in place of counter value
- **auto_close** – close prev files on next_file nad destruction

16.1.5 astwro.sampledata Module

Module contains sample FITS file and other daophot files for testing

astwro.sampledata.**fits_image**()
path of FITS image of NGC6871

astwro.sampledata.**coo_file**()
path of *coo* file for *fits_image()*

astwro.sampledata.**lst_file**()
path of *lst* file for *fits_image()*

astwro.sampledata.**ap_file**()
path of *ap* file for *fits_image()*

astwro.sampledata.**psf_file**()
path of *psf* file for *fits_image()*

astwro.sampledata.**als_file**()
path of *als* file for *fits_image()*

`astwro.sampledata.`**`nei_file`**`()`
: path of *nei* file for `fits_image()`

`astwro.sampledata.`**`head_file`**`()`
: patch of sample ASCII fits header file

Warning: *astwro.pydaophot* and many command line tools requires compatible *DAOPHOT* package installed. *pydaophot* should work with most of modern versions of *daophot II*, but is not compatible with IRAF's daophot.

CHAPTER 17

Indices and tables

- genindex
- modindex
- search

CHAPTER 18

Contact

For any comments or wishes please send an email to the following alias: astwro.0.5@2007.gfdgfdg.com

For any issues, use github tracker: https://github.com/majkelx/astwro/issues

Python Module Index

a

astwro.pydaophot, 39
astwro.pydaophot.OutputProviders, 49
astwro.sampledata, 55
astwro.starlist, 51
astwro.starlist.daofiles, 52
astwro.starlist.ds9, 53
astwro.tools, 53
astwro.tools.gapick, 54
astwro.tools.grepfitshdr, 54
astwro.utils, 54
astwro.utils.CycleFile, 55
astwro.utils.ProgressBar, 55
astwro.utils.TmpDir, 54

Index

A

absolute_path() (astwro.pydaophot.Allstar method), 47
absolute_path() (astwro.pydaophot.Daophot method), 44
Allstar (class in astwro.pydaophot), 46
ALlstar() (astwro.pydaophot.Allstar method), 47
Allstar.ExitError, 47
Allstar.NoFileError, 47
Allstar.RunnerException, 47
Allstar.RunnerTypeError, 47
Allstar.RunnerValueError, 47
als_file() (in module astwro.sampledata), 55
ap_file() (in module astwro.sampledata), 55
apertures_file_create() (astwro.pydaophot.Allstar method), 47
apertures_file_create() (astwro.pydaophot.Daophot method), 44
apertures_file_pull() (astwro.pydaophot.Allstar method), 47
apertures_file_pull() (astwro.pydaophot.Daophot method), 44
apertures_file_push() (astwro.pydaophot.Allstar method), 47
apertures_file_push() (astwro.pydaophot.Daophot method), 44
astwro.pydaophot (module), 39
astwro.pydaophot.OutputProviders (module), 49
astwro.sampledata (module), 55
astwro.starlist (module), 51
astwro.starlist.daofiles (module), 52
astwro.starlist.ds9 (module), 53
astwro.tools (module), 53
astwro.tools.gapick (module), 54
astwro.tools.grepfitshdr (module), 54
astwro.utils (module), 54
astwro.utils.CycleFile (module), 55
astwro.utils.ProgressBar (module), 55
astwro.utils.TmpDir (module), 54
ATtach() (astwro.pydaophot.Daophot method), 41

C

chi (astwro.pydaophot.OutputProviders.DpOp_PSf attribute), 50
clone() (astwro.pydaophot.Allstar method), 47
clone() (astwro.pydaophot.Daophot method), 44
close() (astwro.pydaophot.Allstar method), 47
close() (astwro.pydaophot.Daophot method), 44
converged (astwro.pydaophot.OutputProviders.DpOp_PSf attribute), 50
convert_dao_type() (in module astwro.starlist.daofiles), 52
coo_file() (in module astwro.sampledata), 55
copy_from_runner_dir() (astwro.pydaophot.Allstar method), 47
copy_from_runner_dir() (astwro.pydaophot.Daophot method), 44
copy_to_runner_dir() (astwro.pydaophot.Allstar method), 47
copy_to_runner_dir() (astwro.pydaophot.Daophot method), 45
CycleFile (class in astwro.utils.CycleFile), 55
cyclefile() (in module astwro.utils), 54

D

DAO_hdr (astwro.starlist.StarList attribute), 51
DAO_type (astwro.starlist.StarList attribute), 51
Daophot (class in astwro.pydaophot), 39
Daophot.ExitError, 44
Daophot.NoFileError, 44
Daophot.RunnerException, 44
Daophot.RunnerTypeError, 44
Daophot.RunnerValueError, 44
dir (astwro.pydaophot.Daophot attribute), 40
DPOP_ATtach (class in astwro.pydaophot.OutputProviders), 49
DpOp_FInd (class in astwro.pydaophot.OutputProviders), 49
DpOp_GRoup (class in astwro.pydaophot.OutputProviders), 51

DPOP_OPtion (class in astwro.pydaophot.OutputProviders), 49
DpOp_PHotometry (class in astwro.pydaophot.OutputProviders), 50
DpOp_PIck (class in astwro.pydaophot.OutputProviders), 50
DpOp_PSf (class in astwro.pydaophot.OutputProviders), 50
DpOp_SKy (class in astwro.pydaophot.OutputProviders), 49
DpOp_SUbstar (class in astwro.pydaophot.OutputProviders), 50
dump_dao_hdr() (in module astwro.starlist.daofiles), 52

E

err (astwro.pydaophot.OutputProviders.DpOp_FInd attribute), 49
err_file (astwro.pydaophot.OutputProviders.DpOp_PSf attribute), 50
errors (astwro.pydaophot.OutputProviders.DpOp_PSf attribute), 50
exists_in_runner_dir() (astwro.pydaophot.Allstar method), 47
exists_in_runner_dir() (astwro.pydaophot.Daophot method), 45
expand_path() (astwro.pydaophot.Allstar static method), 47
expand_path() (astwro.pydaophot.Daophot static method), 45

F

file_from_runner_dir() (astwro.pydaophot.Allstar method), 48
file_from_runner_dir() (astwro.pydaophot.Daophot method), 45
FInd() (astwro.pydaophot.Daophot method), 42
fits_image() (in module astwro.sampledata), 55
found_starlist (astwro.pydaophot.OutputProviders.DpOp_FInd attribute), 49
from_skycoord() (astwro.starlist.StarList class method), 52
from_table() (astwro.starlist.StarList class method), 51

G

get_option() (astwro.pydaophot.OutputProviders.DPOP_OPtion method), 49
GRoup() (astwro.pydaophot.Daophot method), 43
groups (astwro.pydaophot.OutputProviders.DpOp_GRoup attribute), 51
groups_file (astwro.pydaophot.OutputProviders.DpOp_GRoup attribute), 51
groups_histogram (astwro.pydaophot.OutputProviders.DpOp_GRoup attribute), 51

H

has_finished_run() (astwro.pydaophot.Allstar method), 48
has_finished_run() (astwro.pydaophot.Daophot method), 45
head_file() (in module astwro.sampledata), 56
hwhm_xy (astwro.pydaophot.OutputProviders.DpOp_PSf attribute), 50

I

import_metadata() (astwro.starlist.StarList method), 51
info() (in module astwro.tools.gapick), 54
info() (in module astwro.tools.grepfitshdr), 54
is_ready_to_run() (astwro.pydaophot.Allstar method), 48
is_ready_to_run() (astwro.pydaophot.Daophot method), 45
iter_fields() (in module astwro.tools.grepfitshdr), 54

L

link_from_runner_dir() (astwro.pydaophot.Allstar method), 48
link_from_runner_dir() (astwro.pydaophot.Daophot method), 45
link_to_runner_dir() (astwro.pydaophot.Allstar method), 48
link_to_runner_dir() (astwro.pydaophot.Daophot method), 45
lst_file() (in module astwro.sampledata), 55

M

main() (in module astwro.tools.gapick), 54
main() (in module astwro.tools.grepfitshdr), 54
mean (astwro.pydaophot.OutputProviders.DpOp_SKy attribute), 49
median (astwro.pydaophot.OutputProviders.DpOp_SKy attribute), 49
mode (astwro.pydaophot.Allstar attribute), 48
mode (astwro.pydaophot.Daophot attribute), 45

N

NEda() (astwro.pydaophot.Daophot method), 43
nei_file (astwro.pydaophot.OutputProviders.DpOp_PSf attribute), 50
nei_file() (in module astwro.sampledata), 55
nei_starlist (astwro.pydaophot.OutputProviders.DpOp_PSf attribute), 50
new() (astwro.starlist.StarList static method), 51

O

options (astwro.pydaophot.OutputProviders.DPOP_OPtion attribute), 49
OPtions() (astwro.pydaophot.Daophot method), 41

P

parse_dao_hdr() (in module astwro.starlist.daofiles), 53
PHotometry() (astwro.pydaophot.Daophot method), 42
photometry_file (astwro.pydaophot.OutputProviders.DpOp_PHotometry attribute), 50
photometry_starlist (astwro.pydaophot.OutputProviders.DpOp_PHotometry attribute), 50
PIck() (astwro.pydaophot.Daophot method), 42
picked_starlist (astwro.pydaophot.OutputProviders.DpOp_PIck attribute), 50
picked_stars_file (astwro.pydaophot.OutputProviders.DpOp_PIck attribute), 50
picture_size (astwro.pydaophot.OutputProviders.DPOP_ATtach attribute), 49
pixels (astwro.pydaophot.OutputProviders.DpOp_SKy attribute), 49
progressbar() (in module astwro.utils), 54
PSf() (astwro.pydaophot.Daophot method), 42
psf_file (astwro.pydaophot.OutputProviders.DpOp_PSf attribute), 50
psf_file() (in module astwro.sampledata), 55

R

raise_if_error() (astwro.pydaophot.OutputProviders.DPOP_ATtach method), 49
raise_if_error() (astwro.pydaophot.OutputProviders.DPOP_OPtion method), 49
raise_if_error() (astwro.pydaophot.OutputProviders.DpOp_PSf method), 50
read_dao_file() (in module astwro.starlist.daofiles), 52
read_dao_header() (in module astwro.starlist.daofiles), 53
read_ds9_regions() (in module astwro.starlist.ds9), 53
read_starlist() (astwro.pydaophot.Allstar method), 48
read_starlist() (astwro.pydaophot.Daophot method), 45
refresh_id() (astwro.starlist.StarList method), 51
renumber() (astwro.starlist.StarList method), 51
rm_from_runner_dir() (astwro.pydaophot.Allstar method), 48
rm_from_runner_dir() (astwro.pydaophot.Daophot method), 45
run() (astwro.pydaophot.Allstar method), 48
run() (astwro.pydaophot.Daophot method), 45
running (astwro.pydaophot.Allstar attribute), 48
running (astwro.pydaophot.Daophot attribute), 45

S

set_options() (astwro.pydaophot.Allstar method), 46
set_options() (astwro.pydaophot.Daophot method), 40
sky (astwro.pydaophot.OutputProviders.DpOp_SKy attribute), 49
SKy() (astwro.pydaophot.Daophot method), 41
skydev (astwro.pydaophot.OutputProviders.DpOp_SKy attribute), 49
SOrt() (astwro.pydaophot.Daophot method), 43
StarList (class in astwro.starlist), 51
starlist_file (astwro.pydaophot.OutputProviders.DpOp_FInd attribute), 49
stars (astwro.pydaophot.OutputProviders.DpOp_FInd attribute), 49
stars (astwro.pydaophot.OutputProviders.DpOp_GRoup attribute), 51
stars (astwro.pydaophot.OutputProviders.DpOp_PIck attribute), 50
stars_number() (astwro.starlist.StarList method), 51
SUbstar() (astwro.pydaophot.Daophot method), 43
subtracted_image_file (astwro.pydaophot.OutputProviders.DpOp_SUbstar attribute), 50

T

TmpDir (class in astwro.utils.TmpDir), 54
tmpdir() (in module astwro.utils), 54
to_table() (astwro.starlist.StarList method), 51

W

wait_for_results() (astwro.pydaophot.Allstar method), 48
wait_for_results() (astwro.pydaophot.Daophot method), 46
write_dao_file() (in module astwro.starlist.daofiles), 52
write_dao_header() (in module astwro.starlist.daofiles), 52
write_ds9_regions() (in module astwro.starlist.ds9), 53
write_starlist() (astwro.pydaophot.Allstar method), 48
write_starlist() (astwro.pydaophot.Daophot method), 46