# imexam

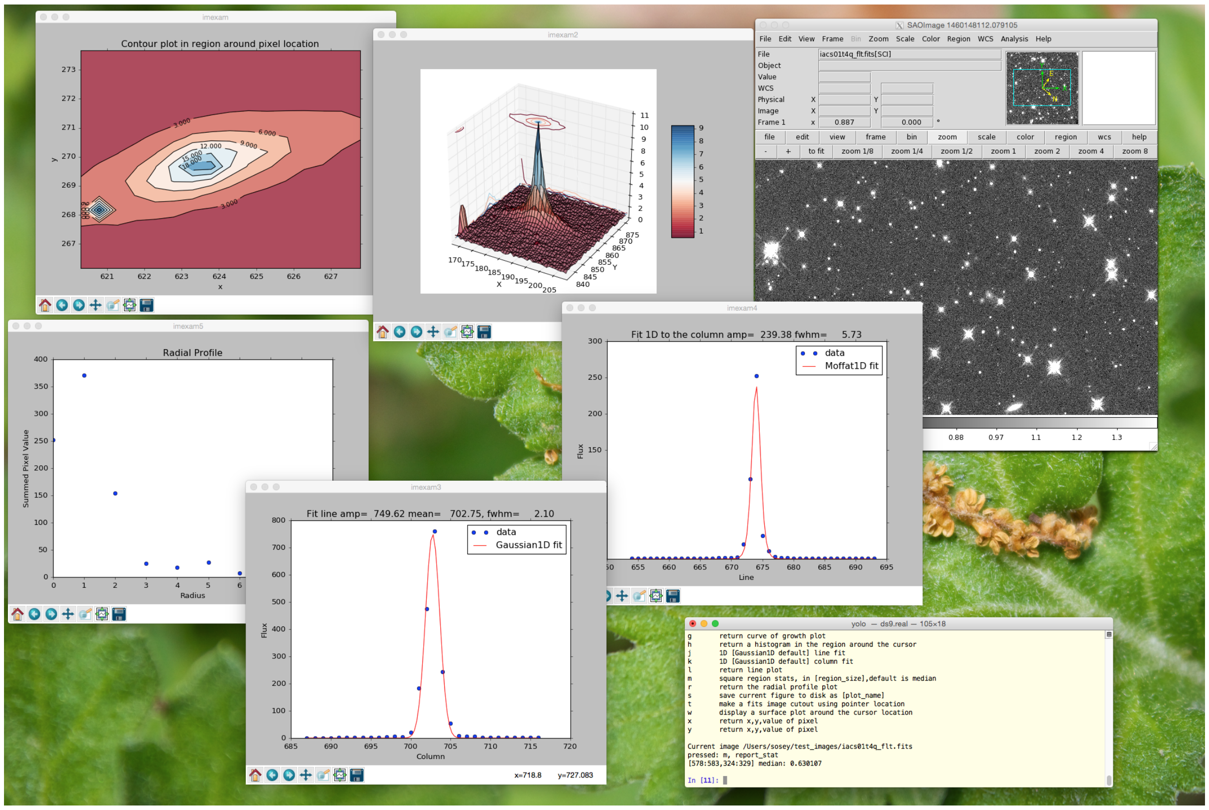
# imexam Documentation

*Release 0.8.0*

Megan Sosey

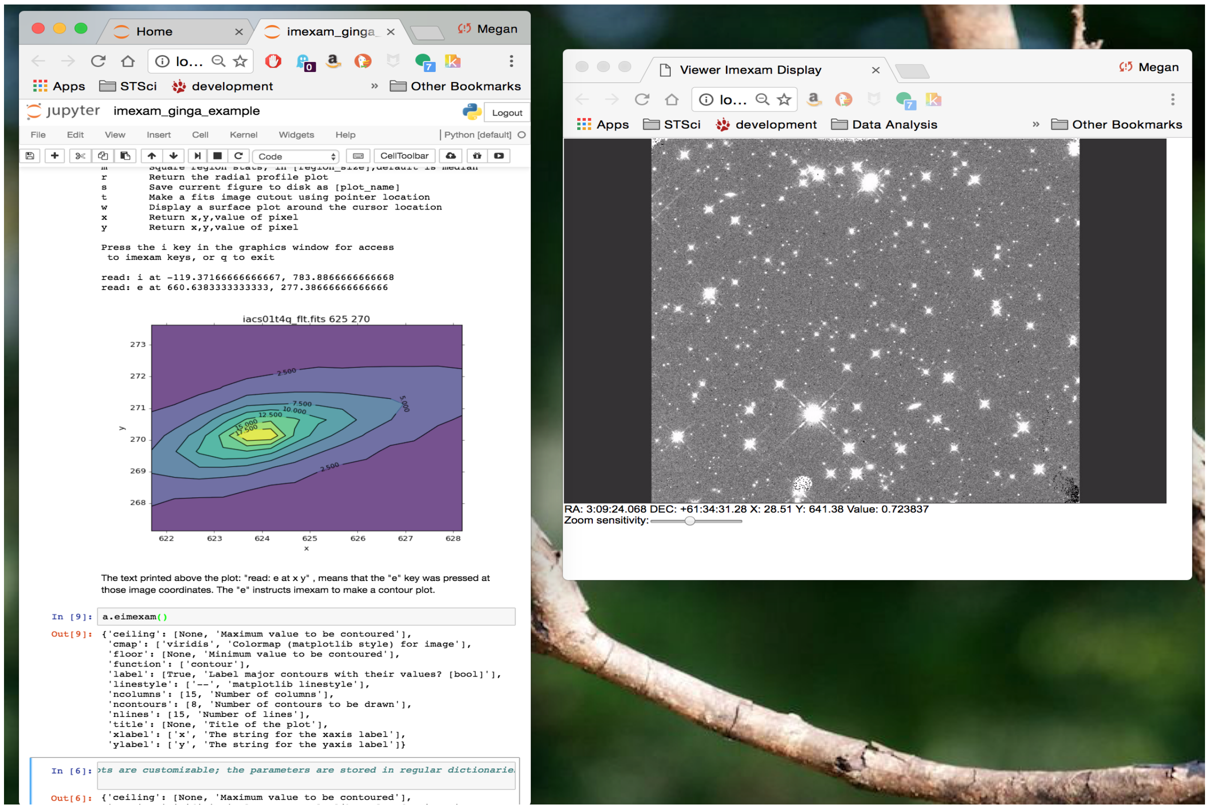
Nov 14, 2017

# Contents


Contour plot in region around pixel location
Radial Profile
Fit 1D to the column amp= 239.38 fwhm= 5.73
Fit line amp= 749.62 mean= 702.75, fwhm= 2.10
data
Moffat1D fit
Gaussian1D fit
SAOImage
yolo — ds9.real — 105×18

```
g       return curve of growth plot
h       return a histogram in the region around the cursor
j       1D [Gaussian1D default] line fit
k       1D [Gaussian1D default] column fit
l       return line plot
m       square region stats, in [region_size],default is median
r       return the radial profile plot
s       save current figure to disk as [plot_name]
t       make a fits image cutout using pointer location
w       display a surface plot around the cursor location
x       return x,y,value of pixel
y       return x,y,value of pixel

Current image /Users/sosey/test_images/iacs01t4q_flt.fits
pressed: m, report_stat
[578:583,324:329] median: 0.630107

In [11]:
```

`imexam` is an affiliated package of AstroPy. It was designed to be a lightweight library which enables users to explore data using common methods which are consistant across viewers. It can be used from a command line interface, through a Jupyter notebook or through a Jupyter console. It can be used with multiple viewers, such as DS9 or Ginga, or without a viewer as a simple library to make plots and grab quick photometry information. The above image is an example desktop interfacing with DS9. Below, is another example desktop using the Jupyter notebook and the Ginga HTML5 viewer.



```
r       Return the radial profile plot
s       Save current figure to disk as [plot_name]
t       Make a fits image cutout using pointer location
w       Display a surface plot around the cursor location
x       Return x,y,value of pixel
y       Return x,y,value of pixel

Press the i key in the graphics window for access
 to imexam keys, or q to exit

read: i at -119.37166666666667, 783.8866666666668
read: e at 660.6383333333333, 277.3866666666666
```

The text printed above the plot: "read: e at x y" , means that the "e" key was pressed at those image coordinates. The "e" instructs imexam to make a contour plot.

```
In [9]: a.eimexam()
Out[9]: {'ceiling': [None, 'Maximum value to be contoured'],
         'cmap': ['viridis', 'Colormap (matplotlib style) for image'],
         'floor': [None, 'Minimum value to be contoured'],
         'function': ['contour'],
         'label': [True, 'Label major contours with their values? [bool]'],
         'linestyle': ['--', 'matplotlib linestyle'],
         'ncolumns': [15, 'Number of columns'],
         'ncontours': [8, 'Number of contours to be drawn'],
         'nlines': [15, 'Number of lines'],
         'title': [None, 'Title of the plot'],
         'xlabel': ['x', 'The string for the xaxis label'],
         'ylabel': ['y', 'The string for the yaxis label']}

In [6]: ots are customizable; the parameters are stored in regular dictionarie

Out[6]: {'ceiling': [None, 'Maximum value to be contoured'],
```

```
RA: 3:09:24.068 DEC: +61:34:31.28 X: 28.51 Y: 641.38 Value: 0.723837
```

The power of this python tool is that it is essentially a library of plotting and analysis routines which can be directed towards any viewer. It attempts to standardize the interface into these functions so that no matter what viewer is in use the calls and results are the same. It can also be used without connecting to any viewer since the calls take only data and location information. This means that given a data array and a list of x,y positions you can create plots and return information without having to interact with the viewers, just by calling the functions directly either from a shell or a private script.

`imexam` may be used as a replacement for the IRAF imexamine task. You should be able to perform all of the most used functions that `imexamine` provided in IRAF, but you also gain the flexibility of python and the ability to add your own analysis functions.

# Part I

# Requirements

This package can be used on Windows, Linux, and MacOS operating systems.

Windows users may download the git repository or do a direct pip install from the git repository. However, they will not have default access to DS9 because compiling the cython+xpa code cannot currently be done with default installed software. Instead, Windows users should make sure they install the Ginga viewer for image examination and plotting using it's HTML5 viewer. You will have all the same imexam functionality available to you, including the use of Jupyter notebooks and screen plotting.

`imexam` currently provides display support two viewers: DS9 and Ginga. The default, when no parameters are supplied to the connect call, is for imexam to start up it's own DS9 process and shut it down nicely upon exit. A Ginga widget using an HTML5 backend is also available as a viewer, most usefull when interacting with the package inside a Jupyter notebook. The package is designed so that it may easily accept other display devices in the future. Additionally, an experimental ginga plugin is available which allows use of the basic ginga gui and interaction with the image display and plots in the imexam style.

The `imexam` library can be used standalone, without a viewer, to create the plots which are available in the interactive sessions by importing the plotting object and feeding the functions your data with x,y coordinates for the plots. It can also be used within the Jupyter notebook framework with either DS9 or the HTML5 backend for viewing. In either case, the images and plots may be saved inside the notebook in conjunction with the notebook (nbAgg) matplotlib backend. If you choose to interact with separate plotting windows, it's still possible to grab an image of the current image display or plot and save it inside the notebook.

**Note:** For DS9, it is important to know if you have XPANS installed on your machine and available through your PATH if you plan to use the nameserver functionality. XPANS is the XPA Name Server, it keeps track of all the open socket connections for DS9 and provides a reference for their names. If you DO NOT have XPANS installed, then `imexam` will still work, but you should either start the DS9 window after importing `imexam` through the imexam.connect() interface, OR after you start DS9 from the shell.

You can connect to an already open DS9 window by specifying the title or the XPA_METHOD. The `XPA_METHOD` is the address in the File->Information dialog. If users don't specify a title in the ds9 window when they open one up, ds9 will just call the window "ds9", so you can end up with multiple windows with the same name. This works for DS9 because the XPA_METHOD is always unique. The most straightforward way is for users to open the DS9 windows with explicit titles, and then tell imexam to connect to that window:

```
python> !ds9 -title megan
python> window=imexam.connect('megan')
```

However, if there are windows already open with no unique titles, the best way is to connect using the method. The `list_active_ds9` function can be used to return a dictionary which contains the information for all the windows, but it's keys are the unique XPA_METHOD strings.:

```
In [3]: !ds9&
In [4]: imexam.list_active_ds9()
DS9 ds9 gs c0a80106:61894 sosey
Out[4]: {'c0a80106:61894': ('ds9', 'sosey', 'DS9', 'gs')}
```

Using this dictionary, you can also you can return the list of windows you can connect to without too much thinking, making it easy to encorporate into your own scripts as well::

```
In [1]: import imexam

In [2]: windows=imexam.list_active_ds9()
DS9 ds9 gs c0a80106:61915 sosey

In [3]: list(windows)
Out[3]: ['c0a80106:61915']

In [4]: !ds9&
In [5]: windows=imexam.list_active_ds9()
DS9 ds9 gs c0a80106:61915 sosey
DS9 ds9 gs c0a80106:61923 sosey

In [6]: list(windows)
Out[6]: ['c0a80106:61915', 'c0a80106:61923']

In [7]: ds9=imexam.connect(list(windows)[0])
```

But you can also use it as below to cycle through connecting to a set of windows::

```
In [8]: windows=imexam.list_active_ds9()
DS9 ds9 gs c0a80106:61915 sosey
DS9 ds9 gs c0a80106:61923 sosey

In [9]: ds=imexam.connect(windows.popitem()[0]) #connect to first window, remove as possible window
In [10]: windows
Out[11]: {'c0a80106:61923': ('ds9', 'sosey', 'DS9', 'gs')}

In [12]: w2=imexam.connect(windows.popitem()[0])

In [13]: windows
Out[31]: {}
```

In order to use the Ginga widget display you must have Ginga installed. More information about Ginga can be found in its package documentation: http://ginga.readthedocs.org/en/latest/. If you are using Python 3 you should also install Pillow which will aid in the image display. The Ginga documentation will tell you of any of it's other dependencies. If you install Ginga you will have access to another display tool for your images and data, the HTML5 widget. You can find the source code on GitHub, but you can also install it with `pip` or `conda`.

You can access this help file on your locally installed copy of the package by using the imexam.display_help() call after import. This will display the help in your web browser.

**Note:** All information returned from this module should be considered an estimate of an actual refined result, more precise analysis of the data should be performed for verification before publication.

# Part II

# How to Install

These are some tips on installing the package, or tracking down problems you might be having during or after installation.

imexam can be installed from the source code in the normal python fashion after downloading it from the git repo:

```
python setup.py install
```

imexam can also be installed using pip or conda, and is included in the Astroconda distribution from STScI:

```
# from PyPI
pip install imexam

# if you already have an older version installed
pip install --upgrade imexam

# from the master trunk on the repository, considered developmental code
pip install git+https://github.com/spacetelescope/imexam.git

#install version 0.6.3 from the git repository, this uses the git tag reference
pip install git+https://github.com/spacetelescope/imexam.git@v0.6.3#egg=imexam

# from the STScI conda release package
conda install imexam -c http://ssb.stsci.edu/astroconda
```

If you want to have access to the photometry features of the `imexam()` analysis, download and install `photutils` - another of the astropy associated packages. The full list of astropy packages can be found here: https://github.com/astropy. If `photutils` is not installed, `imexam` should issue a nice statement saying that the photometry options are not available upon import, and any time an analysis key is pressed during the `imexam()` function loop which requires `photutils` to render a result.

# Part III

# Usage

imexam displays plots using matplotlib, if you find that no windows are popping up after installation it's probably the backend that was loaded. One quick way to get things started is to load ipython and use the %matplotlib magic, this will make sure the proper display backend loads when matplotlib is imported:

```
>ipython
>%matplotlib
>import imexam
```

Matplotlib magic should also be used inside the Jupyter notebook or proper interaction with the plots. Before importing imexam into the notebook, specify the notebook backend if you wish to save your plots into the notebook itself. Otherwise you can use the standard ipython magics.

imexam is a class based library. The user creates an object which is tied to a specific image viewing window, such as a DS9 window. In order to interact with multiple windows the user must create multiple objects. Each object stores all the relevent information about the window and data with which it is associated.

For example, in order to open a new DS9 window and use the object "viewer" to control it, you would issue the command:

```
viewer=imexam.connect()
```

The "viewer" object now has associated methods to view, manipulate and analyze data in the DS9 session. When you start the connection, you also have the option of specifying a currently open DS9 window using the target keyword. This keyword can contain the name, the actual text name that you gave the window, or the address of the window. The address of the window can be found in the File->XPA->Information menu item, is stored as XPA_METHOD, and is of the form "82a7e75f:58576" for INET sockets, and a file path for local sockets. The following is an example of connecting to an already active DS9 window which was started outside of imexam:

```
viewer=imexam.connect("82a7e75f:58576")

or

viewer=imexam.connect("my_window_title")
```

When imexam starts up a DS9 window itself, it will create an inet socket by default. However, imexam will first check to see if XPA_METHOD was set in your environment and default to that option. If you are experiencing problems, or you don't have an internet connection (the two might be related because the XPA structures INET sockets with an ip address), you can set your environment variable XPA_METHOD to local or localhost. This will cause imexam to start a local(unix) socket which will show an XPA_METHOD that is a filename on your computer. imexam defaults to a local socket connection to allow for users who do not have the XPA installed on their machine or available on their PATH.

The full XPA source code is maintained as a submodule to the imexam package. If you don't have the XPA on your path, simply point it to that location, or copy the xpans executable to the location of your choice, and make sure you update your PATH. Any time DS9 is started it will start up the xpa nameserver automatically. Then all the xpans query options will be available through imexam (such as imexam.list_active_ds9()). imexam itself uses Cython wrappers around the get and set methods from the XPA for it's communication which is why the fully installed XPA is not necessary.

If you wish to open multiple DS9 windows outside of imexam, then it's recommended that you give each a unique name. If you've forgotten which window had which name, you can look in the same XPA info menu and use the XPA_NAME specified there. If you haven't given them a unique name, you can list the available windows using imexam.list_active_ds9() (as long as XPANS is running) and specify their unique address.

imexam will attempt to find the current location of the DS9 executable by default, but you may also supply the path to the DS9 executable of your choice using the path keyword when you call connect. The fully optional calling sequence is:

```
imexam.connect(target="",path=None,viewer="ds9",wait_time=10)

Where target is the name of the ds9 window that is already running, path is the location of the ds9␣
↪executable, viewer is the name of the viewer to use (ds9 is the only one which is currently␣
↪activated), and wait_time is the time to wait to establish a connection to the socket before exiting␣
↪the process.
```

If it seems like the ds9 window is opening or hanging, there could be few things going on:

- `imexam` will default to an inet socket connection for the XPA. However, it will first check your environment variable `XPA_METHOD` and preferably use that instead. If you don't have an internet connection, check this environment variable, and set it to "local".
- If things seem in order, it's possible that your machine is waiting for X11 to start up, give it time to start, or when you call `imexam` increase the wait time sufficiently; you can do this by specifying "wait_time=60" when you open your viewing object with connect(). The 60 here is an example of the number of seconds `imexam` should wait before returning a connection error.
- Next, check that the path to the DS9 executable is somewhere on your path and that it has not been aliased to something else. You can check this from any terminal window by trying to start DS9. You can also use the unix "which ds9" command to return the full path to the executable, as well as "ls -al ds9" to return the full path and any soft links which might have been established.

In order to return a list of the current DS9 windows that are running, issue the command:

```
imexam.list_active_ds9()
```

**Note:** More information on DS9 can be found at: http://ds9.si.edu/site/Home.html

If you are using the Ginga widget, the interaction with the `imexam` code stays the same, you simply specify that you would like to use Ginga in the call to connect:

```
viewer=imexam.connect(viewer='ginga')
```

"ginga" tells `imexam` that you'd like to use the Ginga widget with the HTML5 background.

In order to turn logging to a file on, issue the command: window.setlog(). The log will be saved to the default filename imexam_session.log in the current directory unless you give it another filename to use. Here's an example of how that might work:

```
import imexam
window=imexam.connect('ds9')
window.setlog() <-- turns on logging with default filename
window.imexam() <-- all output will be logged to the file and displayed on the screen
window.setlog(on=False) <-- turns off logging to file
window.setlog(filename='my_other_log.txt') <-- turns on logging and sets the save filename
```

The log will look something like this, you can see it contains a mention of the command used along with the results

```
gauss_center
xc=812.984250   yc=706.562612

aper_phot
x        y        radius  flux     mag(zpt=25.00)  sky      fwhm
812.98   706.56   5       1288669.29      9.72     11414.53        4.83

show_xy_coords
813.5 706.625
```

```
gauss_center
xc=812.984250   yc=706.562612

gauss_center
xc=239.856464   yc=233.444783

aper_phot
x       y       radius  flux    mag(zpt=25.00)  sky     fwhm
239.86  233.44  5       126601.26       12.24   11574.32        -12.67

show_xy_coords
253.0 234.75

gauss_center
xc=239.856464   yc=233.444783
```

More detailed examples can be found in the examples section of this documentation.

# Part IV

# Common Problems

You're getting the following error statement when you try to `connect()` to a DS9 window, or display an image:

```
XpaException: Unknown XPA Error : XPAGet returned 0!
```

You can first try using local unix sockets by setting your environment variable `XPA_METHOD` to local:

```
setenv XPA_METHOD local    #csh
```

or if you have a bash-like shell:

```
export XPA_METHOD="local"
```

or if you want to do it from inside Python:

```
import os
os.environ['XPA_METHOD'] = "local"
```

That will create local unix file sockets for communication with ds9. If that doesn't solve the problem, see if your path includes the location of xpans, the XPA name server. If you have it installed, but it's not on your path, put it there.

Alternatively, if you're getting an error on calling `connect()` along the lines of:

```
Connection timeout with the ds9
```

you may want to force XPA to use the "inet" mode, which is the default unless your XPA_METHOD is set. E.g.,

```
setenv XPA_METHOD inet   #csh
export XPA_METHOD='inet' #bash
```

(Or similar based on the examples above)

If you are having display issues, some build problems may exist with the dependency packages which deal with backend graphics, try setting your `matplotlib` backend to "Qt4Agg". You can set this in your .matplotlib/matplotlibrc file

```
backend: Qt4Agg
```

The package works with the Qt5Agg and notebook backends, but on occasion I've seen the matplotlib window take two cycles to update, especially inside the Jupyter notebook with inline plots, meaning you may have to hit the exam key twice for the plot to appear. This issue still needs to be worked out, if you're running into it try using the Qt4Agg backend or plotting outside the notebook and saving the figures through the imexam grab or save calls.

If you get an error about not finding the file "import" when you use the grab() function to save a copy of the DS9 window.

`` `FileNotFoundError: [Errno 2] No such file or directory: 'import' ` ``

"import" is the unix/linux import command, it saves any visible window on an X server and outputs it as an image file, it's included with many macos and linux installations, it's likely not on windows. Users should check their path to see if it's included. This only affects grab() for DS9 which saves a copy of the DS9 window on the workspace, it does not affect saves for ginga or matplotlib plots.

`imexam` switched to using `import` to get around a bug in the XPA for the `saveimage` call to the XPA. The DS9 `saveimage` function basically does a screen capture. In the case of MacOSX (and XQuartz) when you are configured to be rootless, the screen capture fails if your DS9 window is not in the upper left corner of the primary screen - the call should work if you are using a laptop that is not connected to a larger display, or a workstation with only one monitor. Since these are harder things to automatically grab from user environments, the workaround was to 'Print' to a file, generating a postscript image that can be rendered outside of ds9 (for example /Applications/Preview). However, I was unable to get this to save to file, the functions it insisted on sending the image directly to the printer. This also makes for greater unknowns on user machines. The workarounds for users who hit this may be:

- screen grab a copy of the window yourself (grabbing saves any overlays as well)
- move the DS9 window to the appropriate screen and issue the saveimage command, assuming "a" is your control object, that would look like: a.window.xpa.set("saveimage ds9.jpeg")

If you are experiencing an issue not related to those descibed above you can open a new issue on the `imexam` GitHub issue tracker. You can view older closed issues there as well.

# Part V

# User documentation

CHAPTER 1

# The imexam() method

This is the main method which allows live interaction with the image display when you are viewing your image or data array. If you execute imexam() while using the Ginga widget, it will display the available options, however they are always available for use via keystroke and are event-driven (using the same keys described below). In order to turn the key-press capture on and off while you have your mouse in the Ginga widget press the "i" key. Either the "i" or "q" key can be used to quit out of the examination mode.

**imexam ():**
access realtime imexamine functions through the keyboard and mouse

**Current recognized keys available during imexam are:**

```
2   Make the next plot in a new window
a   Aperture sum, with radius region_size
b   Return the 2D gauss fit center of the object
c   Return column plot
e   Return a contour plot in a region around the cursor
g   Return curve of growth plot
h   Return a histogram in the region around the cursor
j   1D [Gaussian1D default] line fit
k   1D [Gaussian1D default] column fit
l   Return line plot
m   Square region stats, in [region_size],default is median
r   Return the radial profile plot
s   Save current figure to disk as [plot_name]
t   Make a fits image cutout using pointer location
w   Display a surface plot around the cursor location
x   Return x,y,value of pixel
y   Return x,y,value of pixel


 aimexam(): return a dict of current parameters for aperture photometery

 cimexam(): return dict of current parameters for column plots

 eimexam(): return dict of current parameters for contour plots
```

```
himexam(): return dict current parameters for histogram plots

jimexam(): return dict current parameters for 1D line plots

kimexam(): return dict of current parameters for 1D column plots

limexam(): return dict of current parameters for  line plots

gimexam(): return dict of current parameters for curve of growth plots

rimexam(): return the dict of current parameters for radial profile plots

wimexam(): return dict of current parameters for surface plots

mimexam(): return dict of current parameters for area statistics

timexam(): return a dict of the current parameters for image cutouts
```

**Note:** Some of the plots accept a marker type, any valid Matplotlib marker may be specified. See this page for the full list: http://matplotlib.org/api/markers_api.html#module-matplotlib.markers

The `imexam` key dictionary is stored inside the user object as <object_name>.exam.imexam_option_funcs{}. Each key in the dictionary is the keyboard key to recognize from the user, it's associated value is a tuple which contains the name of the function to call and a description of what that function does. "q" is always assumed to be the returned key when the user wishes to quit interaction with the window. Users may change the default settings for each of the imexamine recognized keys by editing the associated dictionary. You can edit it directly, by accessing each of the values by their keyname and then reset mydict to values you prefer. You can also create a new dictionary of function which maps to your own

However, you can access the same dictionary and customize the plotting parameters using `set_plot_pars`. In the following example, I'm setting three of the parameters for the contour map, whose imexam key is "e":

```
#customize the plotting parameters (or any function in the imexam loop)
a.set_plot_pars('e','title','This is my favorite galaxy')
a.set_plot_pars('e','ncontours',4)
a.set_plot_pars('e','cmap','YlOrRd') #see http://matplotlib.org/users/colormaps.html
```

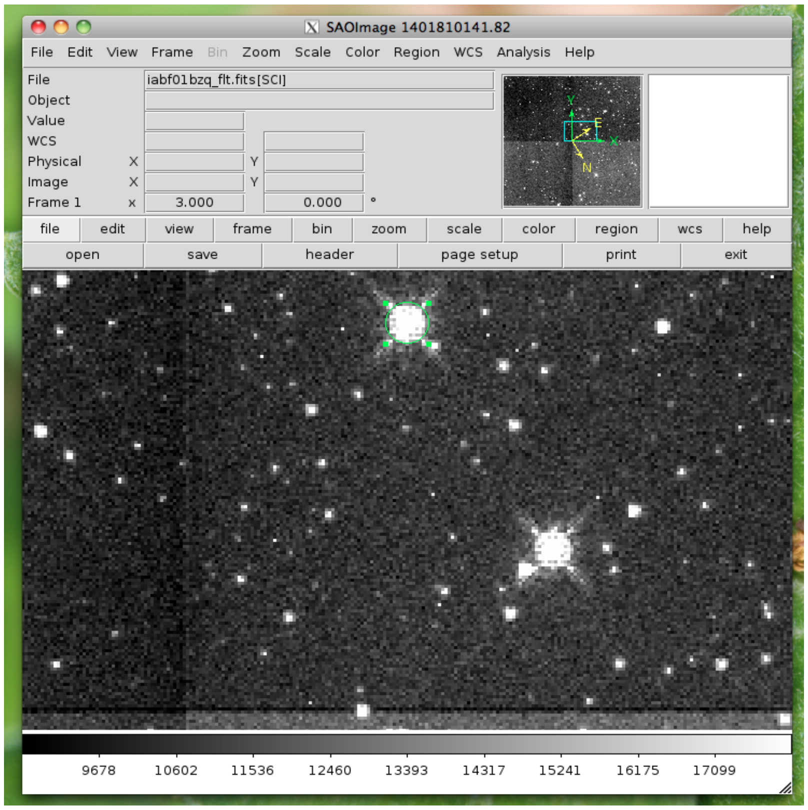
where the full dictionary of available values can be found using the `eimexam()` function described above.:

```
In [1]: a.eimexam()
Out[2]:
{'ceiling': [None, 'Maximum value to be contoured'],
 'cmap': ['RdBu', 'Colormap (matplotlib style) for image'],
 'floor': [None, 'Minimum value to be contoured'],
 'function': ['contour'],
 'label': [True, 'Label major contours with their values? [bool]'],
 'linestyle': ['--', 'matplotlib linestyle'],
 'ncolumns': [15, 'Number of columns'],
 'ncontours': [8, 'Number of contours to be drawn'],
 'nlines': [15, 'Number of lines'],
 'title': [None, 'Title of the plot'],
 'xlabel': ['x', 'The string for the xaxis label'],
 'ylabel': ['y', 'The string for the yaxis label']}
```

Users may also add their own `imexam` keys and associated functions by registering them with the register(user_funct=dict()) method. The new binding will be added to the dictionary of imexamine functions as long as the key is unique. The new functions do not have to have default dictionaries association with them, but users are free to create them.

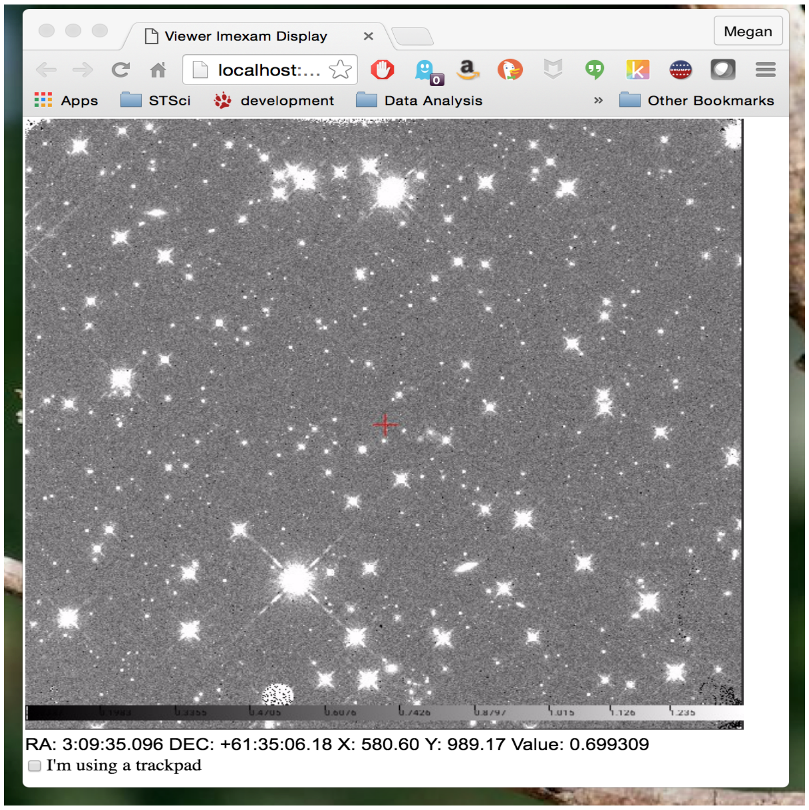
For all the examples below I will use a session similar to the following example:

```
#This will default to DS9 for the viewer

import imexam
viewer=imexam.connect()
viewer.load_fits('iabf01bzq_flt.fits')
viewer.scale()
viewer.panto_image(576,633)
viewer.zoom(3)
```


SAOImage 1401810141.82
File Edit View Frame Bin Zoom Scale Color Region WCS Analysis Help
File iabf01bzq_flt.fits[SCI]
Object
Value
WCS
Physical X Y
Image X Y
Frame 1 x 3.000 0.000 °
file edit view frame bin zoom scale color region wcs help
open save header page setup print exit
9678 10602 11536 12460 13393 14317 15241 16175 17099

This will use Ginga (instead of the default DS9) for the viewer:

```
#Use Ginga for the image viewer, make sure it is installed

import imexam
viewer=imexam.connect(viewer='ginga')
viewer.load_fits('iabf01bzq_flt.fits')
viewer.scale()
viewer.panto_image(576,633)
viewer.zoom(3)
```



RA: 3:09:35.096 DEC: +61:35:06.18 X: 580.60 Y: 989.17 Value: 0.699309

## 1.1 Circular Apterture Photometry

Aperture photometry is performed when you press the "a" key. It is implemented using the photutils python package, an affiliated package of astropy that is still in development.

Currently, the calculation which is performed is similar to the "," or "a" IRAF imexamine keys. It is circular aperture photometry, centered on the mouse location at the time the key is pressed, with a background annulus subtraction for the sky. The radius of the aperture is set with the region_size keyword (default to 5 pixels). The annulus size is also set to the width, and taken a distance of skyrad pixels from the center. The pixels used to calculate the enclosed flux are those whose centers fall inside the radius distance, in the same way that IRAF imexamine computes the flux.

These are the default parameters for aperture photometry. They live in a dictionary in the exam object:

```
The direct access:

viewer.exam.aperphot_pars= {"function":["aperphot",],
                "center":[True,"Center the object location using a Gaussian2D fit"],
                "width":[5,"Width of sky annulus in pixels"],
                "subsky":[True,"Subtract a sky background?"],
                "skyrad":[15,"Distance to start sky annulus is pixels"],
                "radius":[5,"Radius of aperture for star flux"],
                "zmag":[25.,"zeropoint for the magnitude calculation"],
                }
Using the convenience function:

In [1]: a.aimexam()
Out[2]:
{'center': [True, 'Center the object location using a 2d gaussian fit'],
'function': ['aper_phot'],
 'radius': [5, 'Radius of aperture for star flux'],
 'skyrad': [15, 'Distance to start sky annulus is pixels'],
 'subsky': [True, 'Subtract a sky background?'],
 'width': [5, 'Width of sky annulus in pixels'],
 'zmag': [25.0, 'zeropoint for the magnitude calculation']}
```

In order to change the width of the photometry aperture around the object you would do this::

```
viewer.set_plot_pars('a',"radius",10)
```

This is what the return looks like when you do photometry, where I've asked for photometry from the star above:

```
viewer.imexam()

xc=576.855763       yc=634.911425
x               y               radius          flux            mag(zpt=25.00) sky             fwhm
576.86          634.91          10              2191284.53      9.15            10998.89        5.58
```

xc = xcenter, yc=ycenter; these were found using a Gaussian2D fit centered on the pixel location of the mouse. You can turn the fit off by setting the "center" parameter to "False".

## 1.2 Gaussian1D, Moffat1D, MexicanHat1D profiles

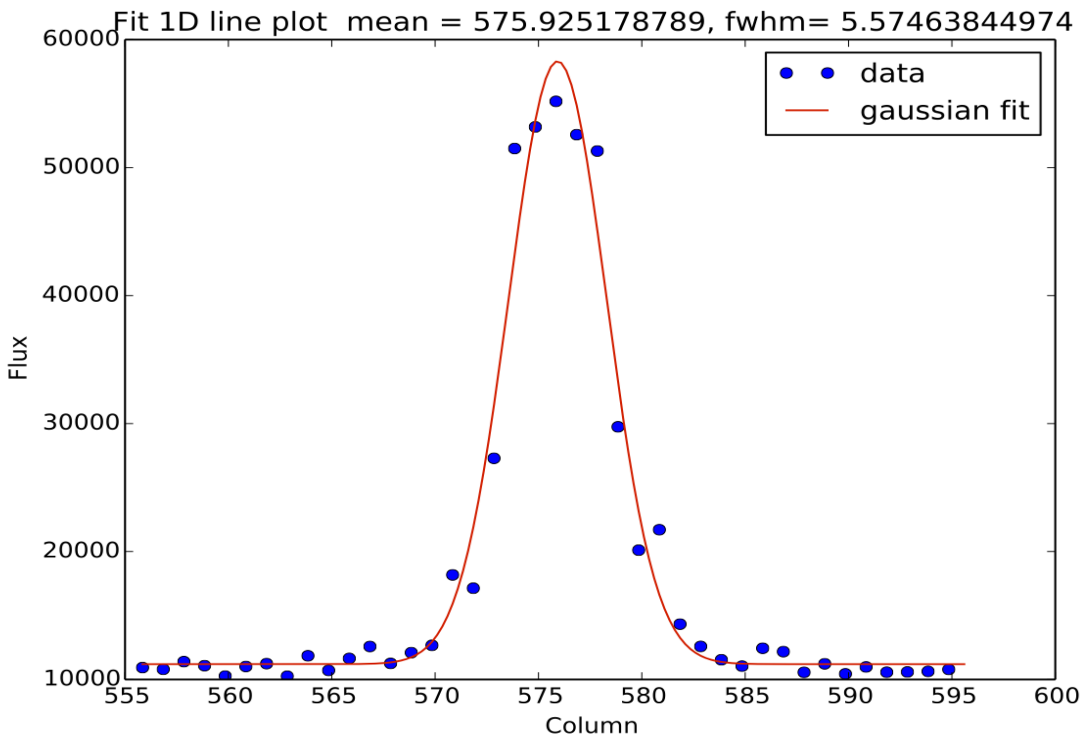
If you press the "j" or "k" keys, a 1D profile is fit to the data in either the line or column of the current pointer location. An option to use a Polynomial1D fit is also available, although not something of use for looking at stellar profiles. A plot of both the data and the fit + parameters is displayed. If the centering option is True, then the center of the flux is computed by fitting a 2d Gaussian to the data.

```
line_fit_pars={"function":["line_fit",],
               "func":["gaussian","function for fitting [see available]"],
               "title":["Fit 1D line plot","Title of the plot"],
               "xlabel":["Line","The string for the xaxis label"],
               "ylabel":["Flux","The string for the yaxis label"],
               "background":[False,"Solve for background? [bool]"],
               "width":[10.0,"Background  width in pixels"],
               "xorder":[0,"Background terms to fit, 0=median"],
               "rplot":[20.,"Plotting radius in pixels"],
               "pointmode":[True,"plot points instead of lines? [bool]"],
               "logx":[False,"log scale x-axis?"],
               "logy":[False,"log scale y-axis?"],
               "center":[True,"Recenter around the local max"],
               }
```

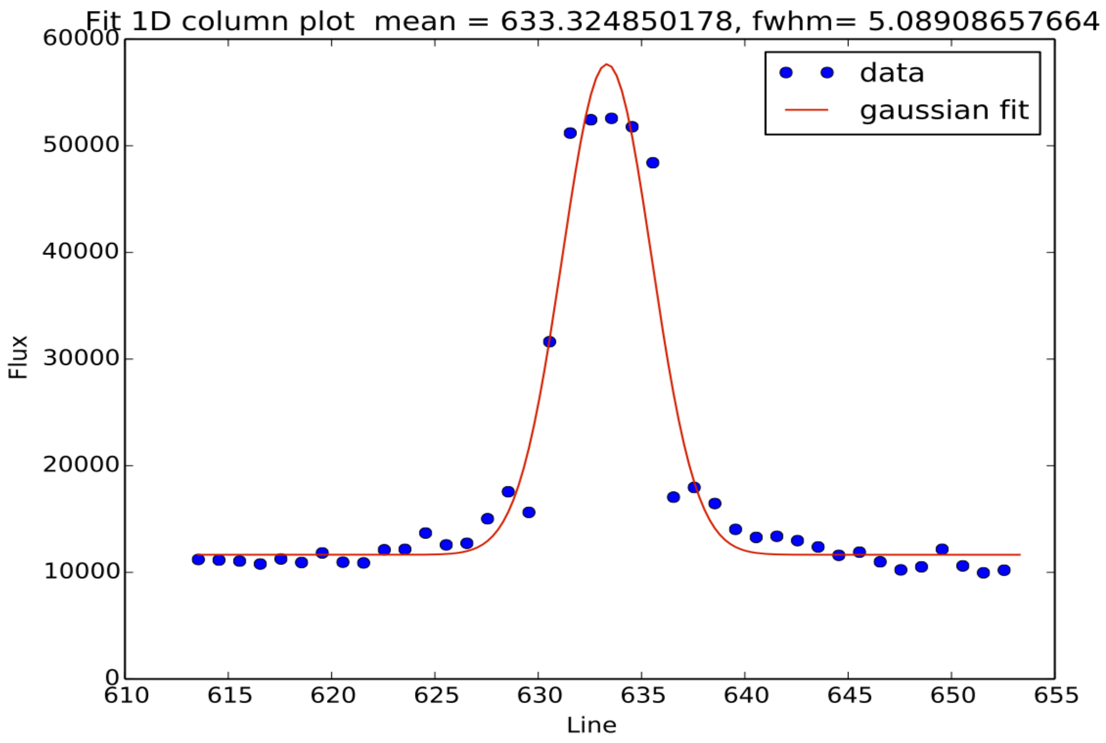
The column fit parameters are similar:

```
column_fit_pars={"function":["column_fit",],
                 "func":["Gaussian1D","function for fitting [see available]"],
                 "title":["Fit 1D column plot","Title of the plot"],
                 "xlabel":["Column","The string for the xaxis label"],
                 "ylabel":["Flux","The string for the yaxis label"],
                 "background":[False,"Solve for background? [bool]"],
                 "width":[10.0,"Background  width in pixels"],
                 "xorder":[0,"Background terms to fit, 0=median"],
                 "rplot":[20.,"Plotting radius in pixels"],
                 "pointmode":[True,"plot points instead of lines? [bool]"],
                 "logx":[False,"log scale x-axis?"],
                 "logy":[False,"log scale y-axis?"],
                 "center":[True,"Recenter around the local max"],
                 }
```

This is the resulting line fit:



and the corresponding column fit:



## 1.3 Square region statistics

If you press the "m" key, the pixel values around the pointer location are calculated inside a box which has a side equal to the region_size, defaulted to 5 pixels, and using the statistical function chosen.

The user can map the function to any reasonable numpy function, it's set to numpy.median by default:

```
report_stat_pars= {"function":["report_stat",],
                    "stat":["median","numpy stat name or describe for scipy.stats"],
                    "region_size":[5,"region size in pixels to use"],
                }


[573:578,629:634] median: 50632.000000
```

You can change the statistic reported by changing the "stat" parameter:

```
viewer.set_plot_pars('m',"stat","max")

[572:577,629:634] amax: 55271.000000
```

You can make a quick comparison of the max reported above with the line fit graph in the 1D gaussian profile example.

You can also choose to use the scipy.stats.describe function if you have scipy installed by changing the stat to "describe"; this will report the combined stats for the region::

```
pressed: m, report_stat
[551:556,653:658] describe:
```

```
nobs: 25
minamx: (0.51326549, 0.85604763)
mean 0.6851165890693665
variance: 0.00780616095289588
skew: 0.05719175934791565
kurtosis: -0.47930471400886976
```

## 1.4 Pixel Coordinates and Value

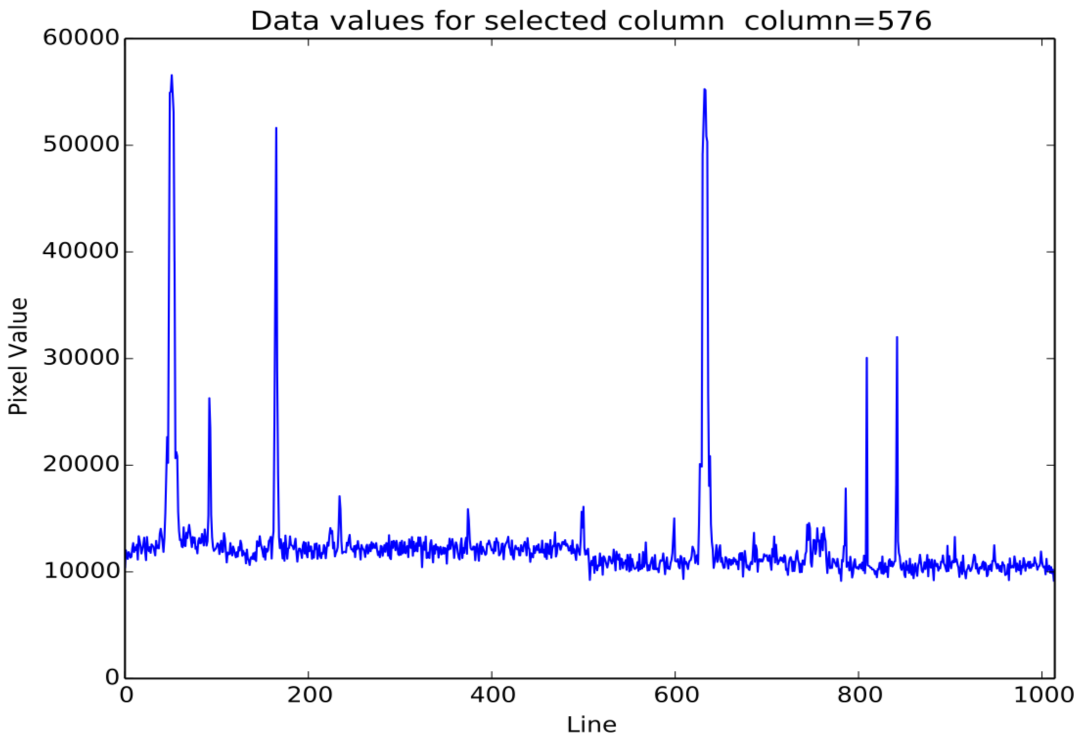
Hitting the 'x' or 'y' will return the x,y coordinate and pixel value under the mouse pointer.:

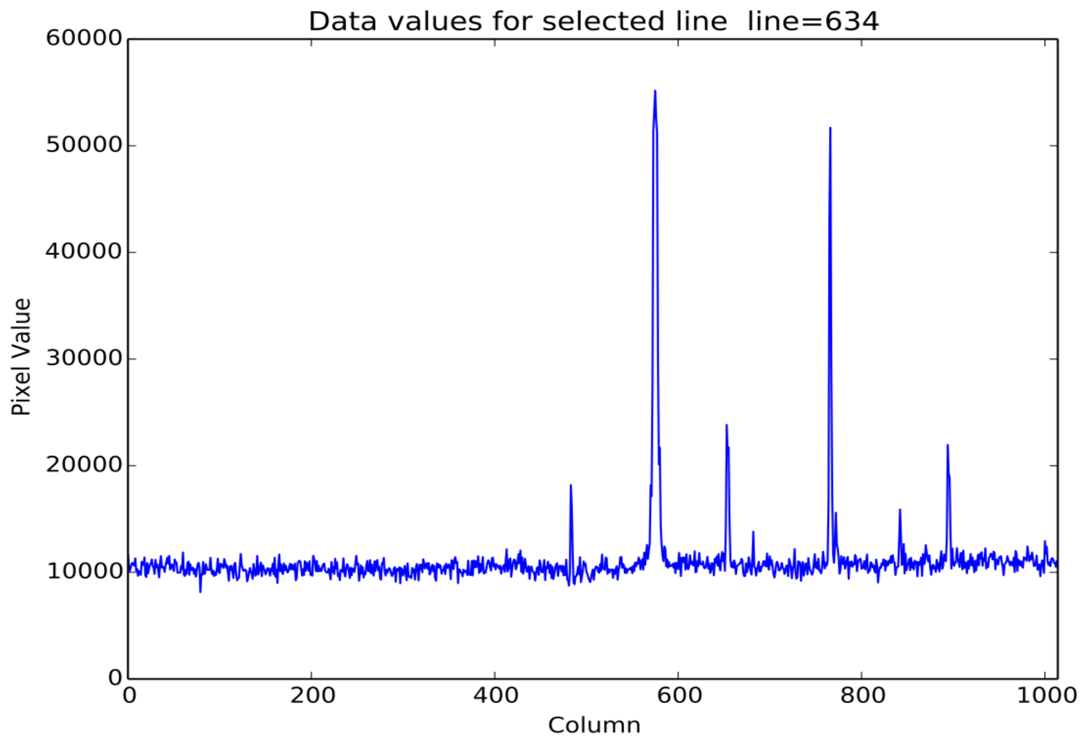
```
576.0 633.66667  55271.0
```

When not inside the imexam() loop, you can also set the location of the pointer using the wcs or pixel location you wish to view.

## 1.5 Line or Column plots

Pressing the "l" or "c" keys will display a plot of the points through either the line or column closest to the cursor location.





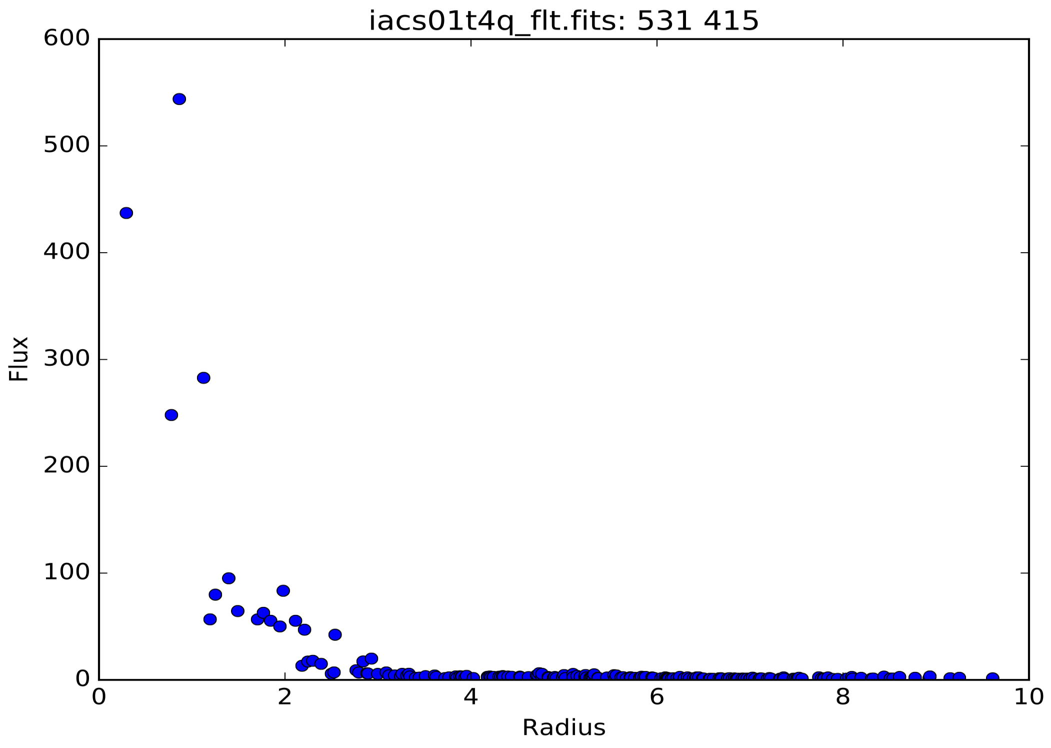
## 1.6 Radial Profile Plot

Pressing the "r" key displays a radial profile plot for the flux around the current pointer location. If centering is on, the center is computed close to the star using a Gaussian2D fit. The default plot uses every pixel

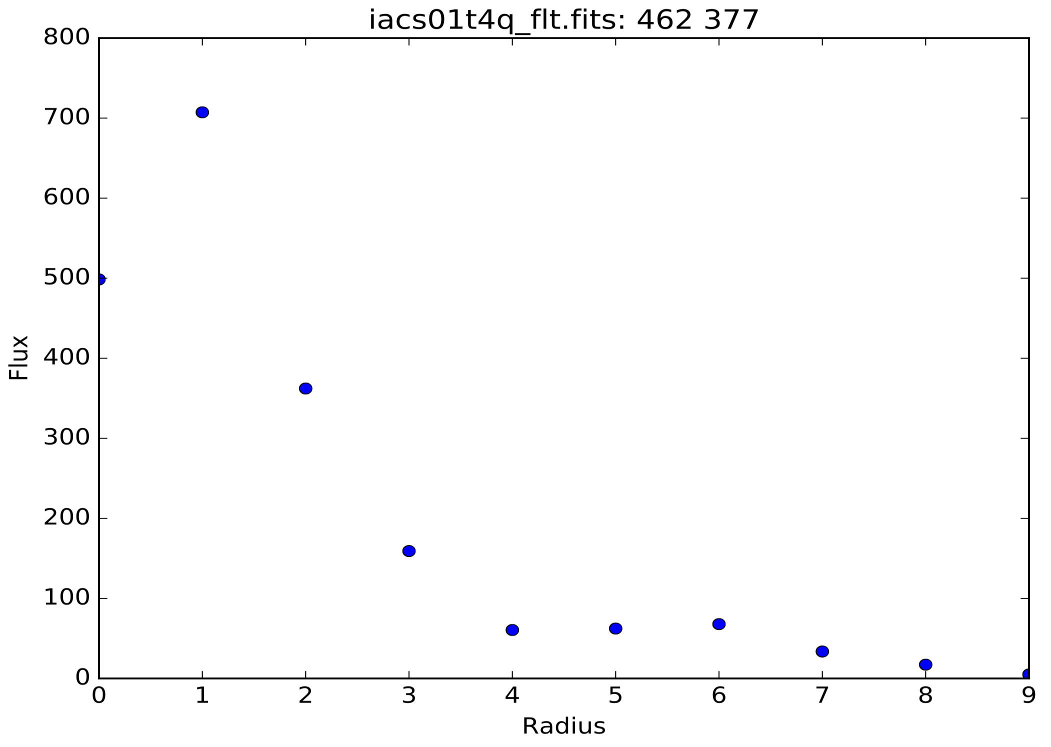
The available parameters are

```
radial_profile_pars = {"function": ["radial_profile_plot", ],
            "title": ["Radial Profile", "Title of the plot"],
            "xlabel": ["Radius", "The string for the xaxis label"],
            "ylabel": ["Summed Pixel Value", "The string for the yaxis label"],
            'pixels': [True, 'Plot all pixels at each radius? (False bins the data)']
            "fitplot": [False,"Overplot profile fit?"],
            "fittype":["Gaussian1D","Profile type to fit (gaussian)"],
            "center": [True, "Solve for center using 2d Gaussian? [bool]"],
            "background": [True, "Subtract background? [bool]"],
            "skyrad": [10., "Background inner radius in pixels, from center of object"],
            "width": [5., "Background annulus width in pixels"],
            "magzero": [25., "magnitude zero point"],
            "rplot": [8., "Plotting radius in pixels"],
            "pointmode": [True, "plot points instead of lines? [bool]"],
            "marker": ["o", "The marker character to use, matplotlib style"],
            "minflux": [0., "only measure flux above this value"],
            "getdata": [True, "return the plotted data values"]
            }
```

Radial profile plot for all pixels around the location:



Radial profile plot for all pixels, binned to integer radii:

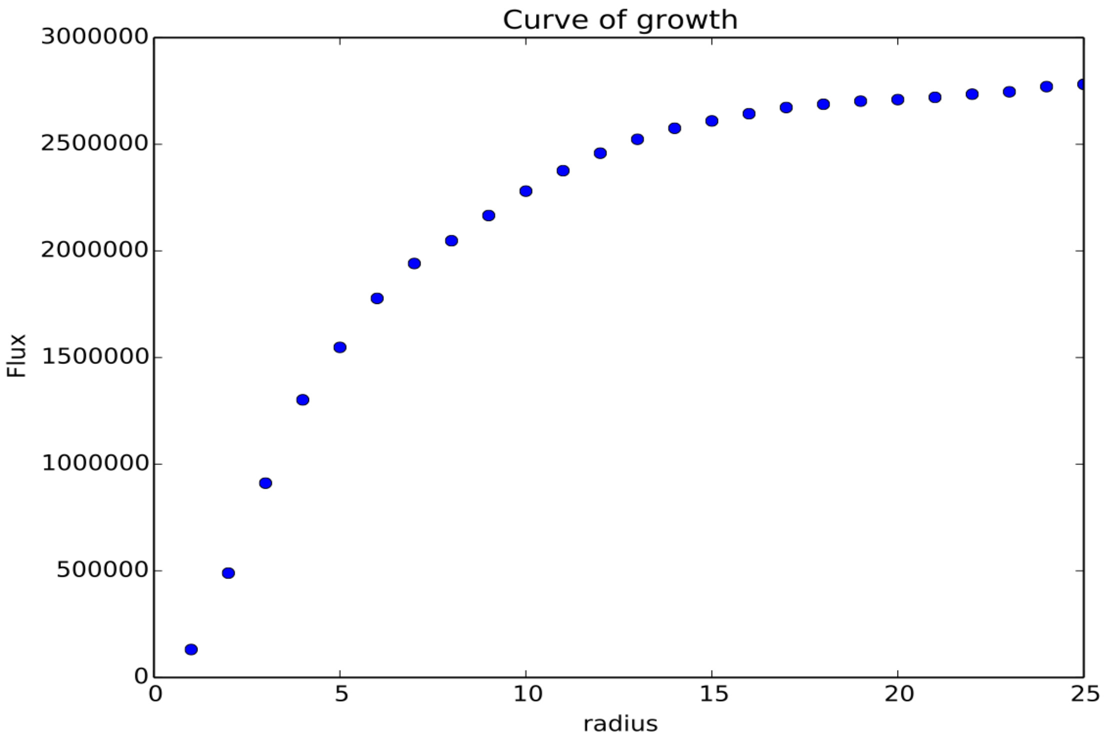


## 1.7 Curve of Growth plot

Pressing the "r" key displays a curve of growth for the flux around the current pointer location in successively larger radii. If centering is on, the center is computed close to the star using a 2d gaussian fit.

The available parameters are

```
curve_of_growth_pars={"function":["curve_of_growth_plot",],
                      "title":["Curve of Growth","Title of the plot"],
                      "xlabel":["radius","The string for the xaxis label"],
                      "ylabel":["Flux","The string for the yaxis label"],
                      "center":[True,"Solve for center using 2d Gaussian? [bool]"],
                      "background":[True,"Fit and subtract background? [bool]"],
                      "buffer":[25.,"Background inner radius in pixels,from center of star"],
                      "width":[5.,"Background annulus width in pixels"],
                      "magzero":[25.,"magnitude zero point"],
                      "rplot":[8.,"Plotting radius in pixels"],
                      "pointmode":[True,"plot points instead of lines? [bool]"],
                      "marker":["o","The marker character to use, matplotlib style"],
                      "logx":[False,"log scale x-axis?"],
                      "logy":[False,"log scale y-axis?"],
                      "minflux":[0., "only measure flux above this value"],
                      }
```



Returned to the screen is the data information from the plot, the (x,y) location of the center, followed by the radius and corresponding flux which was measured:

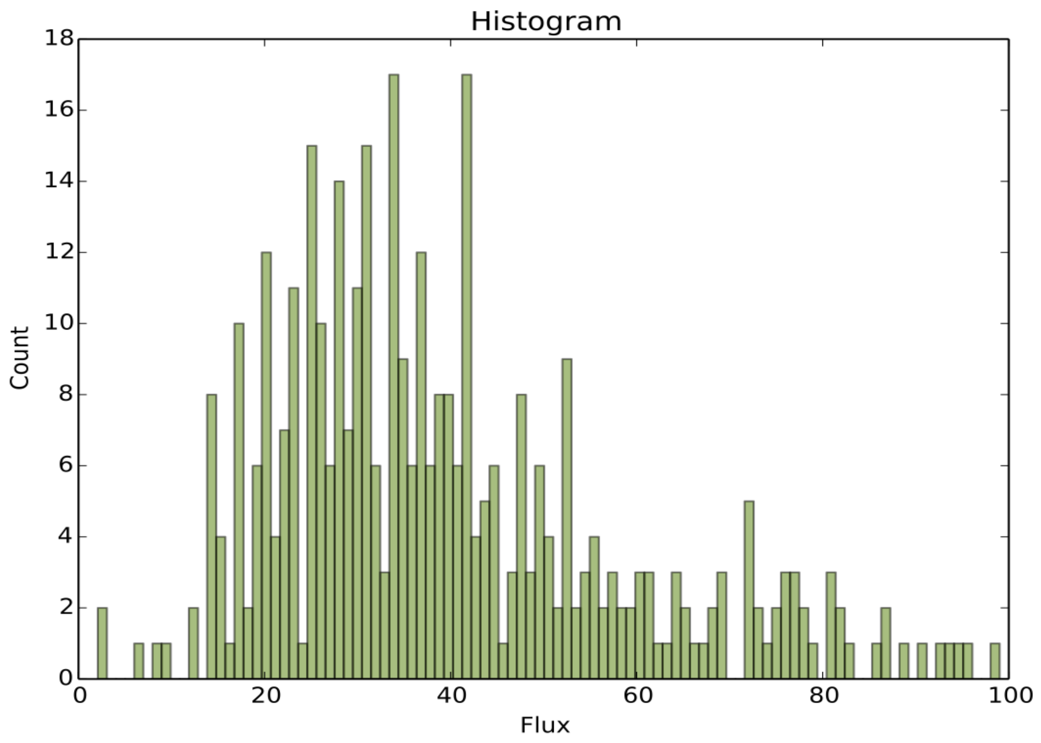
```
viewer.set_plot_pars('g',"rplot",25)  #set the default radius larger

xc=577.242311       yc=634.578361
```

```
at (x,y)=577,634
radii:[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25]
flux:[131192.03694247041, 489485.48536408512, 911376.50226695999, 1301726.7189847208, 1547865.
↪8684735354, 1777547.7859571185, 1940955.1267221647, 2047700.7156964755, 2165971.1952809561, 2280391.
↪5901085823, 2376090.3555588746, 2458370.0006153183, 2523384.2243051622, 2575208.3657517368, 2609309.
↪6524876151, 2643279.3635597304, 2672443.1546003688, 2687659.5178374872, 2702128.5513395425, 2709501.
↪1520242952, 2720134.8632924128, 2734777.3482598308, 2746056.5231984705, 2770352.0070485324, 2781242.
↪3299104609]
```

## 1.8 Histogram Plots

Pressing the "h" key will display a histogram of pixel values around the pixel location under the mouse pointer.

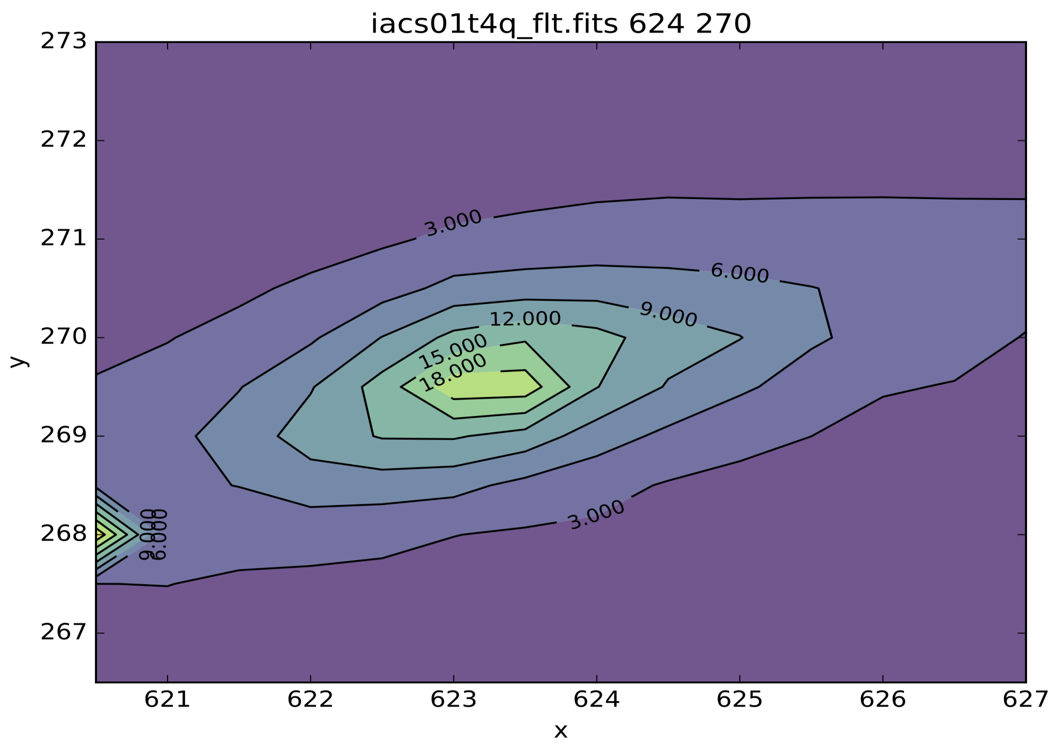
```
histogram_pars={"function":["histogram",],
                "title":["Histogram","Title of the plot"],
                "xlabel":["Flux (bin)","The string for the xaxis label"],
                "ylabel":["Count","The string for the yaxis label"],
                "ncolumns":[21,"Number of columns"],
                "nlines":[21,"Number of lines"],
                "nbins":[100,"Number of bins"],
                "z1":[None,"Minimum histogram intensity"],
                "z2":[100,"Maximum histogram intensity"],
                "pointmode":[True,"plot points instead of lines? [bool]"],
                "marker":["o","The marker character to use, matplotlib style"],
                "logx":[False,"log scale x-axis?"],
                "logy":[False,"log scale y-axis?"],
                }
```



## 1.9 Contour Plots

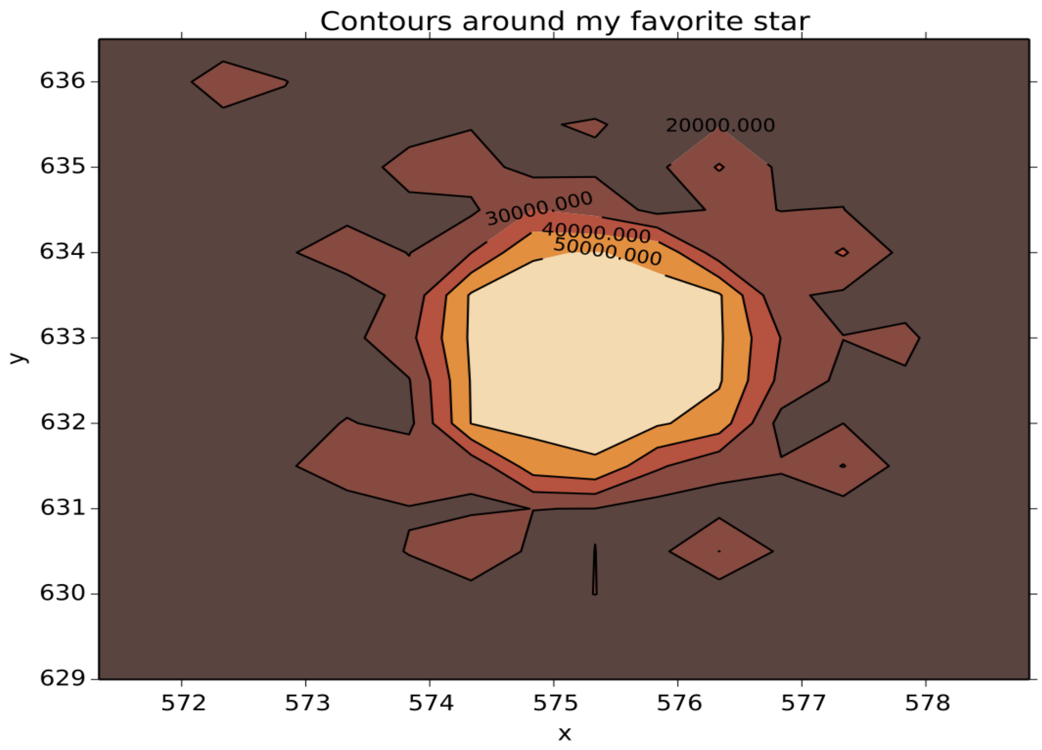
Pressing the “e” key will display a contour plot around the clicked pixel location.

```
contour_pars={"function":["contour",],
                   "title":["Contour plot in region around pixel location","Title of the plot"],
                   "xlabel":["x","The string for the xaxis label"],
                   "ylabel":["y","The string for the yaxis label"],
                   "ncolumns":[15,"Number of columns"],
                   "nlines":[15,"Number of lines"],
                   "floor":[None,"Minimum value to be contoured"],
                   "ceiling":[None,"Maximum value to be contoured"],
                   "ncontours":[8,"Number of contours to be drawn"],
                   "linestyle":["--","matplotlib linestyle"],
                   "label":[True,"Label major contours with their values? [bool]"],
                   "cmap":["viridis","Colormap (matplotlib style) for image"],
                   }
```



Here's what it looks like if we change some of the default parameters:

```
viewer.set_plot_pars('e', "cmap", "gist_heat")
viewer.set_plot_pars('e', "title", "Contours around my favorite star")
viewer.set_plot_pars('e', "ncontours", 4)
viewer.set_plot_pars('e', "floor", 0)
```

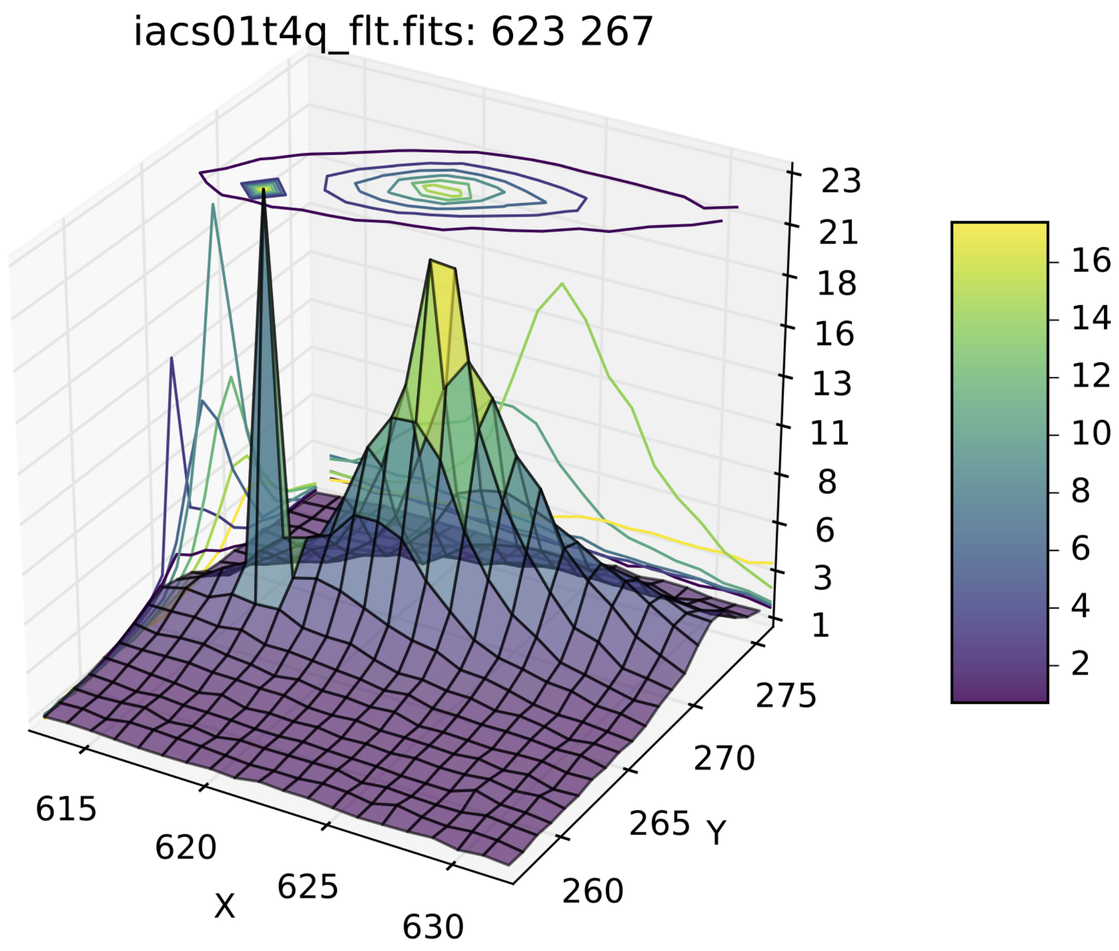


**Note:** You can use any of the matplotlib standard cmaps, see the following link for more information: http://matplotlib.org/api/pyplot_summary.html?highlight=colormaps#matplotlib.pyplot.colormaps

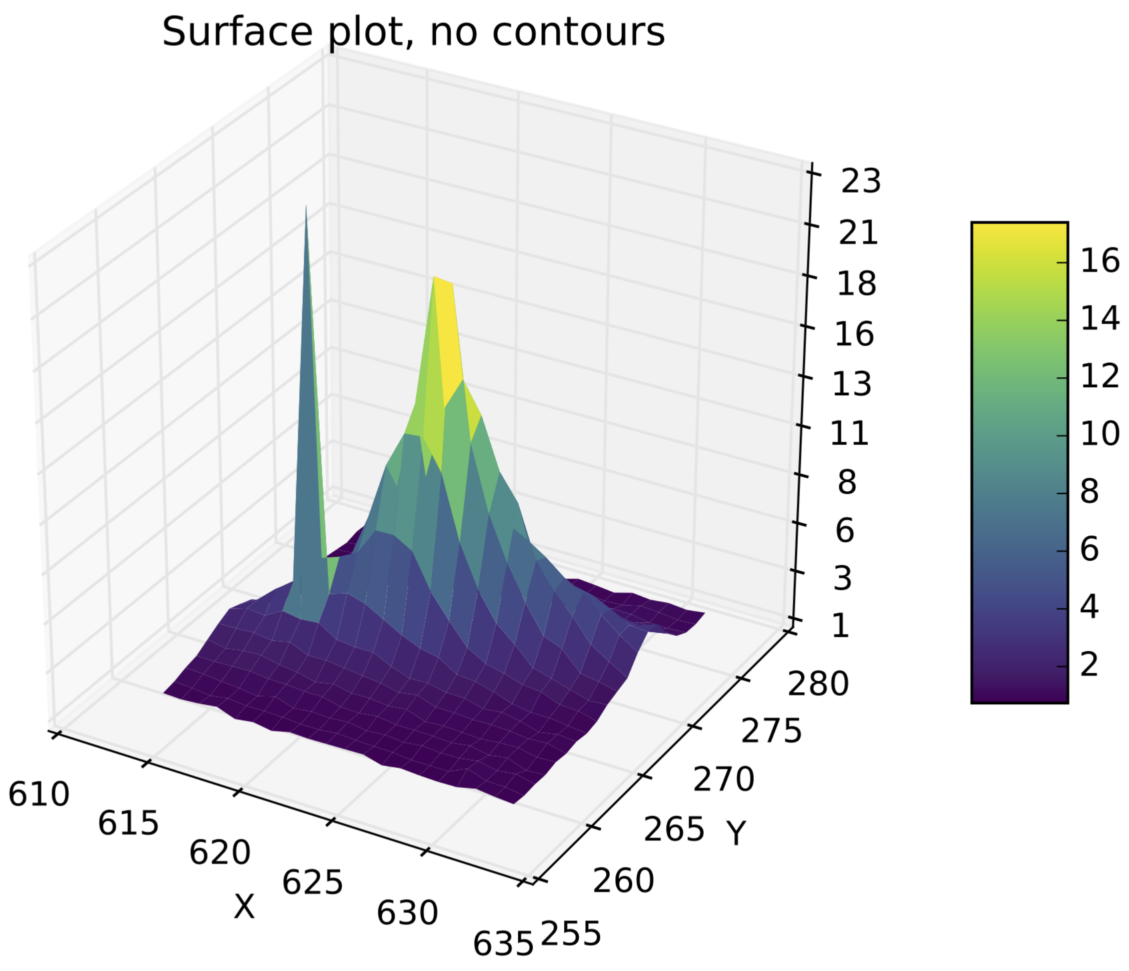
## 1.10 Surface Plots

Pressing the "s" key will display a 3D surface plot of pixel values around the mouse pointer location with the default parameters:

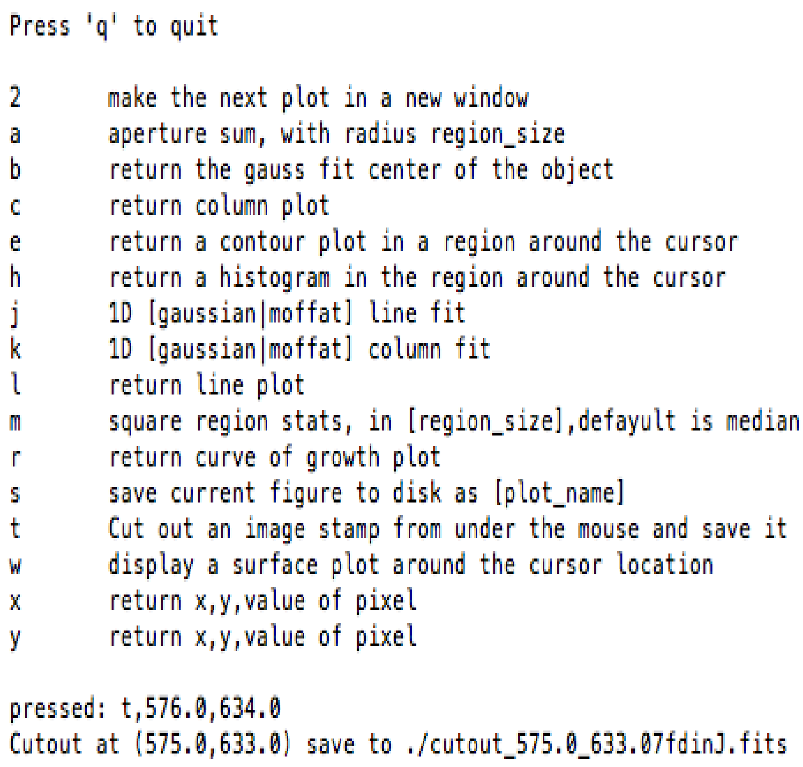
```
surface_pars = {"function": ["surface", ],
                "title": [None, "Title of the plot"],
                "xlabel": ["X", "The string for the xaxis label"],
                "ylabel": ["Y", "The string for the yaxis label"],
                "zlabel": [None, "Label for zaxis"],
                "ncolumns": [10, "Number of columns"],
                "nlines": [10, "Number of lines"],
                "azim": [None, "azimuthal viewing angle in degrees"],
                "floor": [None, "Minimum value to be contoured"],
                "ceiling": [None, "Maximum value to be contoured"],
                "stride": [1, "step size, higher vals will have less contour"],
                "cmap": ["viridis", "colormap (matplotlib) for display"],
                "fancy": [True, "This aint your grandpas iraf"],
                }
```



Or, with the contours turned off (by setting fancy to False) and changing the title:


Surface plot, no contours
X
Y
610
615
620
625
630
635
255
260
265
270
275
280
1
3
6
8
11
13
16
18
21
23
2
4
6
8
10
12
14
16

## 1.11 Cutout a Simple FITS Image

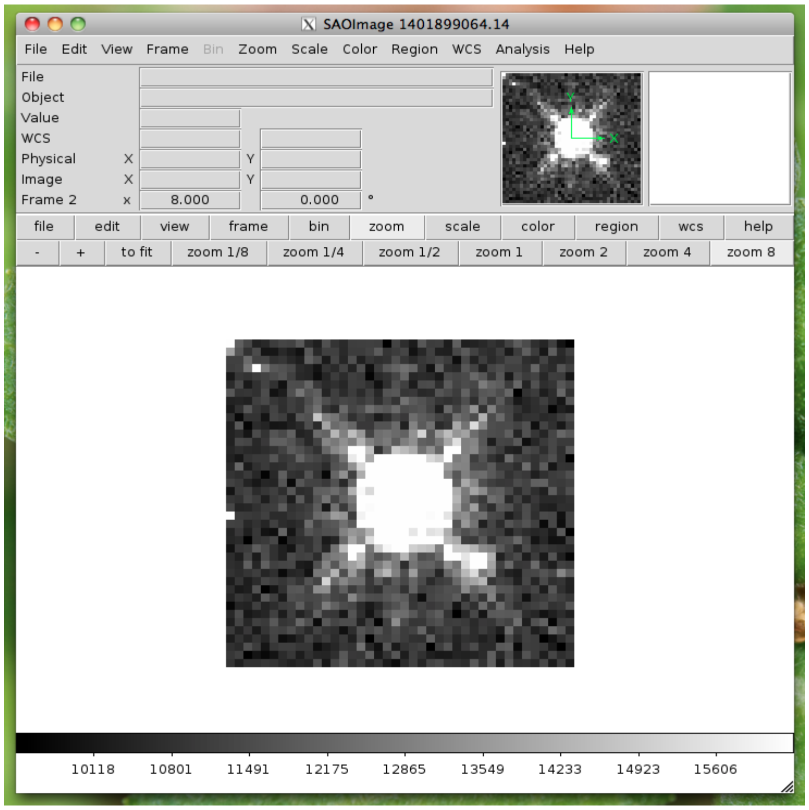


```
Press 'q' to quit

2       make the next plot in a new window
a       aperture sum, with radius region_size
b       return the gauss fit center of the object
c       return column plot
e       return a contour plot in a region around the cursor
h       return a histogram in the region around the cursor
j       1D [gaussian|moffat] line fit
k       1D [gaussian|moffat] column fit
l       return line plot
m       square region stats, in [region_size],defayult is median
r       return curve of growth plot
s       save current figure to disk as [plot_name]
t       Cut out an image stamp from under the mouse and save it
w       display a surface plot around the cursor location
x       return x,y,value of pixel
y       return x,y,value of pixel

pressed: t,576.0,634.0
Cutout at (575.0,633.0) save to ./cutout_575.0_633.07fdinJ.fits
```

Okay, I went to the star I like and pressed “t”. Let’s verify that we got what we wanted, it should be a cutout centered on the star that we’ve used in all the examples here:

```
image=fits.open('cutout_575.0_633.07fdinJ.fits')
viewer.frame(2)
viewer.view(image)
```

And the resulting frame view?


SAOImage 1401899064.14
File Edit View Frame Bin Zoom Scale Color Region WCS Analysis Help
File
Object
Value
WCS
Physical X Y
Image X Y
Frame 2 x 8.000 0.000 °
file edit view frame bin zoom scale color region wcs help
- + to fit zoom 1/8 zoom 1/4 zoom 1/2 zoom 1 zoom 2 zoom 4 zoom 8
10118 10801 11491 12175 12865 13549 14233 14923 15606

Sweet. Because this is a often used function I've made it a part of the standard selection set. If you wish to use the astropy 2D cutout method, you can create your own function which will also pass in the WCS object for the data so that the cutout retains it's WCS information.

## 1.12 User Specified functions

Users may code their own functions and bind them to keys by registering them with the `imexam` dictionary through the register method that lives in the exam object. As long as a unique key is provided, the new binding will be added to the dictionary of imexamine functions. The new functions do not have to have default dictionaries associated with them. The binding is only good for the current object, new instantiations of `imexam.connect()` will not have the new function unless the user specifically registers them.

Here's all the code for a function which saves the cursor location to a file called 'test.list' when the user presses the 'p' key:

```
def save_to_file(self,x,y,data):
    """Save the cursor location only to a file"""
    if data is None:
      data = self._data
    with open('test.list','a') as ofile:
      ofile.write("{0}\t{1}\n".format(x,y))
    print("Saved star to ",'test.list')
```

Now, import that into your python session, file, or here I'll just copy paste the definition to the session. This is an important step because the function reference is what you are going to send to the registration method. The registration method wants you to supply a dictionary which contains the key you want to assign that function to during the imexam() loop, and a tuple with the function name and description:

```
my_dict = {'p': (save_to_file, 'Save cursor location to file')}

viewer.exam.register(my_dict)
User function: save_to_file added to imexam options with key p
```

Okay, so let's try out our new function! We should be able to see it in the list of available options.

```
In [18]: a.imexam()

Press 'q' to quit

2       Make the next plot in a new window
a       Aperture sum, with radius region_size
b       Return the 2D gauss fit center of the object
c       Return column plot
e       Return a contour plot in a region around the cursor
g       Return curve of growth plot
h       Return a histogram in the region around the cursor
j       1D [Gaussian1D default] line fit
k       1D [Gaussian1D default] column fit
l       Return line plot
m       Square region stats, in [region_size],default is median
p       Save cursor location to file
r       Return the radial profile plot
s       Save current figure to disk as [plot_name]
t       Make a fits image cutout using pointer location
w       Display a surface plot around the cursor location
x       Return x,y,value of pixel
y       Return x,y,value of pixel

Current image /Users/sosey/test_images/iacs01t4q_flt.fits
pressed: p, save_to_file
Saved star to  test.list

In [19]: !more test.list
463.0   376.75
```

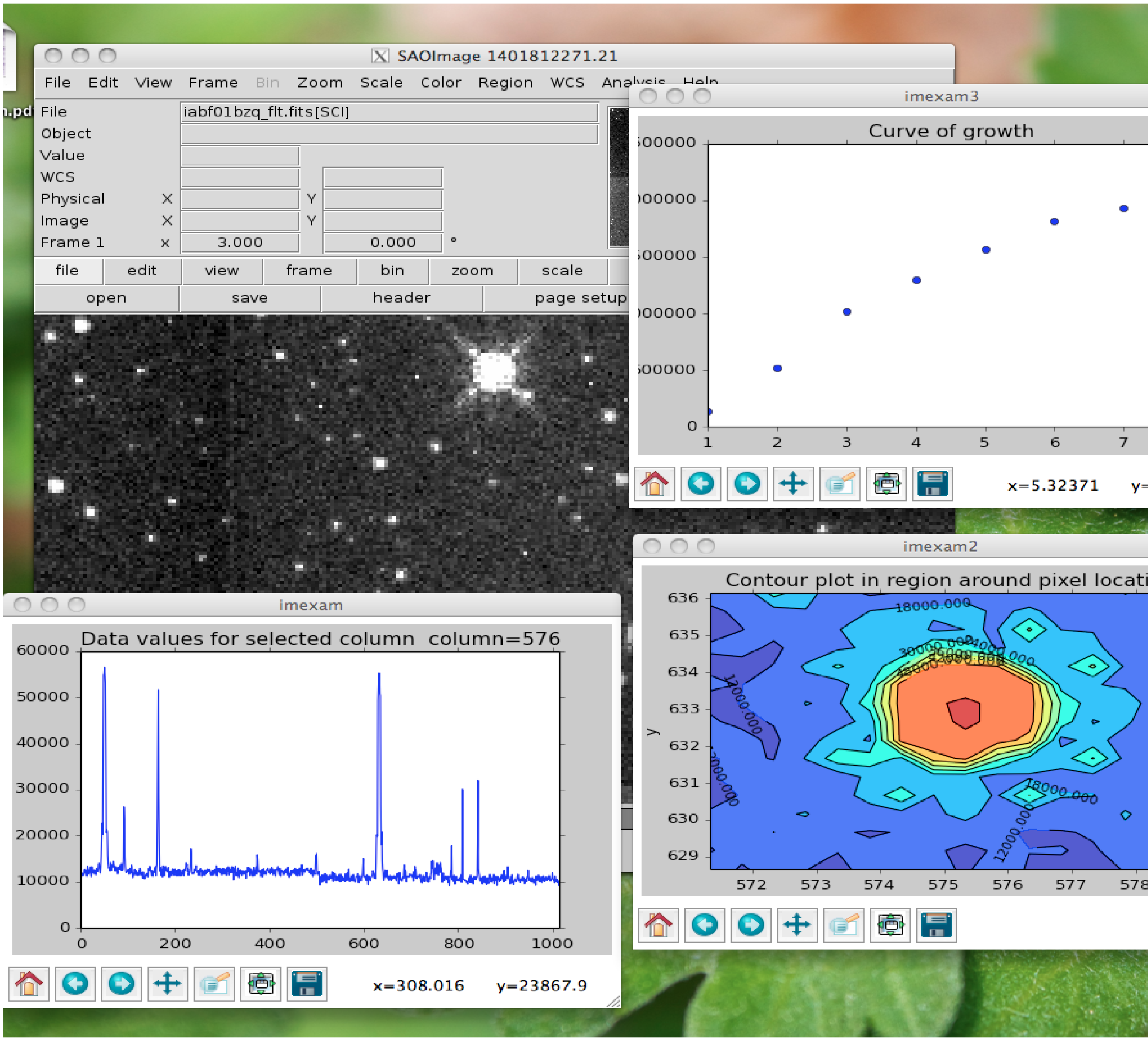
## 1.13 Plot Multiple Windows

During a single viewer.imexam() session, you can choose to send your plots to multiple windows. Each window may only be used once, but if you would like to plot multiple things to compare, either the same plots for multiple objects or multiple types of plots for a single object, you can press the "2" key. This will save the current plotting window on your desktop and send the next plot to a new window. The plotting windows will be closed when you exit the imexam loop, so be sure to use the "s" key to save a quick copy of any plots you'd like to save for refernce. Here's what that might look like:

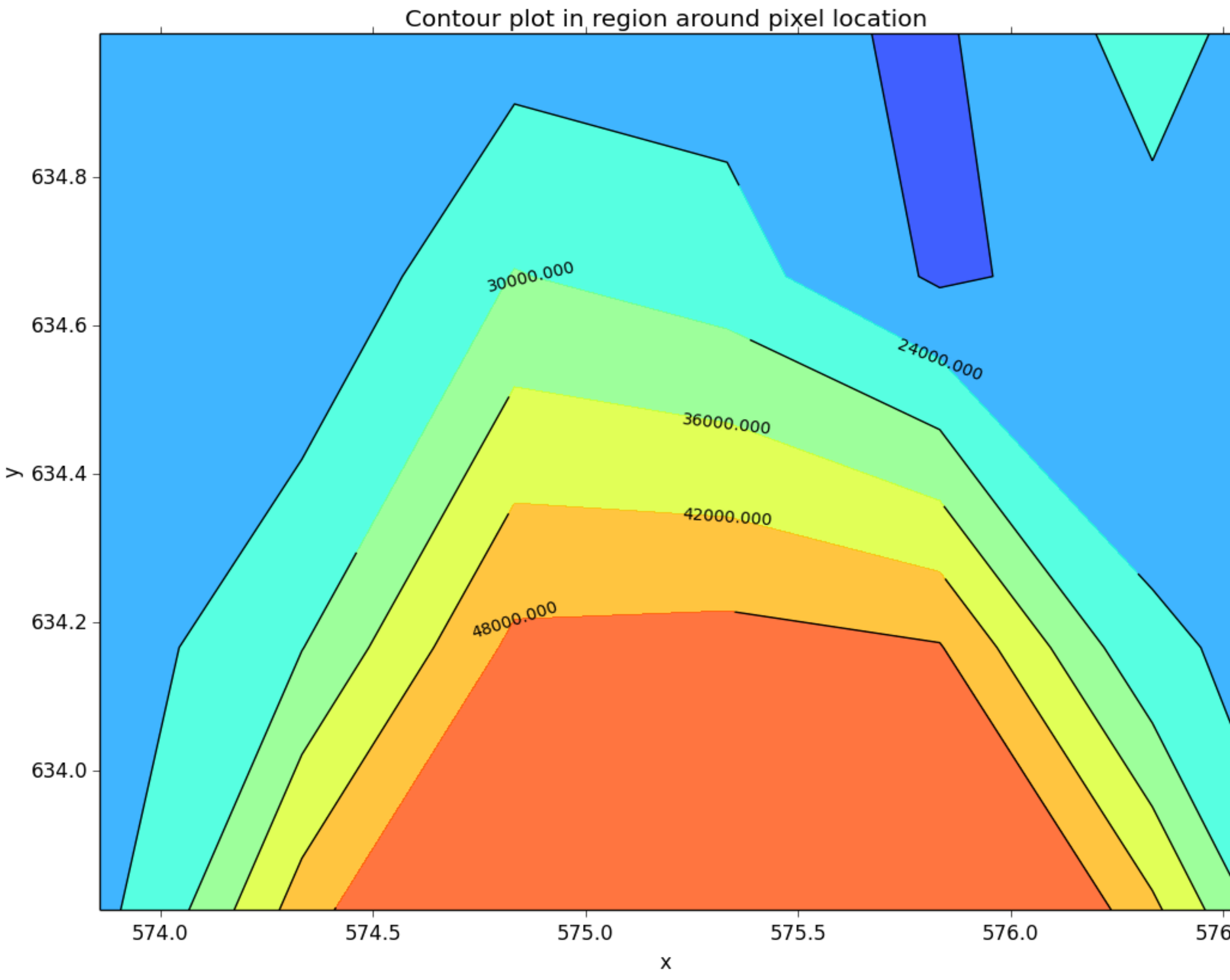
```
#run aperture photometry("a"):

xc=576.522433       yc=634.578085
x              y              radius         flux           mag(zpt=25.00) sky            fwhm
576.52         634.58         5              1560462.68     9.52           10996.52       5.58

#make a column plot ("c")

#direct to a new window and make a contour plot ("e")
```

```
Plots now directed towards imexam2

#direct to a new window and make a curve of growth ("r")
Plots now directed towards imexam3

#the resulting curve of growth information on the screen
xc=576.855763       yc=634.911425

at (x,y)=576,634
radii:[1 2 3 4 5 6 7 8]
flux:[134294.19631173008, 521208.13904411002, 1017231.0442446949, 1297592.7076232315, 1568629.
↪6771239617, 1813434.3810552177, 1935335.7549474821, 2049080.846300941]
```

This is what the workspace could look like with DS9 as the viewer:


SAOImage 1401812271.21
File Edit View Frame Bin Zoom Scale Color Region WCS Analysis Help
File iabf01bzq_flt.fits[SCI]
Object
Value
WCS
Physical X Y
Image X Y
Frame 1 x 3.000 0.000 °
file edit view frame bin zoom scale
open save header page setup
imexam3
Curve of growth
x=5.32371
imexam2
Contour plot in region around pixel locati
imexam
Data values for selected column column=576
x=308.016 y=23867.9

As an aside, you can use the GUI tools on the bottom of the plot windows to move around the displayed data, such as zooming in and out, as shown below for the contour plot, which was also saved using the GUI save button:


Contour plot in region around pixel location
30000.000
24000.000
36000.000
42000.000
48000.000
634.8
634.6
634.4
634.2
634.0
y
574.0
574.5
575.0
575.5
576.0
x

CHAPTER 2

# imexam User Methods

These are methods particular to the imexam package which are meant to aid users in their image analysis. They are called from the main object you created with imexam.connect().

At the top library level, the follow commands are available::

```
imexam.connect(): connect to a viewer and return a control object
imexam.display_help(): Takes you to the help documents for your installed version of imexam
imexam.defpars: contains the default plotting function dictionaries
imexam.imexamine: this class contains the plotting functions and can be instantiated by itself
imexam.set_logging(): set the logging parameters for your session.
```

Each object has access to it's own logging which can be edited using viewer.setlog() The following will also be available for those not on a Windows system, where the XPA and DS9 are installed::

```
imexam.display_xpa_help(): Takes you to the XPA help page for DS9
imexam.list_active_ds9(): returns a dictionary of available DS9 sessions for connection
```

You can always get the commands available to your local viewer by asking the control object for them directly. If you called your control object "viewer" then the following example will return the list::

```
viewer.show_window_commands()  # will return a list of available commands
```

Not all viewers have all commands implemented, commands which are available but not yet fully implemented should return an error to that affect.

**alignwcs(on=True):**
Align the images in the viewer using the WCS in their headers

```
viewer.alignwcs()
```

**blink(blink=True, interval=None):**
For viewers with multiple frames, blink the images

**clear_contour():**
Clear contours from the screen

**close():**
close the image viewing window and end the connection.

```
viewer.close()
```

**cmap(color=None, load=None, invert=False, save=False,filename='colormap.ds9'):**
Set the colormap for the window

**colorbar(on=True):**
Turn the colorbar in the window on or off

**contour(on=True, construct=True):**
Show contours in the window

**crosshair(x=None, y=None, coordsys="physical", skyframe="wcs", skyformat="fk5", match=False, lock=False):**

Control the position of the crosshair in the current frame

**cursor(x=None, y=None):**
Move the cursor in the window to the specified pixel location

**disp_header():**
Display the image header

**frame(n=None):**
Convenience function to change or report the frame

**get_data():**
Return a numpy array of the data displayed in the current frame

**get_filename():**
Return the filename for the data in the current window

```
In [1]: viewer.get_data_filename()
Out[2]: '/Users/sosey/ssb/imexam/iabf01bzq_flt.fits'
```

**get_frame_info():**
Return more explicit information about the data displayed in the current frame. A dictionary of the information is returned.

```
In [1]: viewer.get_frame_info()

    {'extname': 'SCI',
    'extver': 1,
    'filename': '/Users/sosey/ssb/imexam/iabf01bzq_flt.fits',
    'iscube': False,
    'mef': True,
    'naxis': 0,
    'numaxis': 2,
    'user_array': None}
```

**get_header():**
Return the header of the dataset in the current frame

**get_image():**
Return the full image object for the data in the current frame

**get_slice_info():**
Return the slice tuple for the image currently displayed

**get_viewer_info():**
Return a dictionary which contains information about all frames which have data loaded. This could be useful to users who are scripting an analysis for polling what items are available, how many frames or displayed, what type of data is hanging around, etc ...

```
In [1]: viewer.get_viewer_info()

{'1': {'extname': 'SCI',
  'extver': 1,
  'filename': '/Users/sosey/ssb/imexam/iabf01bzq_flt.fits',
  'iscube': False,
  'mef': True,
  'naxis': 0,
  'numaxis': 2,
  'user_array': None}}
```

**grab():**
Take a snapshop of the image view

**grid(on=True, param=False):**
Turn a grid on and off in the window

**hideme():**
Reduce the precedence of the window

**iscube():**
Boolean return if the image is multidimensional cube

**load_fits(fname="", extver=1, extname=None):**
Load a fits image into the current frame

**load_mef_as_cube(filename=None):**
Load a Mult-Extension-Fits image into one frame as an image cube

**load_mef_as_multi(filename=None):**
Load a Mult-Extension-Fits image into multiple frames

**load_region(filename):**
Load regions from a file which uses standard formatting

**load_rgb(red, green, blue, scale=False, lockwcs=False):**
Load three images into an RGB colored frame

**make_region(infile,doLabels=False):**
Make an input reg file which contains rows with "x,y,comment" into a region file that the DS9 viewer recognizes.

**infile: str**
input filename

**labels: bool**
add labels to the regions

**header: int**
number of header lines in text file to skip

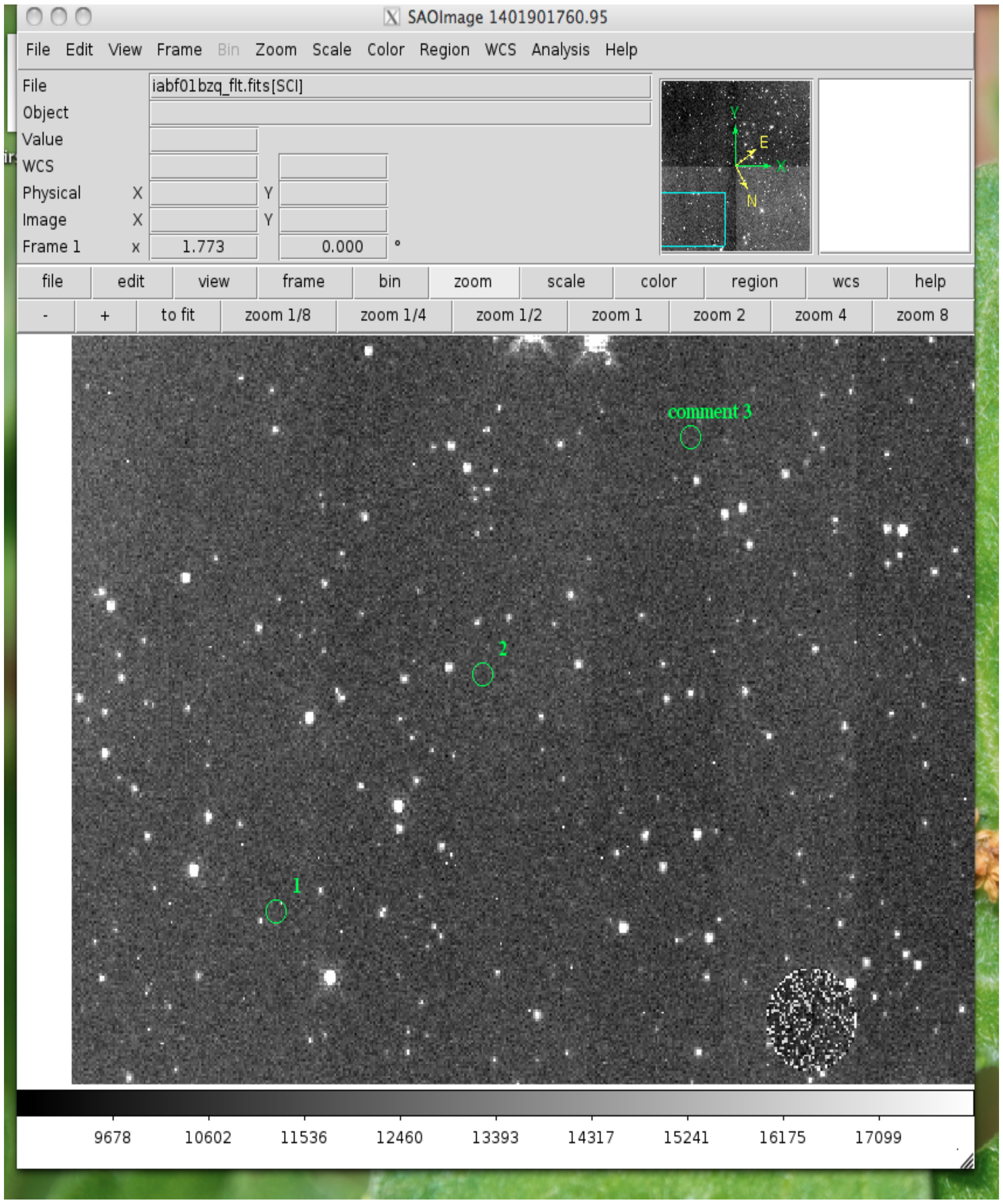
**textoff: int**
offset in pixels for labels

**rtype: str**
region type, one of the acceptable DS9 regions

**size: int**
size of the region type

```
Here's what the input file 'test' looks like:

100,100, 1
200,200, 2
300,300, comment 3


viewer.make_region('test',labels=True)

And the output region file:

image; circle(100,100,5)
image;text(110.0,110.0{ 1 })# font="time 12 bold"
image; circle(200,200,5)
image;text(210.0,210.0{ 2 })# font="time 12 bold"
image; circle(300,300,5)
image;text(310.0,310.0{ comment 3 })# font="time 12 bold"
```

Now let's load the region file into our image:


SAOImage 1401901760.95
File Edit View Frame Bin Zoom Scale Color Region WCS Analysis Help
File iabf01bzq_flt.fits[SCI]
Object
Value
WCS
Physical X Y
Image X Y
Frame 1 x 1.773 0.000 °
file edit view frame bin zoom scale color region wcs help
- + to fit zoom 1/8 zoom 1/4 zoom 1/2 zoom 1 zoom 2 zoom 4 zoom 8
comment 3
2
1
9678 10602 11536 12460 13393 14317 15241 16175 17099

**mark_region_from_array(input_points,rtype="circle",ptype="image",textoff=10,size=5):**
mark regions on the display given a list of tuples, a single tuple, or a string, where each object has x,y,comment specified

**input_points: an iterable**
contains: (x,y,comment) tuples

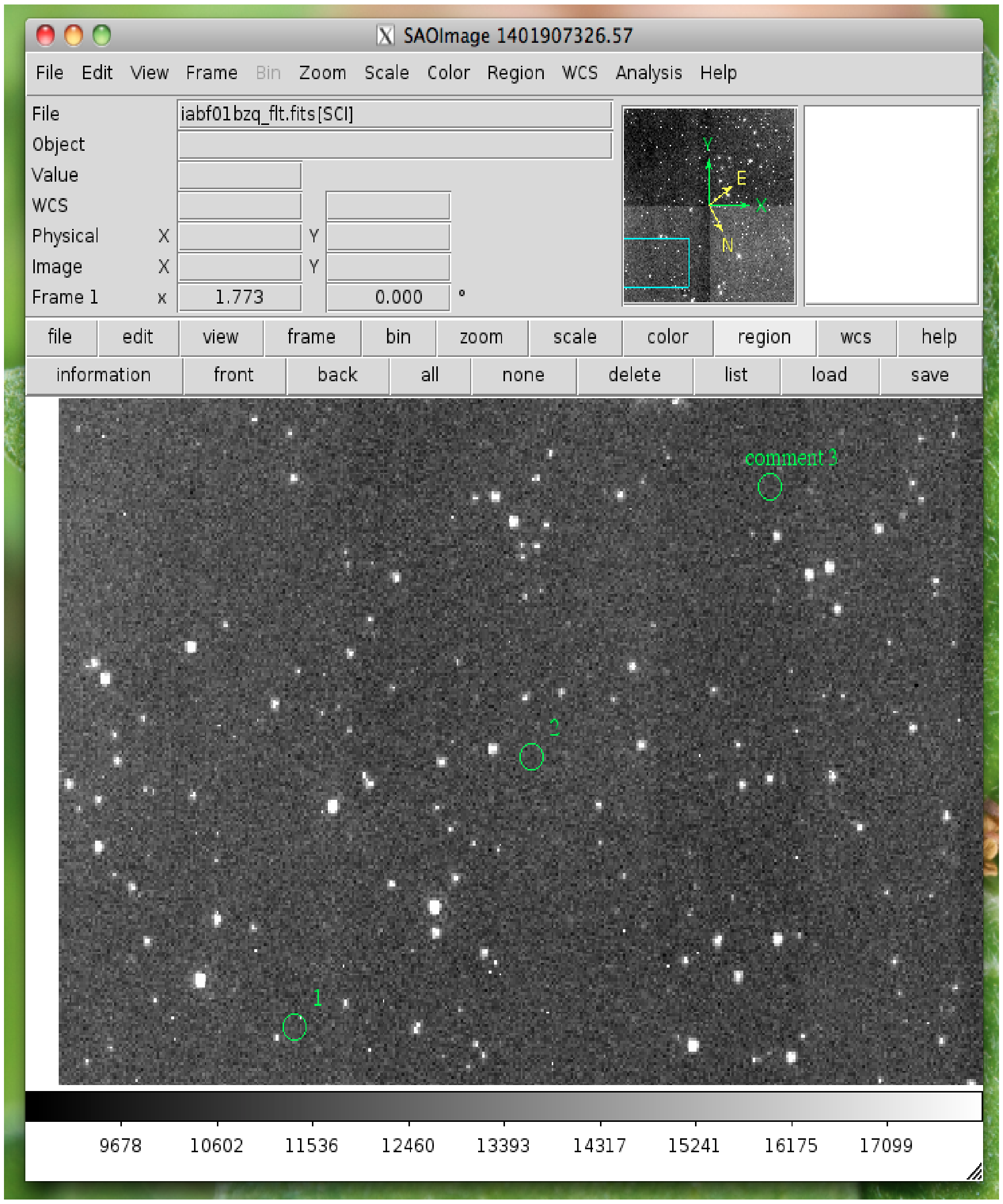
**ptype: string**
the reference system for the point locations, image|physical|fk5

**rtype: string**
the matplotlib style marker type to display

**size: int**
the size of the region marker

**textoff: string**
the offset for the comment text, if comment is empty it will not show

```
locations=list()
locations.append( (100,100,1) )
locations.append( (200,200,2) )
locations.append( (300,300,'comment 3') )

viewer.mark_region_from_array(locations)
```


SAOImage 1401907326.57
File Edit View Frame Bin Zoom Scale Color Region WCS Analysis Help
File
iabf01bzq_flt.fits[SCI]
Object
Value
WCS
Physical X Y
Image X Y
Frame 1 x 1.773 0.000 °
file edit view frame bin zoom scale color region wcs help
information front back all none delete list load save
comment 3
2
1
9678 10602 11536 12460 13393 14317 15241 16175 17099

**match(coordsys="wcs", frame=True, crop=False, fslice=False,**

scale=False, bin=False, colorbar=False, smooth=False, crosshair=False):

Match all other frames to the current frame

**nancolor(color='red'):**
Set the not-a-number color

**panto_image(x, y):**

Convenience function to change to x,y physical image coordinates

**panto_wcs(x, y, system='fk5'):**
Pan to the wcs location in the image

**readcursor():**
Returns image coordinate postion and key pressed as a tuple of the for float(x), float(y), str(key).

```
In [1]: viewer.readcursor()
Out[2]: (56.0, 28.333333, 'a')

or with a click of the first mouse button

In [1]: viewer.readcursor()
Out[2]: (67.333333, 80.0, '<1>')
```

**reopen():**
Reopen a closed viewing window, mostly used for ginga windows right now

**rotate(value=None, to=False):**
Rotate the current frame (in degrees)

**save_regions(filename=None):**
Save the regions currently displayed in the window to a regions file

**save_rgb(filename=None):**
Save an rgbimage frame as an MEF fits file

**scale(scale='zscale'):**
Scale the pixel values in the window, zscale is the default

**set_region(region_string):**
Use this to send the DS9 viewer a formatted region string it's expecting

For example, in DS9:

```
viewer.set_region("text 110.0 110.0 '1' #font=times")


See the DS9 XPA documentation for more examples.
```

**show_xpa_commands():**
Print the available XPA commands (DS9 only)

**showme():**
Raise the precedence of the viewing window

**showpix():**
Display a pixel value table

**snapsave(filename=None, format=None, resolution=100):**
Create a snapshot of the current window in the specified format

**valid_data_in_viewer():**
Return bool if valid file or array is loaded into the viewer

**view(img, header=None, frame=None, asFits=False):**
Load an image array into the image viewing frame, if no frame is specified, the current frame is used. If no frame exists, then a new one is created. A basic header is created and sent to DS9. You can look at this header with disp_header() but get_header() will return an error because it looks for a filename, and no file was loaded, just the array.

```
image_array=fits.getdata('image.fits')
viewer.view(image_array)

or

image_array=numpy.ones([100,100])*numpy.random.rand(100)
viewer.view(image_array)
```

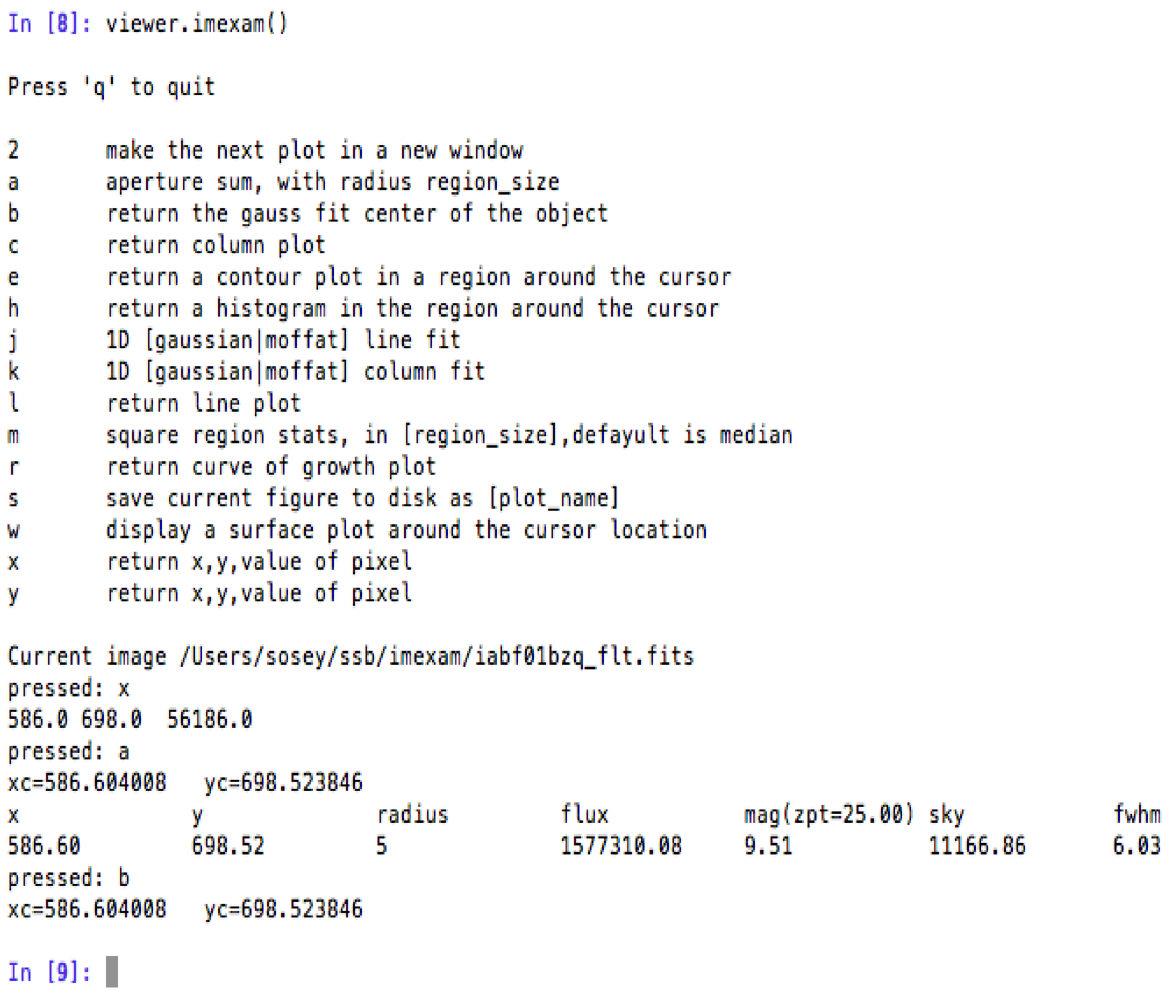
**zoom(par=None):**
Zoom using the specified command in par

**zoomtofit():**
Zoom the image to fit the window

**setlog(self, filename=None, on=True, level=logging.DEBUG):**
Turn on and off `imexam` logging to the a file. You can set the filename to something specific or let the package record to the default logfile. Once you give the object a logfile name, it will continue to use that file until you change it.

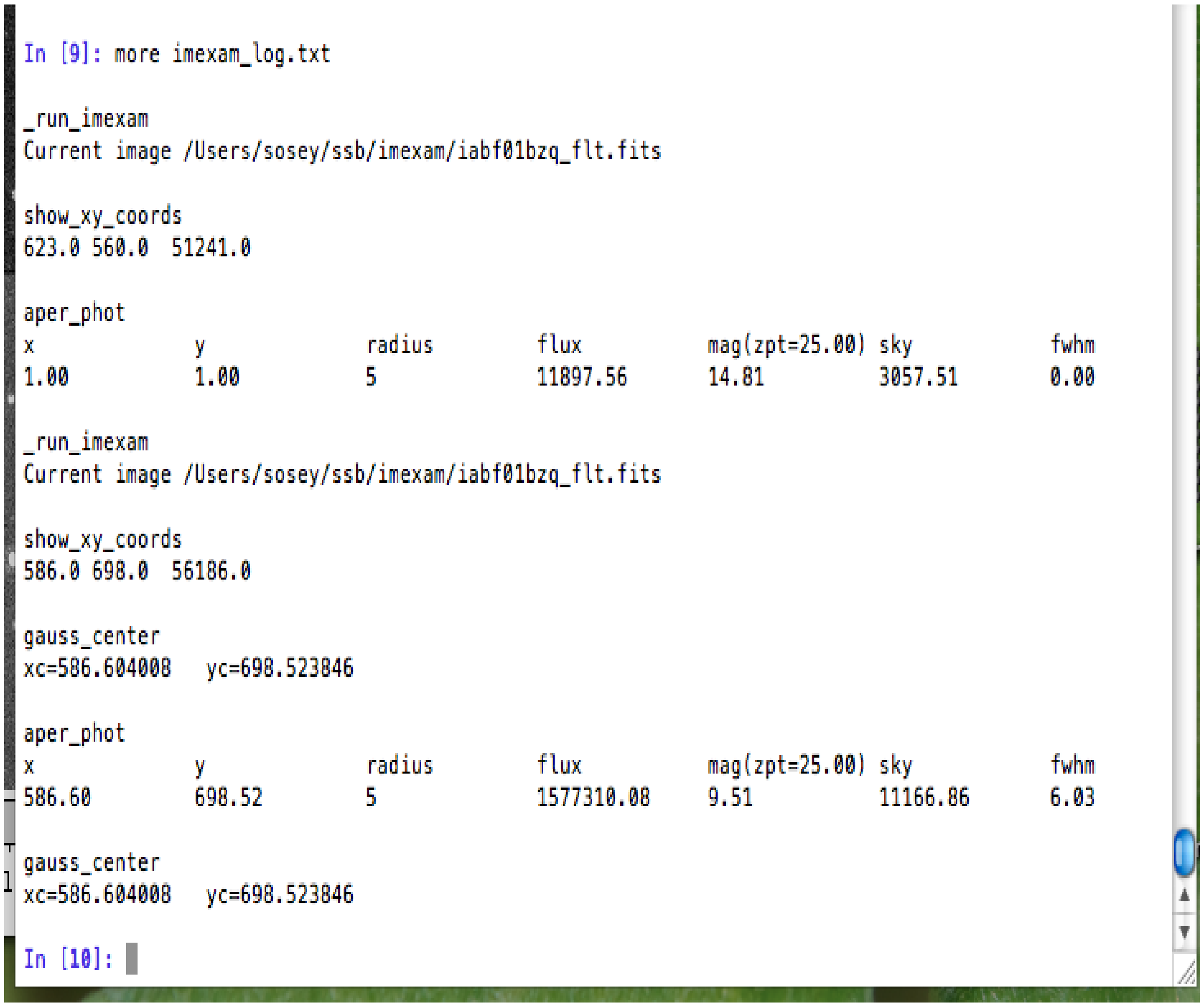
```
In [5]: viewer.setlog()
Saving ``imexam`` commands to imexam_log.txt
```

This is what's displayed in the terminal when you use imexam():



```
In [8]: viewer.imexam()

Press 'q' to quit

2       make the next plot in a new window
a       aperture sum, with radius region_size
b       return the gauss fit center of the object
c       return column plot
e       return a contour plot in a region around the cursor
h       return a histogram in the region around the cursor
j       1D [gaussian|moffat] line fit
k       1D [gaussian|moffat] column fit
l       return line plot
m       square region stats, in [region_size],defayult is median
r       return curve of growth plot
s       save current figure to disk as [plot_name]
w       display a surface plot around the cursor location
x       return x,y,value of pixel
y       return x,y,value of pixel

Current image /Users/sosey/ssb/imexam/iabf01bzq_flt.fits
pressed: x
586.0 698.0  56186.0
pressed: a
xc=586.604008   yc=698.523846
x              y              radius         flux           mag(zpt=25.00) sky            fwhm
586.60         698.52         5              1577310.08     9.51           11166.86       6.03
pressed: b
xc=586.604008   yc=698.523846

In [9]: 
```

and this is what shows up in the logfile:



```
In [9]: more imexam_log.txt

_run_imexam
Current image /Users/sosey/ssb/imexam/iabf01bzq_flt.fits

show_xy_coords
623.0 560.0  51241.0

aper_phot
x              y              radius         flux           mag(zpt=25.00) sky            fwhm
1.00           1.00           5              11897.56       14.81          3057.51        0.00

_run_imexam
Current image /Users/sosey/ssb/imexam/iabf01bzq_flt.fits

show_xy_coords
586.0 698.0  56186.0

gauss_center
xc=586.604008   yc=698.523846

aper_phot
x              y              radius         flux           mag(zpt=25.00) sky            fwhm
586.60         698.52         5              1577310.08     9.51           11166.86       6.03

gauss_center
xc=586.604008   yc=698.523846

In [10]:
```

You can see there are some leftovers from a previous logging session to the same file. You can toggle logging during a session too:

```
viewer.setlog(on=False)

#and to turn off even messages to the screen:

viewer.setlog(on=False,level=logging.CRITICAL)
```

**unlearn():**
Reset all the `imexam` default function parameters

**plotname():**
change or show the default save plotname for imexamine

```
In [1]: viewer.plotname()
imexam_plot.pdf

In [2]: viewer.plotname('myplot.jpg')
In [3]: viewer.plotname()
myplot.jpg
```

The extension of the filename controls the plot type.

**display_help():**

Display the help documentation into a webpage from the locally installed version. This is done from the main package:

```
In [1]: import imexam

In [2]: imexam.display_help()
```

CHAPTER 3

# Convenience functions for DS9's (XPA) commands

**Note:** The full list of XPA access points can be found at: http://ds9.si.edu/doc/ref/xpa.html and XPA itself is maintained here https://github.com/ericmandel/xpa

If there is no convenience function for an access point that you would like to use, you can still call it using the `imexam` hooks into the xpa GET and SET functions. They are aliased to your object (for example "window") as window.window.xpa.get() or window.window.xpa.set()

**alignwcs (on=True):**
align loaded images by wcs,

**blink (blink=None,interval=None):**
blink frames

**clear_contour ():**
clear contours from the screen

**cmap (color=None,load=None,invert=False,save=False, filename='colormap.ds9'):**
set the color map of the current frame The available maps are "heat","grey","cool","aips0","a","b","bb","he","i8"

**color: string**
color must be set to one of the available DS9 color map names

**load: string, optional**
set to the filename which is a valid colormap lookup table valid contrast values are from 0 to 10, and valid bias values are from 0 to 1

**invert: bool, optional**
invert the colormap

**save: bool, optional**
save the current colormap as a file

**filename: string, optional**
the name of the file to save the colormap to

**colorbar (on=True):**
turn the colorbar at the bottom of the screen on and off

**contour (on=True, construct=True):**
on: Set to true to turn on contours

**construct: optional**
Will open the contour dialog box which has more options

**contour_load (filename):**
load contours into the window from the specified filename

**crosshair (x=none,y=none,coordsys="physical",skyframe="wcs",skyformat="fk5",match=False,lock=False):**
control the current position of the crosshair in the current frame, crosshair mode is turned on

**x: string or int**
The value of x is converted to a string for the call to XPA, use a value here appropriate for the skyformat you choose

**y: string or int**
The value of y is converted to a string for the call to XPA, use a value here appropriate for the skyformat you choose

**coordsys: string, optional**
The coordinate system your x and y are defined in

**skyframe: string, optional**
If skyframe has "wcs" in it then skyformat is also sent to the XPA

**skyformat: string, optional**
Used with skyframe, specifies the format of the coordinate which were given in x and y

**match: bool, optional**
If set to True, then the wcs is matched for the frames

**lock: bool, optional**
If set to True, then the frame is locked in wcs

**cursor (x=None,y=None):**
move the cursor in the current frame to the specified image pixel, it will also move selected regions

**disp_header ():**
display the current header using the ds9 header display window

**frame (n=None):**
convenience function to switch frames or load a new frame (if that number does not already exist)

**n: int, string, optional**
The frame number to open or change to. If the number specified doesn't exist, a new frame will be opened If nothing is specified, then the current frame number will be returned. The value of n is converted to a string before passing to the XPA

frame(1) sets the current frame to 1 frame("last") set the current frame to the last frame frame() returns the number of the current frame frame("new") opens a new frame frame(3) opens frame 3 if it doesn't exist already, otherwise goes to frame 3

**get_header ():**
return the header of the current extension as a string, or None if there's a problem

**grid (on=True, param=False):**
turn the grid on and off if param is True, then a diaglog is opened for the grid parameters

**hideme ():**
lower the ds9 window on your display

**load_fits (fname=None, extname=1, extver='SCI'):**
load a fits image to the current frame. You provide just the name, or either of the extname or extver, or you can specify the extension with the filename string. For example:

load_fits('something.fits',extver='SCI') will load the SCI,1 extension

load_fits('something.fits[SCI,1]') will load the SCI,1 extension

load_fits('something.fits') will load the main data extension; the only data information in the case of simple fits, or the first extension in the case of a multiextension file

**load_region (filename):**
load the specified DS9 formatted region filename

**load_rgb (red, green, blue,scale=False, lockwcs=False):**
load 3 images into an RGBimage frame, the parameters are:

```
red: string, The name of the fits file which will be loaded into the red channel

green: string, The name of the fits file which will be loaded into the green channel

blue: string, The name of the fits file which will be loaded into the blue channel

scale: bool, If True, then each image will be scale with zscale() after loading

lockwcs: bool, If True, then the image positions will be locked to each other using the WCS␣
↪information in their headers
```

**load_mef_as_cube (filename=None):**
Load a Mult-Extension-Fits image into one frame as an image cube in the image viewer

**load_mef_as_multi (filename=None):**
Load a Mult-Extension-Fits image into multiple frames in the image viewer

**match (coordsys="physical",frame=False,crop=False,fslice=False,scale=False,bin=False,colorbar=False,smooth=False,crosshair**

match all other frames to the current frame using the specified option. You can only choose one of the options at a time, so set frame=False and something else in addition to your choice if you don't want the default option.

**coordsys: string, optional**
The coordinate system to use

**frame: bool, optional**
Match all other frames to the current frame, using the set coordsys

**crop: bool, optional**
Set the current image display area, using the set coordsys

**fslice: bool, optional**
Match current slice in all frames

**scale: bool, optional**
Match to the current scale for all frames

**bin: bool, optional**
Match to the current binning for all frames

**colorbar: bool, optional**
Match to the current colorbar for all frames

**smooth: bool, optional**
Match to the current smoothing for all frames

**crosshair: bool, optional**
Match the crosshair in all frames, using the current coordsys

**nancolor (color=”red”):**
set the not-a-number color, default is red

**panto_image (x, y)**
convenience function to change to x,y images coordinates using ra,dec

**panto_wcs (x, y,system=’fk5’):**
pan to the wcs coordinates in the image using the specified system

**x: string**
The x location to move to, specified using the given system

**y: string**
The y location to move to

**system: string**
The reference system that x and y were specified in, they should be understood by DS9

**rotate (value, to=False):**

**value: float [degrees]**
Rotate the current frame {value} degrees If value is 0, then the current rotation is printed

**to: bool**
Rotate the current frame to the specified value instead

**save_header (filename=None):**
save the header of the current image to a file

**save_rgb (filename=None):**
save an rgbimage frame as an MEF fits file

**save_regions (filename=None):**
Save the regions in the current window to a DS9 style regions file

**filename: string**
The nameof th file to which the regions displayed in the current window are saved If no filename is provided then it will try and save the regions to the name of the file in the current display with _regions.txt appended

If a file of that name already exists on disk it will no attempt to overwrite it

**scale (scale=’zscale’):**
Scale the image on display. The default zscale is the most widely used option:

```
Syntax

scales available: [linear|log|pow|sqrt|squared|asinh|sinh|histequ]

[log exp <value>]
[datasec yes|no]
[limits <minvalue> <maxvalue>]
[mode minmax|<value>|zscale|zmax]
[scope local|global]
[match]
[lock [yes|no]]
[open|close]
```

**set_region (region_string):**
display a region using the specifications in region_string example: set_region("physical; ruler 200 300 200 400")

**showme ():**
raise the ds9 display window

**showpix ():**
display the pixel value table

**snapsave (filename,format=None,resolution=100):**
create a snap shot of the current window and save in specified format. If no format is specified the filename extension is used

**filename: str, optioan**
filename of output image, the extension in the filename can also be used to specify the format If no filename is specified, then the filename will be constructed from the name of the currently displayed image with _snap.jpg appended.

**format: str, optional**
available formats are fits, eps, gif, tiff, jpeg, png If no format is specified the filename extension is used

**resolution: int, optional**
1 to 100, for jpeg images

**zoom (par="to fit"):**

**par: string**
it can be a number (ranging 0.1 to 8), and successive calls continue zooming in the same direction it can be two numbers '4 2', which specify zoom on different axis if can be to a specific value 'to 8' or 'to fit', "to fit" is the default it can be 'open' to open the dialog box it can be 'close' to close the dialog box (only valid if the box is already open)

**zoomtofit ():**
zoom to the best fit for the display window

CHAPTER 4

# Example 1

**Note:** More examples in the form of Jupyter notebooks can be downloaded from the git repository and are contained in the “example_notebooks” directory.

## 4.1 Basic Usage

First you need to import the package

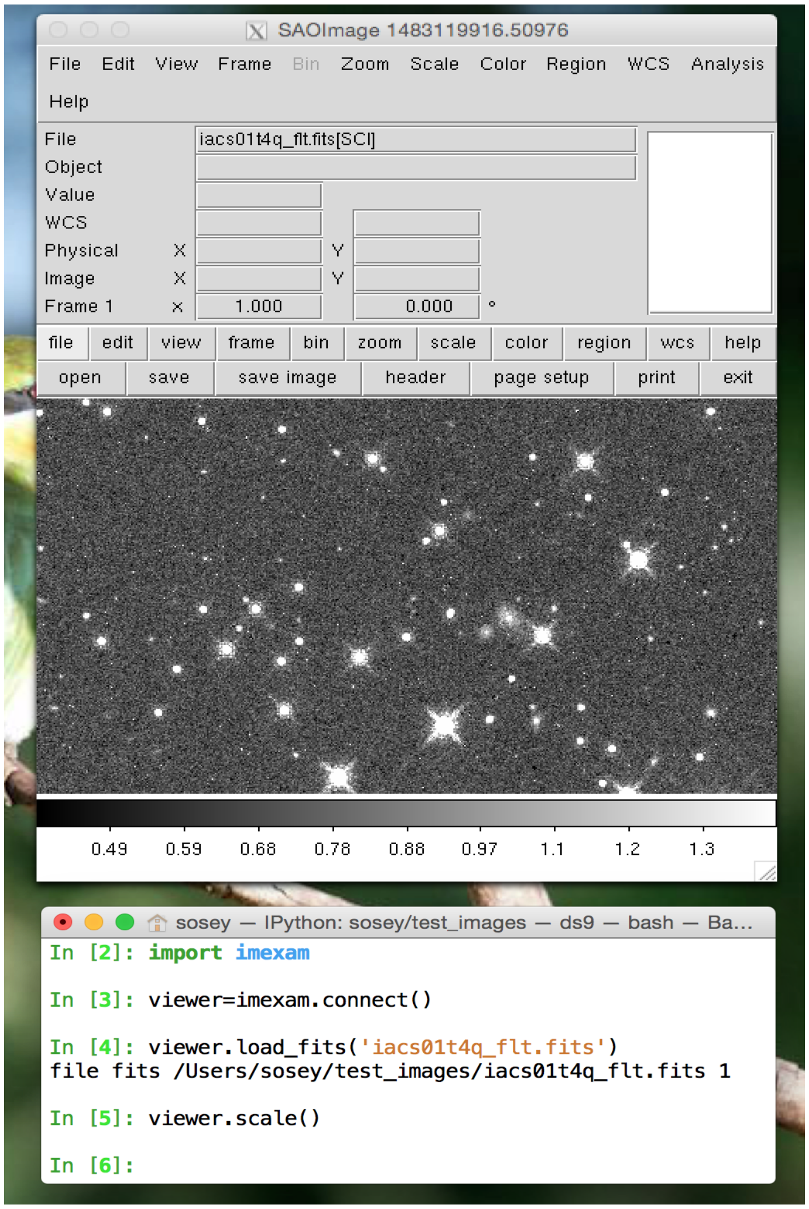
```
import imexam
```

## 4.2 Usage with D9 (the current default viewer)

If you are on a windows system, DS9 may not be available, so move on to the Ginga specification.

Start up a `DS9` window (by default), a new `DS9` window will be opened, open a fits image, and scale it:

```
viewer=imexam.connect()
viewer.load_fits('iacs01t4q_flt.fits')
viewer.scale()
```


SAOImage 1483119916.50976
File Edit View Frame Bin Zoom Scale Color Region WCS Analysis
Help
File iacs01t4q_flt.fits[SCI]
Object
Value
WCS
Physical X Y
Image X Y
Frame 1 × 1.000 0.000 °
file edit view frame bin zoom scale color region wcs help
open save save image header page setup print exit
0.49 0.59 0.68 0.78 0.88 0.97 1.1 1.2 1.3
sosey — IPython: sosey/test_images — ds9 — bash — Ba...

```
In [2]: import imexam

In [3]: viewer=imexam.connect()

In [4]: viewer.load_fits('iacs01t4q_flt.fits')
file fits /Users/sosey/test_images/iacs01t4q_flt.fits 1

In [5]: viewer.scale()

In [6]:
```

If you already have a window running, you can ask for a list of windows; windows that you start from the imexam package will not show up, this is to keep control over their processes and prevent double assignments.

```
# This will display if you've used the default command above and have no other DS9 windows open
In [1]: imexam.list_active_ds9()
No active sessions registered
Out[2]: {}

# open a window in another process
In [3]: !ds9&
In [4]: imexam.list_active_ds9()
DS9 ds9 gs /tmp/xpa/DS9_ds9.60457 sosey
Out[5]: {'/tmp/xpa/DS9_ds9.60457': ('ds9', 'sosey', 'DS9', 'gs')}imexam.list_active_ds9()
DS9 ds9 gs 82a7e75f:57222 sosey
```

You can attach to a current DS9 window be specifying its unique name

```
viewer1=imexam.connect('ds9')
```

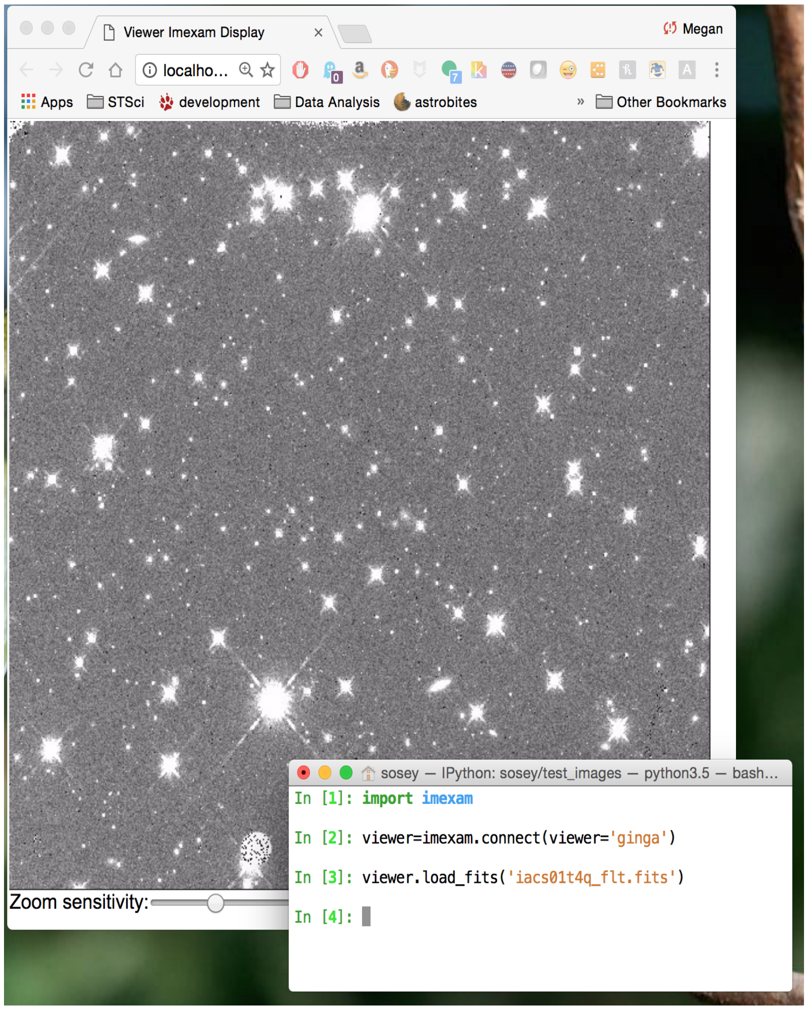
If you haven't given your windows unique names using the `-t <name>` option from the commandline, then you must use the ip:port address:

```
viewer=imexam.connect('82a7e75f:57222')
```

## 4.3 Usage with Ginga viewer

Start up a ginga window using the HTML5 backend and display the same image as above. Make sure that you have installed the most recent version of ginga, `imexam` will return an error that the viewer cannot be found otherwise.:

```
viewer=imexam.connect(viewer='ginga')
viewer.load_fits()
```



```
In [1]: import imexam

In [2]: viewer=imexam.connect(viewer='ginga')

In [3]: viewer.load_fits('iacs01t4q_flt.fits')

In [4]:
```

**Note:** All commands after your chosen viewer is opened are the same. Each viewer also has it's own set of commands which you can additionally use. You may use any viewer for the examples which follow.

Load a fits image into the window:

```
viewer.load_fits('test.fits')
```

Scale the image to the default scaling, which is a zscale algorithm, but the viewers other scaling options are also available:

```
viewer.scale()
viewer.scale('asinh')  <-- uses asinh
```

Change to heat map colorscheme:

```
viewer.cmap(color='heat')
```

Make some marks on the image and save the regions using a `DS9` style regions file:

```
viewer.save_regions('test.reg')
```

Delete all the regions you made, then load from file:

```
viewer.load_regions('test.reg')
```

Plot stuff at the cursor location, in a while loop. Type a key when the mouse is over your desired location and continue plotting with the available options:

```
viewer.imexam()
```

Quit out and delete windows and references, for the ginga HTML5 window, this will not close the browser window with the image display, you'll need to exit that manually. However, if you've accidentally closed that window you can reopen and reconnect to the server:

```
viewer.close()
viewer.reopen()
```

# CHAPTER 5

# Example 2

## 5.1 Aperture Photometry

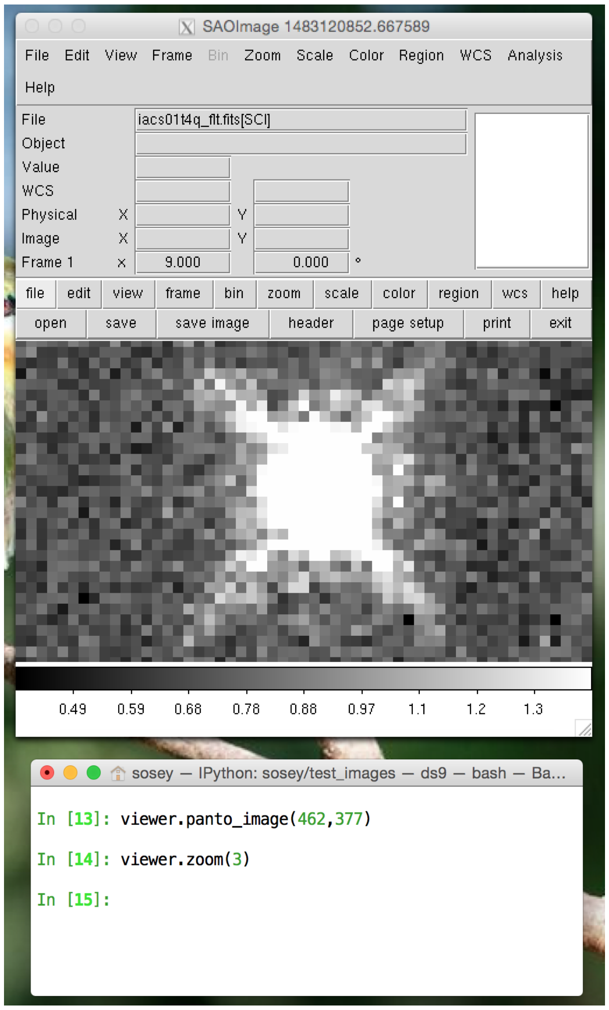
- Perform manual aperture photometry on supplied image
- Make curve of growth and radial profile plots
- Save the profile data and plot to files.

### 5.1.1 Method 1

Assuming we've already connected to the window where the data is displayed:

- This method first uses the "a" key to check out the aperture photometry with the default settings
- Display a radial profile "r" plot around the start we choose
- Look at the curve of growth "g" plot
- Make a new profile plot, print the plotted points to the screen, and save a copy of the plotting window for reference

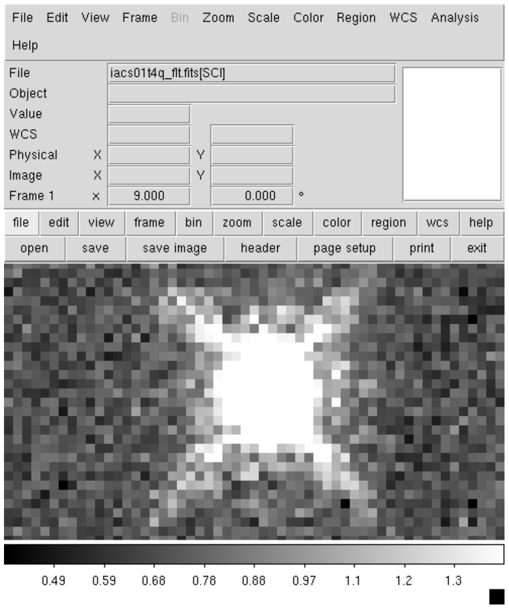
Here a picture of the area I'm looking at on my desktop:


SAOImage 1483120852.667589
File Edit View Frame Bin Zoom Scale Color Region WCS Analysis
Help
file edit view frame bin zoom scale color region wcs help
open save save image header page setup print exit
0.49 0.59 0.68 0.78 0.88 0.97 1.1 1.2 1.3
sosey — IPython: sosey/test_images — ds9 — bash — Ba…

File iacs01t4q_flt.fits[SCI]
Object
Value
WCS
Physical X Y
Image X Y
Frame 1 x 9.000 0.000 °

```
In [13]: viewer.panto_image(462,377)

In [14]: viewer.zoom(3)

In [15]:
```

If you wanted to save a screenshot of the viewer display you can use viewer.grab(), in DS9 this will save a snap of the whole DS9 window for reference:

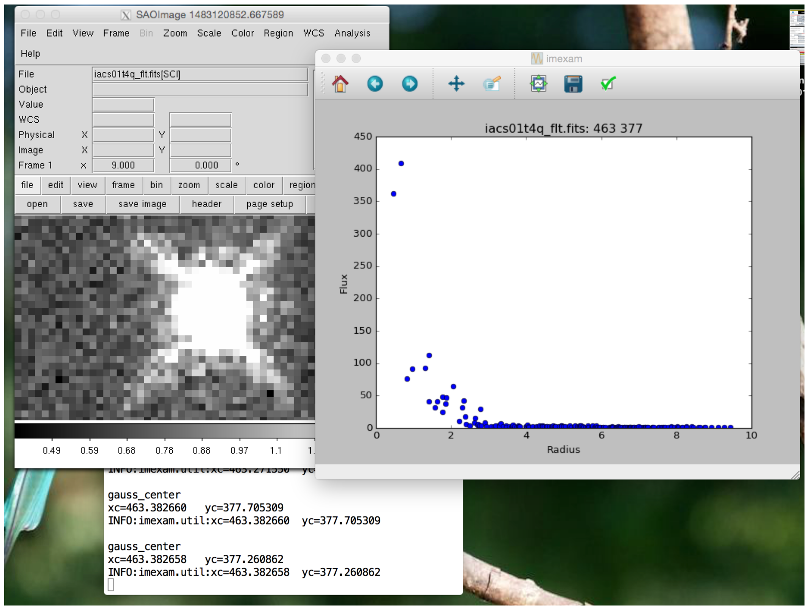

File Edit View Frame Bin Zoom Scale Color Region WCS Analysis
Help
file edit view frame bin zoom scale color region wcs help
open save save image header page setup print exit
0.49 0.59 0.68 0.78 0.88 0.97 1.1 1.2 1.3

File iacs01t4q_flt.fits[SCI]
Object
Value
WCS
Physical X Y
Image X Y
Frame 1 x 9.000 0.000 °

Now let's start up the `imexam()` loop and look at a plot of star:

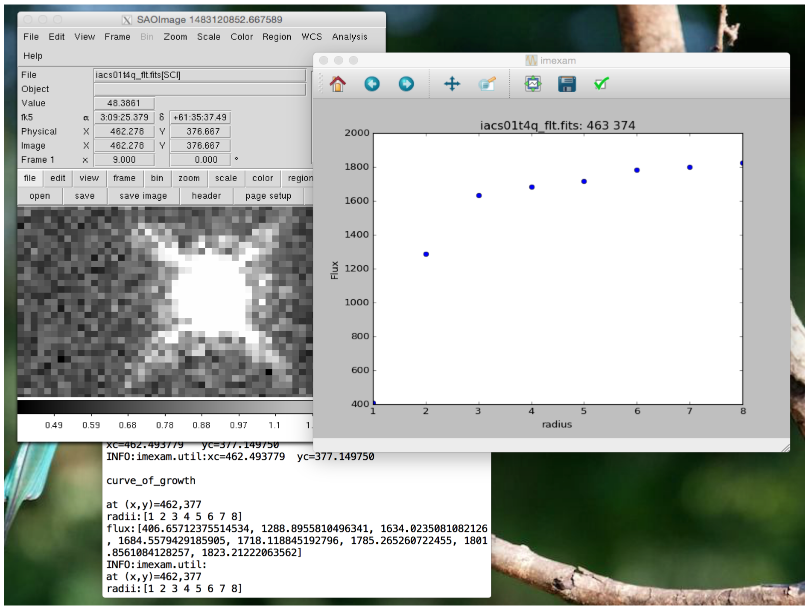
```
viewer.imexam() #start an imexam session
```

Use the "r" and "g" keys to look at the radial profile and growth curves:


SAOImage 1483120852.667589
File Edit View Frame Bin Zoom Scale Color Region WCS Analysis Help
iacs01t4q_flt.fits[SCI]
imexam
iacs01t4q_flt.fits: 463 377
Flux
Radius

```
gauss_center
xc=463.382660   yc=377.705309
INFO:imexam.util:xc=463.382660  yc=377.705309

gauss_center
xc=463.382658   yc=377.260862
INFO:imexam.util:xc=463.382658  yc=377.260862
```

Note that part of the screen information that's returned includes the flux and radii information:

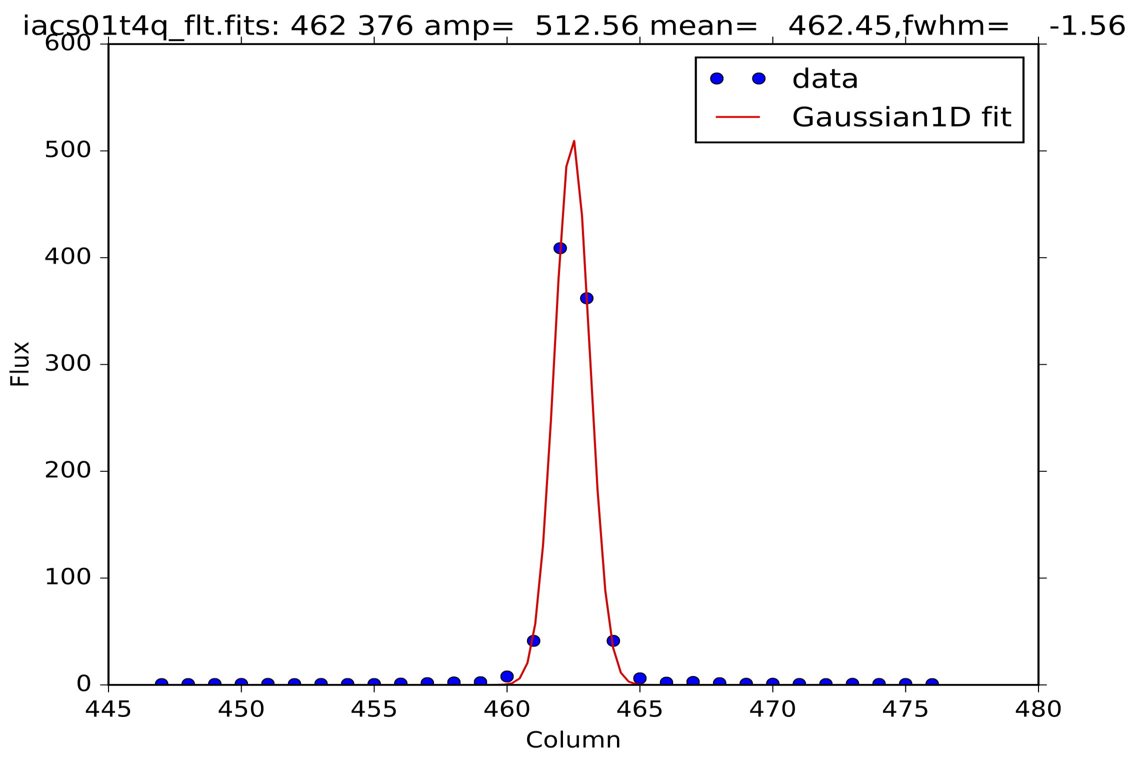

SAOImage 1483120852.667589
File Edit View Frame Bin Zoom Scale Color Region WCS Analysis Help
iacs01t4q_flt.fits[SCI]
imexam
iacs01t4q_flt.fits: 463 374
Flux
radius

```
INFO:imexam.util:xc=462.493779  yc=377.149750

curve_of_growth

at (x,y)=462,377
radii:[1 2 3 4 5 6 7 8]
flux:[406.65712375514534, 1288.8955810496341, 1634.0235081082126
, 1684.5579429185905, 1718.118845192796, 1785.265260722455, 1801
.8561084128257, 1823.21222063562]
INFO:imexam.util:
at (x,y)=462,377
radii:[1 2 3 4 5 6 7 8]
```

Let's take this information and set the radii for our quick aperture photometry:

```
In [1]: viewer.aimexam()
Out[2]:
{'center': [True, 'Center the object location using a 2d gaussian fit'],
'function': ['aper_phot'],
'radius': [5, 'Radius of aperture for star flux'],
'skyrad': [15, 'Distance to start sky annulus is pixels'],
'subsky': [True, 'Subtract a sky background?'],
'width': [5, 'Width of sky annulus in pixels'],
'zmag': [25.0, 'zeropoint for the magnitude calculation']}

In [3]: viewer.set_plot_pars('a','radius',4)
set aper_phot_pars: radius to 4

In [4]: viewer.set_plot_pars('a','skyrad',8)
set aper_phot_pars: skyrad to 8

In [23]: viewer.imexam()

Press 'q' to quit

2   Make the next plot in a new window
a   Aperture sum, with radius region_size
b   Return the 2D gauss fit center of the object
c   Return column plot
e   Return a contour plot in a region around the cursor
g   Return curve of growth plot
h   Return a histogram in the region around the cursor
j   1D [Gaussian1D default] line fit
k   1D [Gaussian1D default] column fit
l   Return line plot
m   Square region stats, in [region_size],default is median
r   Return the radial profile plot
s   Save current figure to disk as [plot_name]
t   Make a fits image cutout using pointer location
w   Display a surface plot around the cursor location
x   Return x,y,value of pixel
y   Return x,y,value of pixel
Current image /Users/sosey/test_images/iacs01t4q_flt.fits

gauss_center
xc=462.827108       yc=377.705312

aper_phot
x          y        radius    flux       mag(zpt=25.00)  sky    fwhm
462.83     377.71   4         1686.24    16.93           0.92   1.71
```

Just for some more information on the star, below is the gaussian fit “j” to the columns of the same star.



### 5.1.2 Method 2

Assuming we've already connected to the DS9 window where the data is displayed:

- First we turn on logging so that everything gets saved to a file
- We then use the "a" key to check out the aperture photometry with the default settings
- Use the "g" to look at the curve of growth
- Adjust the aperture photometry with our our own settings
- We can then use the log file, to create a plot

```
In [1]: viewer.setlog('mystar.log')
Saving imexam commands to mystar.log
In [2]: viewer.unlearn()

In [3]: viewer.imexam()

Press 'q' to quit

2   Make the next plot in a new window
a   Aperture sum, with radius region_size
b   Return the 2D gauss fit center of the object
c   Return column plot
e   Return a contour plot in a region around the cursor
g   Return curve of growth plot
h   Return a histogram in the region around the cursor
j   1D [Gaussian1D default] line fit
k   1D [Gaussian1D default] column fit
```

```
l   Return line plot
m   Square region stats, in [region_size],default is median
r   Return the radial profile plot
s   Save current figure to disk as [plot_name]
t   Make a fits image cutout using pointer location
w   Display a surface plot around the cursor location
x   Return x,y,value of pixel
y   Return x,y,value of pixel
Current image /Users/sosey/test_images/iacs01t4q_flt.fits

xc=462.938220       yc=377.260860
x          y         radius     flux       mag(zpt=25.00)  sky    fwhm
462.94     377.26    5          1739.97    16.90           0.72   1.44

at (x,y)=462,377
radii:[1 2 3 4 5 6 7 8]
flux:[406.65712375514534, 1288.8955810496341, 1634.0235081082126,
1684.5579429185905, 1718.118845192796, 1785.265260722455,
1801.8561084128257, 1823.21222063562]
```

Lets get some more aperture photometry at larger radii by resetting some of the “a” key values::

```
In [4]: viewer.set_plot_pars("a","radius",4)
set aper_phot_pars: radius to 4

In [5]: viewer.set_plot_pars("a","skyrad",8)
set aper_phot_pars: skyrad to 8

In [5]: viewer.imexam()  #use the "a" key

xc=463.049330       yc=377.038640
x          y         radius     flux       mag(zpt=25.00)  sky    fwhm
463.05     377.04    4          1679.23    16.94           0.93   1.71
```

This is what mystar.log contains, you can parse the log, or copy the data and use as you like to make interesting plots later or just have for reference.:

```
gauss_center
xc=462.938220   yc=377.260860

aper_phot
x          y         radius     flux       mag(zpt=25.00)  sky    fwhm
462.94     377.26    5          1739.97    16.90           0.72   1.44

gauss_center
xc=462.827110   yc=377.371969

gauss_center
xc=462.827109   yc=377.260860

gauss_center
xc=462.827109   yc=377.260860

curve_of_growth

at (x,y)=462,377
radii:[1 2 3 4 5 6 7 8]
flux:[406.65712375514534, 1288.8955810496341, 1634.0235081082126,
1684.5579429185905, 1718.118845192796, 1785.265260722455,
1801.8561084128257, 1823.21222063562]

gauss_center
xc=463.049330   yc=377.038640

aper_phot
x           y         radius     flux       mag(zpt=25.00)  sky    fwhm
463.05      377.04    4          1679.23    16.94           0.93   1.71
```

# CHAPTER 6

# Example 3

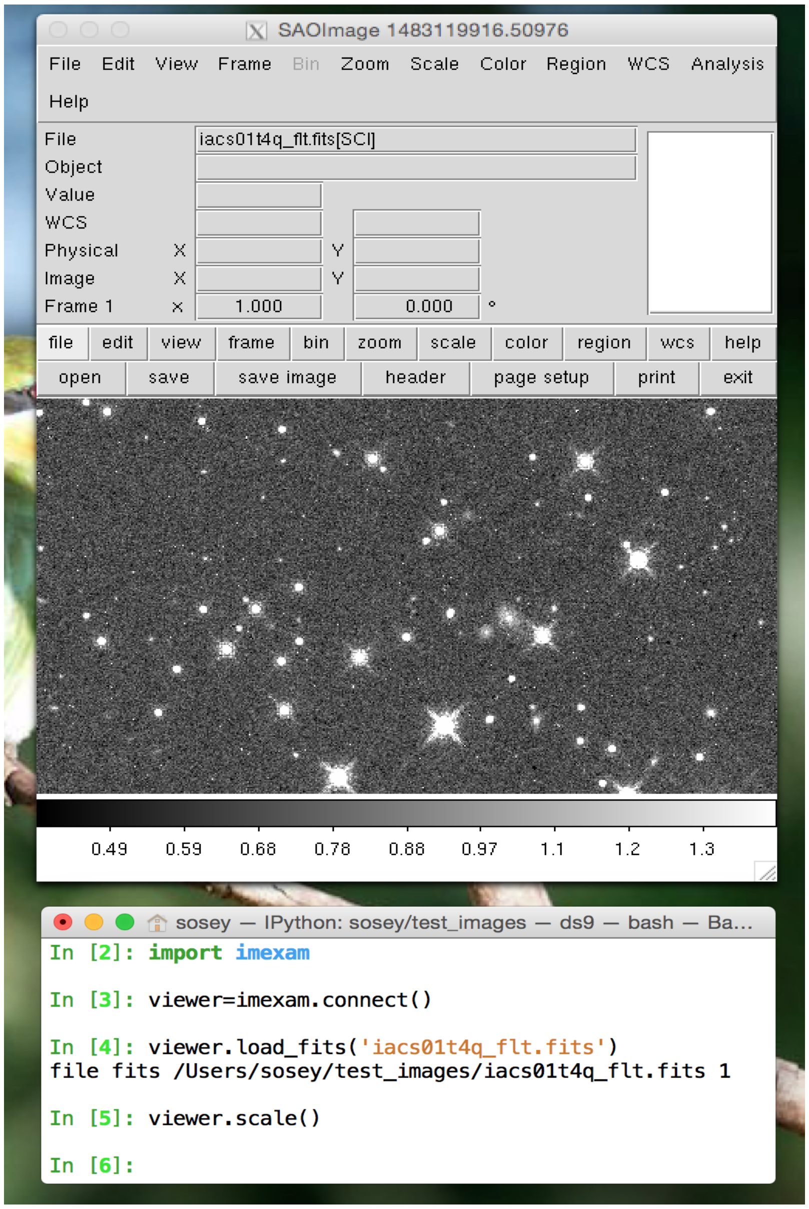
## 6.1 Advanced Usage - Interact with Daophot and Astropy

While the original intent for the imexam module was to replicate the realtime interaction of the old IRAF imexamine interface with data, there are other possibilities for data analysis which this module can support.One such example, performing more advanced interaction which can be scripted, is outlined below, using familiar IRAF tasks.

**Note:** You can see a similar photometry example which uses `photutils` and it's implementation of DAOPhot aperture photometry instead of IRAF in the imexam_ds9_photometry example jupyter notebook.

If you have a list of source identifications, perhaps prepared by SExtractor, DAOFind, Starfind or a similar program, you can use `imexam` to display the science image and overlay apertures for all their locations. From there you can do some visual examination and cleaning up of the list with a combination of region manipulation and useful `imexam` methods.

Here's our example image to work with, which is a subsection of a larger image:


SAOImage 1483119916.50976
File Edit View Frame Bin Zoom Scale Color Region WCS Analysis
Help
File
Object
Value
WCS
Physical X Y
Image X Y
Frame 1 x
file edit view frame bin zoom scale color region wcs help
open save save image header page setup print exit
0.49 0.59 0.68 0.78 0.88 0.97 1.1 1.2 1.3
sosey — IPython: sosey/test_images — ds9 — bash — Ba...

File: iacs01t4q_flt.fits[SCI]
Frame 1 x 1.000 0.000 °

```
In [2]: import imexam

In [3]: viewer=imexam.connect()

In [4]: viewer.load_fits('iacs01t4q_flt.fits')
file fits /Users/sosey/test_images/iacs01t4q_flt.fits 1

In [5]: viewer.scale()

In [6]:
```

I'll use the IRAF DAOFind to find objects in my field:

```
from pyraf import iraf
from iraf import noao,digiphot,daophot
from astropy.io import fits

image='iabf01bzq_flt.fits'

fits.info('iabf01bzq_flt.fits')

    Filename: iabf01bzq_flt.fits
    No.    Name         Type      Cards   Dimensions   Format
    0     PRIMARY     PrimaryHDU     210   ()           int16
    1     SCI         ImageHDU        81   (1014, 1014)   float32
    2     ERR         ImageHDU        43   (1014, 1014)   float32
    3     DQ          ImageHDU        35   (1014, 1014)   int16
    4     SAMP        ImageHDU        30   ()           int16
    5    TIME        ImageHDU        30   ()           float32



#set up some finding parameters, you can make this more explicit
iraf.daophot.findpars.threshold=3.0 #3sigma detections only
iraf.daophot.findpars.nsigma=1.5 #width of convolution kernal in sigma
iraf.daophot.findpars.ratio=1.0 #ratio of gaussian axes
iraf.daophot.findpars.theta=0.
iraf.daophot.findpars.sharplo=0.2 #lower bound on feature
iraf.daophot.findpars.sharphi=1.0 #upper bound on feature
iraf.daophot.findpars.roundlo=-1.0 #lower bound on roundness
iraf.daophot.findpars.roundhi=1.0 #upper bound on roundness
iraf.daophot.findpars.mkdetections="no"

    In [84]: iraf.lpar(iraf.daophot.datapars)
           (scale = 1.0)            Image scale in units per pixel
         (fwhmpsf = 2.5)            FWHM of the PSF in scale units
        (emission = yes)            Features are positive?
           (sigma = 1.0)            Standard deviation of background in counts
         (datamin = 0.0)            Minimum good data value
         (datamax = INDEF)          Maximum good data value
           (noise = "poisson")      Noise model
         (ccdread = "")             CCD readout noise image header keyword
            (gain = "ccdgain")      CCD gain image header keyword
       (readnoise = 2.0)            CCD readout noise in electrons
           (epadu = 1.0)            Gain in electrons per count
        (exposure = "exptime")      Exposure time image header keyword
         (airmass = "")             Airmass image header keyword
          (filter = "")             Filter image header keyword
         (obstime = "")             Time of observation image header keyword
           (itime = 1.0)            Exposure time
        (xairmass = INDEF)          Airmass
         (ifilter = "INDEF")        Filter
           (otime = "INDEF")        Time of observation
            (mode = "ql")

iraf.daophot.datapars.datamin=0.
iraf.daophot.datapars.gain="ccdgain"
iraf.daophot.datapars.exposure="exptime"
iraf.daophot.datapars.sigma=105.


#assume the science extension and find some stars
sci="[SCI,1]"
output_locations='iabf01bzq_stars.dat'
iraf.daofind(image=image+sci,output=output_locations,interactive="no",verify="no",verbose="no")

#This is just the top of the file that daofind produced:

    In [24]: more iabf01bzq_stars.dat
    #K IRAF       = NOAO/IRAFV2.16          version    %-23s
    #K USER       = sosey                   name       %-23s
    #K HOST       = intimachay.stsci.edu    computer   %-23s
    #K DATE       = 2014-03-28              yyyy-mm-dd %-23s
    #K TIME       = 15:34:56                hh:mm:ss   %-23s
    #K PACKAGE    = apphot                  name       %-23s
    #K TASK       = daofind                 name       %-23s
```

```
#
#K SCALE      = 1.                     units      %-23.7g
#K FWHMPSF    = 2.5                    scaleunit  %-23.7g
#K EMISSION   = yes                    switch     %-23b
#K DATAMIN    = 0.                     counts     %-23.7g
#K DATAMAX    = INDEF                  counts     %-23.7g
#K EXPOSURE   = exptime                keyword    %-23s
#K AIRMASS    = ""                     keyword    %-23s
#K FILTER     = ""                     keyword    %-23s
#K OBSTIME    = ""                     keyword    %-23s
#
#K NOISE      = poisson                model      %-23s
#K SIGMA      = 105.                   counts     %-23.7g
#K GAIN       = ccdgain                keyword    %-23s
#K EPADU      = 2.5                    e-/adu     %-23.7g
#K CCDREAD    = ""                     keyword    %-23s
#K READNOISE  = 0.                     e-         %-23.7g
#
#K IMAGE      = iabf01bzq_flt.fits[SCI, imagename  %-23s
#K FWHMPSF    = 2.5                    scaleunit  %-23.7g
#K THRESHOLD  = 3.                     sigma      %-23.7g
#K NSIGMA     = 2.                     sigma      %-23.7g
#K RATIO      = 1.                     number     %-23.7g
#K THETA      = 0.                     degrees    %-23.7g
#
#K SHARPLO    = 0.2                    number     %-23.7g
#K SHARPHI    = 1.                     number     %-23.7g
#K ROUNDLO    = -1.                    number     %-23.7g
#K ROUNDHI    = 1.                     number     %-23.7g
#
#N XCENTER    YCENTER   MAG      SHARPNESS   SROUND      GROUND      ID         \
#U pixels     pixels    #        #           #           #           #          \
#F %-13.3f    %-10.3f   %-9.3f   %-12.3f     %-12.3f     %-12.3f     %-6d       \
#
   194.694    2.357     -3.335   0.919       0.141       -0.004      1
   232.659    2.889     -1.208   0.768       0.572       -0.289      2
   237.782    2.925     -1.182   0.669       0.789       -0.971      3
   265.715    2.797     -1.395   0.976       -0.450      -0.669      4
   419.792    2.902     -3.045   0.925       -0.990      0.213       5
   424.566    3.081     -1.202   0.923       0.513       -0.555      6
   534.758    2.856     -1.341   0.659       -0.676      -0.302      7
   580.964    2.485     -1.326   0.821       -0.489      -0.752      8
   587.521    3.568     -1.282   0.911       -0.537      -0.119      9
   725.016    3.999     -1.103   0.714       -0.653      -0.490      10
   736.495    2.808     -1.345   0.710       -0.996      -0.730      11
   746.529    3.200     -0.868   0.303       -0.376      -0.682      12
   757.672    3.172     -1.527   0.420       0.271       0.211       13
   768.768    2.830     -1.321   0.741       -0.842      -0.252      14
   799.199    2.696     -2.096   0.926       0.476       -0.511      15
   807.575    2.445     -4.136   0.745       0.171       -0.131      16
   836.661    2.790     -1.482   0.709       0.205       0.636       17
   879.390    3.069     -1.018   0.549       -0.479      -0.495      18
   912.820    2.806     -1.414   0.576       0.504       0.109       19
   938.794    3.448     -1.731   0.997       -0.239      0.100       20
   17.713     2.731     -1.896   0.286       -0.947      -0.359      21
   48.757     2.755     -1.172   0.586       0.646       -0.543      22
   105.894    3.030     -1.700   0.321       -0.233      -0.006      23
```

Now we want to read in the file that Daofind produced and save the x,y and ID information. I'm going to read the results using astropy.io.ascii

```
reader=ascii.Daophot()
photfile=reader.read(output_locations)

#some quick information on what we have now
photfile.colnames

    ['XCENTER', 'YCENTER', 'MAG', 'SHARPNESS', 'SROUND', 'GROUND', 'ID']

photfile.print()

    In [103]: photfile.pprint()
       XCENTER      YCENTER       MAG     SHARPNESS       SROUND        GROUND       ID
    ------------- ---------- --------- ------------ ------------ ------------ ------
    194.694       2.357      -3.335    0.919        0.141        -0.004       1
    232.659       2.889      -1.208    0.768        0.572        -0.289       2
    237.782       2.925      -1.182    0.669        0.789        -0.971       3
    265.715       2.797      -1.395    0.976        -0.450       -0.669       4
    419.792       2.902      -3.045    0.925        -0.990       0.213        5
    424.566       3.081      -1.202    0.923        0.513        -0.555       6
    534.758       2.856      -1.341    0.659        -0.676       -0.302       7
    580.964       2.485      -1.326    0.821        -0.489       -0.752       8
    587.521       3.568      -1.282    0.911        -0.537       -0.119       9
    725.016       3.999      -1.103    0.714        -0.653       -0.490       10
    736.495       2.808      -1.345    0.710        -0.996       -0.730       11
    746.529       3.200      -0.868    0.303        -0.376       -0.682       12
    757.672       3.172      -1.527    0.420        0.271        0.211        13
    768.768       2.830      -1.321    0.741        -0.842       -0.252       14
    799.199       2.696      -2.096    0.926        0.476        -0.511       15
    807.575       2.445      -4.136    0.745        0.171        -0.131       16
```

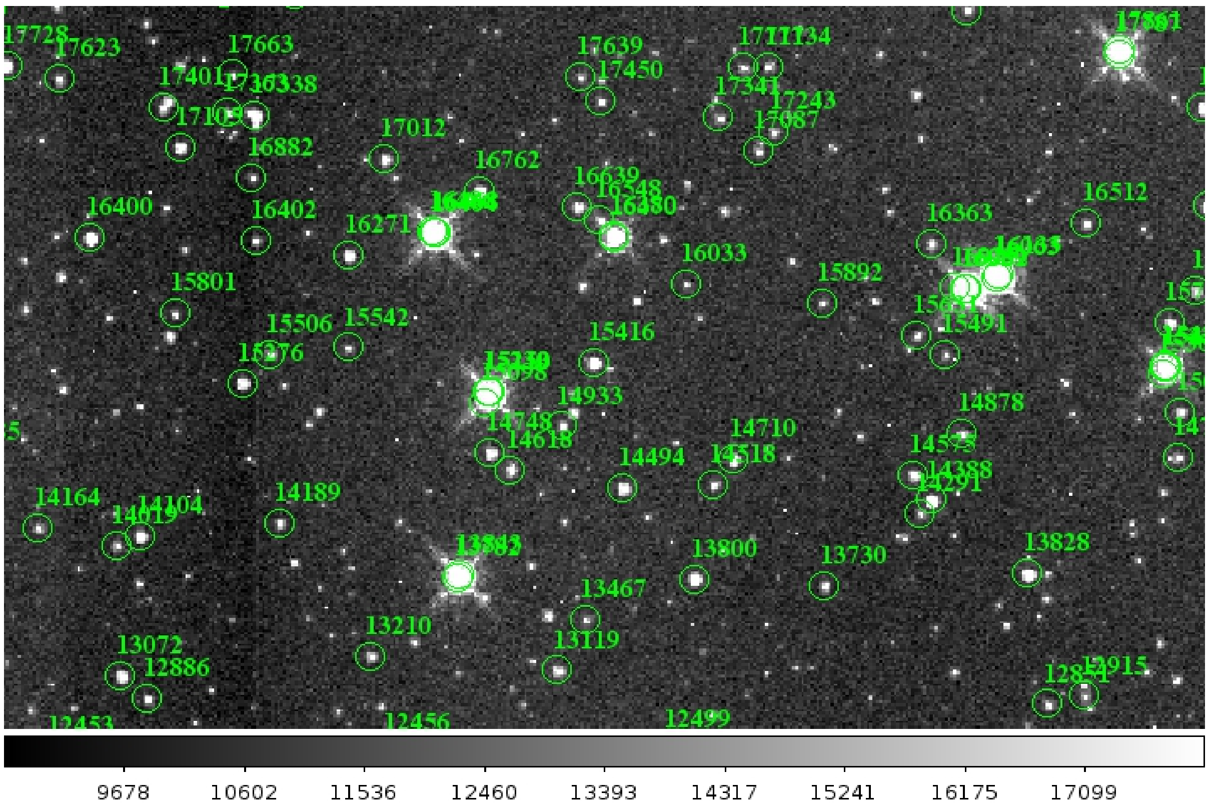
You can even pop this up in your web browser if that's a good format for you: `photfile.show_in_browser()`. `imexam` has several functions to help display regions on the DS9 window. Since we have this data loaded into memory, the one we will use here is `mark_region_from_array()`.

Let's make an array that the method will accept, namely a list of tuples which contain the (x,y,comment) that we want marked to the display. It will also accept any iterator containing a tuple of (x,y,comment).

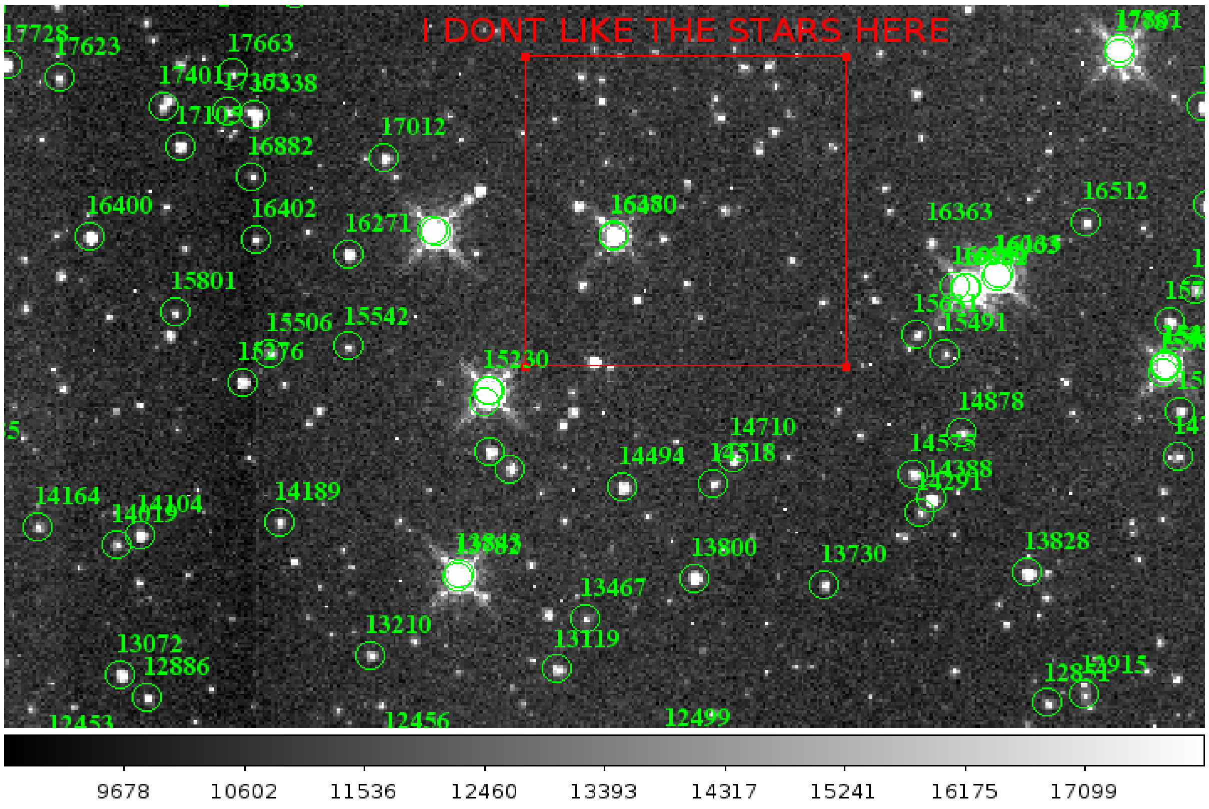
```
#lets make a list of our locations as a tuple of x,y,comment
#we'll cut the list to a smaller area and only include those points whose mag is < -4.
locations=list()
for point in range(0,len(photfile['XCENTER']),1):
    if photfile['MAG'][point] < -4:
        locations.append((photfile['XCENTER'][point],photfile['YCENTER'][point],photfile['ID'][point]))

#so the first item looks like:
In [91]: locations[0]
Out[91]: (807.57500000000005, 2.4449999999999998, 16)
```

Let's open up a DS9 window (if you haven't already) and display your image. This will let us display our source locations and play with them

```
viewer=imexam.connect()
viewer.load_fits('iabf01bzq_flt.fits')
viewer.scale() #scale to DS9 zscale by default
viewer.mark_region_from_array(locations)
```



Now we can get rid of some of the stars by hand and save a new file of locations we like. I did this arbitrarily because I decided I didn't like stars in this part of space. Click on the regions you don't want and delete them from the screen. You can even add more regions of your own choosing.



You can save these new regions to a DS9 style region file, either through DS9 or imexam

```
viewer.save_regions('badstars.reg')
```

**Note:** A future version of the imexam package will make use of the region interpreter currently being developed with astropy for smoother creation and use of parsable regions files

Here is what the saved region file looks like, you can choose to import this file into any future DS9 display of the same image using the `viewer.load_regions()` method. You might also want to parse the file to save just the location and comment information in a separate text file.

```
In [7]: !head badstars.reg
# Region file format: DS9 version 4.1
# Filename: /Users/sosey/ssb/sosey/testme/iabf01bzq_flt.fits[SCI]
global color=green dashlist=8 3 width=1 font="helvetica 10 normal roman" select=1 highlite=1 dash=0
↪fixed=0 edit=1 move=1 delete=1 include=1 source=1
fk5
circle(0:22:38.709,-72:02:50.58,0.677464")
# text(0:22:39.097,-72:02:50.86) font="time 12 bold" text={ 16 }
circle(0:22:36.340,-72:02:58.27,0.677464")
# text(0:22:36.729,-72:02:58.55) font="time 12 bold" text={ 140 }
circle(0:22:29.068,-72:03:20.78,0.677464")
# text(0:22:29.457,-72:03:21.06) font="time 12 bold" text={ 225 }


            . . .


# text(0:22:56.855,-72:04:23.16) font="time 12 bold" text={ 21985 }
circle(0:22:42.791,-72:05:04.04,0.677464")
# text(0:22:43.180,-72:05:04.32) font="time 12 bold" text={ 22002 }
box(0:22:45.694,-72:04:19.19,14.593",13.1774",149.933) # color=red font="helvetica 16 normal roman"
↪text={I DONT LIKE THE STARS HERE}
```

## 6.2 Advanced Usage II - Cycle through objects from a list

This example will step through a list of object locations and center that object in the DS9 window with a narrow zoom so that you can examine it further (think about PSF profile creation options here..)

If you haven't already, start DS9 and load your image into the viewer. I'll assume that you started `DS9` outside of `imexam` and will need to connect to the window first.

```
import imexam
imexam.list_active_ds9()

    DS9 1396283378.28 gs 82a7e75f:53892 sosey

viewer=imexam.connect('82a7e75f:53892')

#A little unsure this is the correct window? Let's check by asking what image is loaded. The image I'm␣
↪working with is iabf01bzq_flt.fits

viewer.get_data_filename()

    '/Users/sosey/ssb/sosey/testme/iabf01bzq_flt.fits'  <-- notice it returned the full pathname to the␣
↪file

viewer.zoomtofit()  <-- let's zoom out  to see the whole image, incase just a small section was loaded
```

Read in your list of object locations, I'll use the same DAOphot targets from the previous example

```
from astropy.io import ascii
reader=ascii.Daophot()
output_locations='iabf01bzq_stars.dat'
photfile=reader.read(output_locations)

#make some cuts on the list

locations=list()
for point in range(0,len(photfile['XCENTER']),1):
    if photfile['MAG'][point] < -4:
        locations.append((photfile['XCENTER'][point],photfile['YCENTER'][point],photfile['ID'][point]))
↪<-- appending tuple to the list
```

Take your list of locations and cycle through each one, displaying a zoomed in section on the `DS9` window and starting `imexam` for each coordinate. I'm just going to go through 10 or so random stars. You can set this up however you like, including using a keystroke as your stopping condition in conjunction with `viewer.readcursor()`

I'll also mark the object we're interested in on the display for reference

```
viewer.zoom(8)
for object in locations[100:110]:
    viewer.panto_image(object[0],object[1])
    viewer.mark_region_from_array(object)
    viewer.imexam()
```

# CHAPTER 7

# Example 4

## 7.1 Load and examine an image CUBE

**Note:** image cubes are currently only supported for the DS9 viewer.

Image cubes can be multi-extension fits files which have multidimensional (> 2) images in any of their extensions. When they are loaded into DS9, a cube dialog frame is opened along with a box which allows the user to control which slices are displayed. Here's what the structure of such a file might look like:

```
astropy.io.fits.info('test_cube.fits')

Filename: test_cube.fits
No.    Name         Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU     215   ()
1    SCI         ImageHDU        13   (1032, 1024, 35, 5)   int16
2    REFOUT      ImageHDU        13   (258, 1024, 35, 5)   int16
```

You can use all the regular imexam methods with this image, including imexam() and the current slice which you have selected will be used for analysis. You can also ask imexam which slice is display, or the full image information of what is in the current frame for your own use (ds9 is just the name I chose, you can call the control object connected to your display window anything)

```
viewer=imexam.connect()
viewer.load_fits('test_cube.fits')
viewer.window.get_filename()

Out[24]: '/Users/sosey/ssb/imexam/test_cube.fits'

viewer.window.get_frame_info()
Out[25]: '/Users/sosey/ssb/imexam/test_cube.fits[SCI,1](0, 0)'
```

Now I'm going to use the Cube dialog to change the slice I'm looking at to (4,14) -> as displayed in the dialog. DS9 displayed 1-indexed numbers, and the fits utitlity behind imexam uses 0-indexed numbers, so expect the return to be off by a value of 1.

Let's ask for the information again:

```
In [26]: viewer.window.get_filename()
Out[26]: '/Users/sosey/ssb/imexam/test_cube.fits'

In [27]: viewer.window.get_frame_info()
Out[27]: '/Users/sosey/ssb/imexam/test_cube.fits[SCI,1](3, 13)'
```

You can ask for just the information about which slice is displayed and it will return the tuple(extension n, ...., extension n-1). The extensions are ordered in row-major form in astropy.io.fits:

```
In [28]: viewer.window.get_slice_info()
Out[28]: (3, 13)
```

The returned tuple contains just which 2d slice is displayed. In our cube image, which is 4D (1032, 1024, 35, 5) == (NAXIS1, NAXIS2, NAXIS3, NAXIS4) in DS9, however in astropy.io.fits this is (5,35,1024,1032) == (NAXIS4, NAXIS3, NAXIS2, NAXIS1)

By default, the first extension will be loaded from the cube fits file if none is specified. If you would rather see another extension, you can load it the same as with simpler fits files:

```
viewer.load_fits('test_cube.fits',extname='REFOUT')
```

CHAPTER 8

# Example 5

## 8.1 Use the imexamine library standalone to create plots without viewing

It's possible to use the imexamine library of plotting functions without loading an image into the viewer. All of the functions take 3 inputs: the x, y, and data array. In order to access the function, first create an imexamine object:

```
from imexam.imexamine import Imexamine
import numpy as np

data=np.random.rand((100,100)) #create a random array thats 100x100 pixels
plots=Imexamine()
```

These are the functions you now have access to:

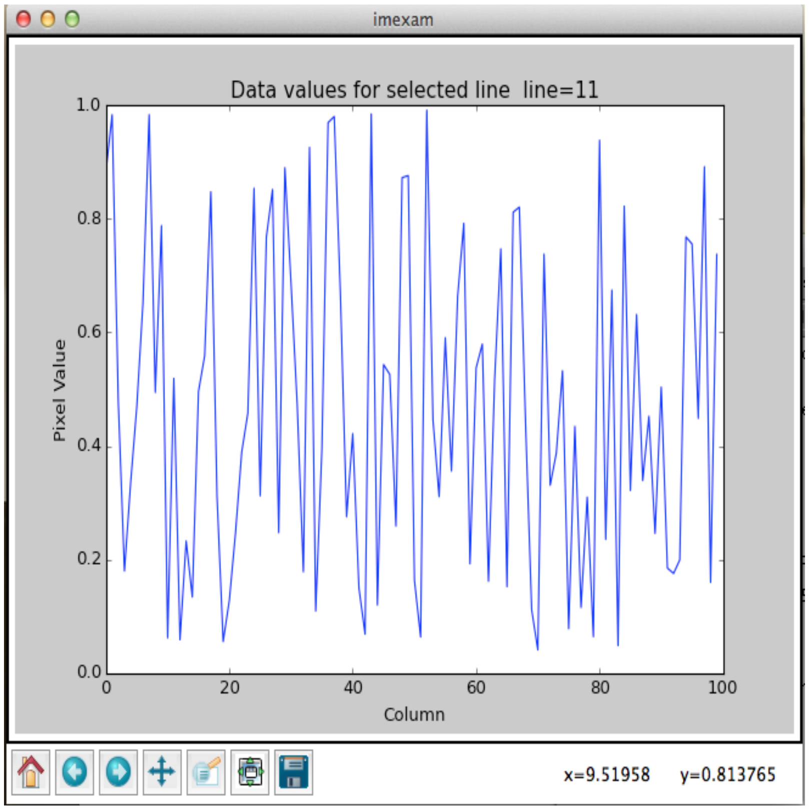
```
plots.aper_phot                plots.contour_plot              plots.histogram_plot           plots.
↪plot_line                 plots.set_colplot_pars         plots.set_surface_pars
plots.aperphot_def_pars        plots.curve_of_growth_def_pars  plots.imexam_option_funcs      plots.
↪plot_name                 plots.set_column_fit_pars      plots.show_xy_coords
plots.aperphot_pars            plots.curve_of_growth_pars      plots.line_fit                 plots.
↪print_options             plots.set_contour_pars         plots.showplt
plots.colplot_def_pars         plots.curve_of_growth_plot      plots.line_fit_def_pars        plots.
↪register                  plots.set_data                 plots.sleep_time
plots.colplot_pars             plots.do_option                 plots.line_fit_pars            plots.
↪report_stat               plots.set_histogram_pars       plots.surface_def_pars
plots.column_fit               plots.gauss_center              plots.lineplot_def_pars        plots.
↪report_stat_def_pars      plots.set_line_fit_pars        plots.surface_pars
plots.column_fit_def_pars      plots.get_options               plots.lineplot_pars            plots.
↪report_stat_pars          plots.set_lineplot_pars        plots.surface_plot
plots.column_fit_pars          plots.get_plot_name             plots.new_plot_window          plots.
↪reset_defpars             plots.set_option_funcs         plots.unlearn_all
plots.contour_def_pars         plots.histogram_def_pars        plots.option_descrip           plots.
↪save_figure               plots.set_plot_name
plots.contour_pars             plots.histogram_pars            plots.plot_column              plots.
↪set_aperphot_pars         plots.set_radial_pars
```

```
```

To create a plot, just specify the method:

```
plots.plot_line(10,10,data)
```

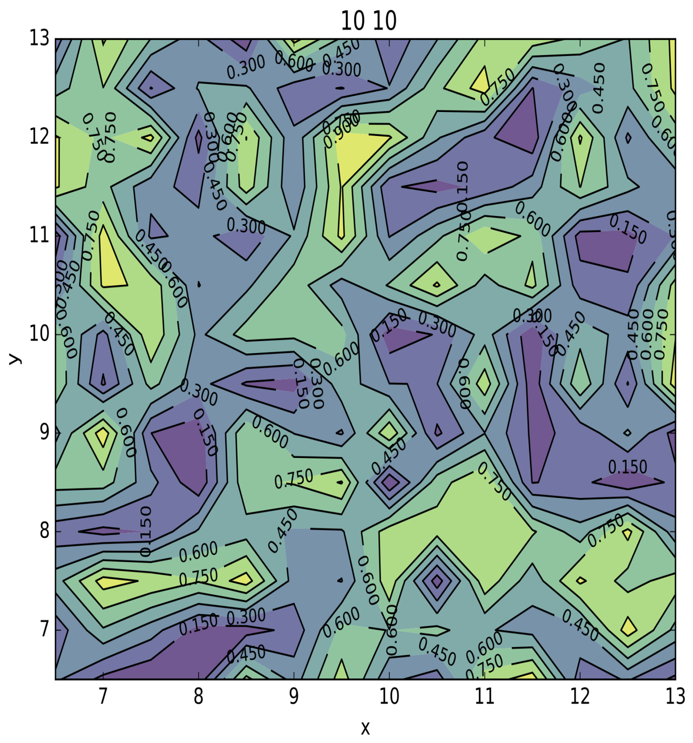
produces the following plot:



You can then save the current plot using the save method:

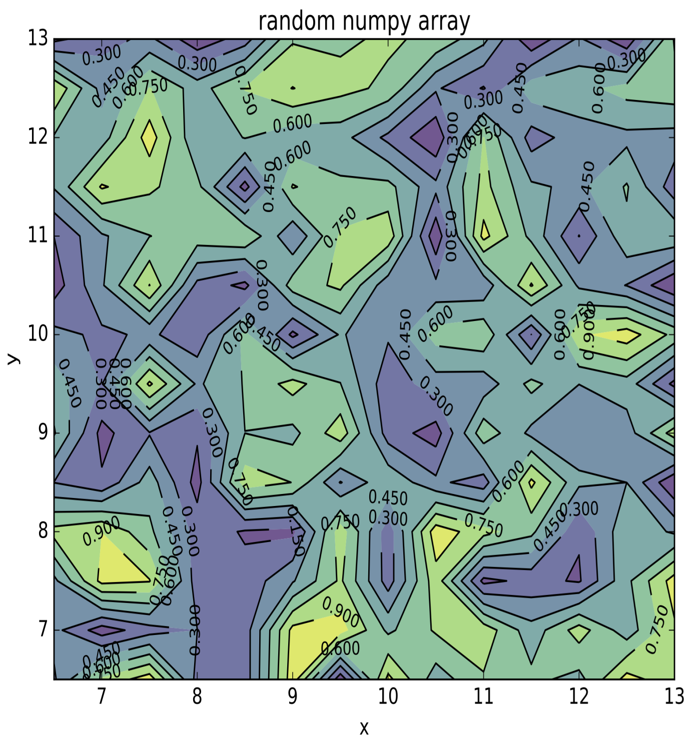
```
plots.contour(10,10,data)
plots.save() # with an optional filename using filename="something.extname"

In [1]: plots.plot_name
Out[2]: 'imexam.pdf'

plots.close() # close the plot window
```

Where the extname specifies the format of the file, ex: jpg or pdf. A pdf file will be the default output, using the curent self.plot_name.



Note that no name is attached to the above contour plot because we plotted a data array. When you are using the plotting class without a viewer, you can attach any title you like by editing the plotting parameters using the dictionary directly::

```
plots.contour_pars['title'][0] = "random numpy array"
```



## 8.2 Return information to variables without plotting

Some of the imexamine() methods are capable of returning their results as data objects. First, lets import some useful things to use in the examples:

```
from astropy.io import fits
from imexam.imexamine import Imexamine

# get my example data from a fits image
data=fits.getdata()
```

Return the fitting result for a line (the same can be done for column_fit):

```
In [1]: plots.line_fit(462, 377, data, genplot=False)
using model: <class 'astropy.modeling.functional_models.Gaussian1D'>
Name: Gaussian1D
Inputs: ('x',)
Outputs: ('y',)
Fittable parameters: ('amplitude', 'mean', 'stddev')
xc=462.438219       yc=377.038640
Out[1]: <Gaussian1D(amplitude=512.5638896303021, mean=462.45102207881393, stddev=-0.6638566150545719)>

# I could have specified an output object here instead and saved the model object:

In [1]: results = plots.line_fit(462, 377, data, genplot=False)
using model: <class 'astropy.modeling.functional_models.Gaussian1D'>
Name: Gaussian1D
Inputs: ('x',)
```

```
Outputs: ('y',)
Fittable parameters: ('amplitude', 'mean', 'stddev')
xc=462.438219       yc=377.038640

In [2]: results
Out[2]: <Gaussian1D(amplitude=512.5638896303021, mean=462.4510220788l393, stddev=-0.6638566150545719)>

In [3]: type(results)
Out[3]:
<class 'astropy.modeling.functional_models.Gaussian1D'>
Name: Gaussian1D
Inputs: ('x',)
Outputs: ('y',)
Fittable parameters: ('amplitude', 'mean', 'stddev')
```

Return the radial profile data points:

```
In [1]: results = plots.radial_profile(462, 377, data, genplot=False)
xc=462.438220       yc=377.038640

# here, results is a tuple of the radius and the flux arrays
In [2]: type(results)
Out[2]: tuple

In [3]: results
Out[3]:
(array([ 0.43991986,  0.56310764,  1.05652729,  1.11346785,  1.12730166,
         1.18083435,  1.4387386 ,  1.56225828,  1.72993907,  1.77404857,
         1.83394967,  1.8756147 ,  2.00971898,  2.0402282 ,  2.08520709,
         2.11462747,  2.43216151,  2.43852579,  2.49490037,  2.50720797,
         2.56207175,  2.56811411,  2.62090222,  2.65022406,  2.73622589,
         2.76432473,  2.99360832,  3.0141751 ,  3.07007625,  3.09013412,
         3.12919301,  3.17820187,  3.22639932,  3.27395339,  3.29213154,
         3.34795643,  3.36181609,  3.41650254,  3.43843675,  3.56198995,
         3.57009352,  3.59167466,  3.68924014,  3.71012829,  3.83595742,
         3.89592694,  3.91565741,  3.95831886,  3.97442453,  3.98552521,
         3.9971748 ,  4.00099637,  4.0623451 ,  4.06610542,  4.0775248 ,
         4.10394097,  4.21436241,  4.25811375,  4.28708374,  4.33010037,
         4.43838783,  4.53773166,  4.541146  ,  4.55813187,  4.56194401,
         4.58853854,  4.63205502,  4.65159003,  4.66197958,  4.67852677,
         4.68183843,  4.71753044,  4.71757631,  4.78260702,  4.85229095,
         4.88403989,  4.96555878,  4.98067583,  4.99306443,  4.99658806,
         5.05766026,  5.06986075,  5.16561429,  5.20137031,  5.2398823 ,
         5.24535309,  5.27513495,  5.30395753,  5.32716192,  5.33548947,
         5.37876614,  5.3848761 ,  5.43835691,  5.43870338,  5.48116519,
         5.52253984,  5.52811091,  5.53651564,  5.56191459,  5.58370969,
         5.59757142,  5.64425498,  5.65248702,  5.65793014,  5.78110428,
         5.80777797,  5.89748546,  5.92363512,  5.94896363,  5.97744528,
         5.98777194,  6.00070036,  6.03626122,  6.04170629,  6.05451954,
         6.06471496,  6.09265553,  6.09993812,  6.10748513,  6.13239687,
         6.16254603,  6.17042707,  6.19224411,  6.20754751,  6.22957178,
         6.23733343,  6.30103604,  6.33772298,  6.43833558,  6.44070886,
         6.48849245,  6.50959949,  6.51230262,  6.52146032,  6.5595647 ,
         6.56189413,  6.63183044,  6.64347305,  6.65679268,  6.71458743,
         6.72804634,  6.73034962,  6.73980232,  6.75327507,  6.77383526,
         6.79689127,  6.82830694,  6.84864187,  6.87117266,  6.87342797,
         6.8817999 ,  6.94435706,  6.9488506 ,  6.97513961,  6.98399121,
         7.01080949,  7.08663012,  7.10837617,  7.11926989,  7.13440215,
         7.19907049,  7.23120275,  7.3613401 ,  7.37600509,  7.41364442,
         7.41776616,  7.43206628,  7.45308634,  7.49419535,  7.50475127,
         7.50650756,  7.55930201,  7.56802554,  7.60008443,  7.66481157,
         7.70503555,  7.76414132,  7.81964293,  8.06920371,  8.12646314,
         8.12808509,  8.15298819,  8.17548548,  8.20966328,  8.22630274,
         8.25580581,  8.27314042,  8.32288269,  8.77430839,  8.8269951 ,
         8.83372905,  8.86536955,  8.91032754,  8.91751826,  9.48215209,
         9.56647781]),
 array([ 408.87057495,   41.23228073,   91.90717316,   48.38606262,
         112.11755371,   64.6014328 ,  361.9876709 ,    7.88528776,
          76.15605927,   92.4905777 ,    5.74170589,    8.54299355,
          37.25744629,   17.17868423,   41.94879532,   29.16669464,
          25.11438942,   41.24355316,   31.41527748,    2.35880852,
           2.51266503,    3.61639667,   31.96870041,   47.24103928,
           1.86882472,    2.25345397,    3.43679786,    2.95230484,
           7.01711893,    4.25243187,   10.45163536,   15.06377506,
           2.06799817,    1.55962014,    3.2355001 ,    3.58886528,
           4.77823544,    2.61030412,    6.15013599,    2.26734257,
           3.79847336,    5.18475103,    2.02961087,    1.86825836,
           2.26850033,    1.98072493,    2.40412855,    2.35658216,
           2.2638216 ,    1.48555958,    2.15530491,    1.40320516,
           2.42260337,    3.59516048,    1.49309242,    2.70001984,
           1.35936797,    2.50372696,    1.99834633,    2.1075139 ,
           2.10088921,    3.91031456,    1.40116227,    1.58724546,
           1.64244962,    4.27553177,    2.86458731,    2.07594514,
           1.24715221,    1.55571783,    3.28257489,    1.08224833,
           1.99108934,    1.28673184,    2.22391272,    2.01411462,
           1.27933741,    2.57424259,    2.27977562,    1.34119225,
           2.46366167,    2.04145074,    2.27879167,    3.32902098,
           2.0256803 ,    3.04667783,    3.214293  ,    2.71672273,
           1.18290937,    3.39013147,    2.61141396,    1.24552131,
           2.7109127 ,    1.20734   ,    1.065956  ,    2.0110569 ,
           2.63785267,    2.08804011,    1.23607028,    1.53105474,
           2.9585526 ,    0.92856985,    1.70498252,    0.98702717,
           3.00484014,    2.96310997,    1.10799265,    1.02301562,
           2.59040713,    1.55507016,    1.1307373 ,    1.46614468,
           3.7729485 ,    0.8989926 ,    1.81300449,    1.49930847,
           0.97070342,    3.58096623,    1.45315814,    1.37846851,
           1.22037327,    2.02710581,    3.06499743,    1.60018504,
           3.15293145,    1.34511912,    1.04039967,    0.94602752,
           1.5991565 ,    1.11648059,    0.90265507,    1.25119698,
           1.32048595,    1.331002  ,    1.26167858,    0.81102282,
           0.99124312,    0.76625013,    1.42264056,    1.41574192,
           1.67775941,    1.15894651,    1.19685972,    0.99676919,
           1.16761708,    1.20492256,    1.09948123,    1.0989542 ,
           0.92135239,    0.89912277,    1.15777898,    1.07870626,
           1.32945871,    1.06859183,    0.77524334,    1.4281857 ,
           1.05790067,    1.08861005,    1.03711545,    1.00277674,
           1.11795783,    1.04079187,    1.77855933,    0.875655  ,
           1.70616186,    0.95955884,    1.2846061 ,    0.9819802 ,
           1.09096873,    1.12618971,    2.52278042,    1.14947557,
           2.55132389,    1.16845107,    1.0366509 ,    1.03310716,
           0.76811701,    0.98454052,    1.38449657,    1.41319823,
           1.30402267,    1.26531458,    0.88282102,    1.33250594,
           0.86149669,    1.13119161,    0.89653128,    1.47101414,
           2.82045436,    2.37812138,    0.82307637,    1.3075676 ,
           1.45813155,    1.30278611,    1.60565269,    1.01857305], dtype=float32))
```

Return the curve of growth points:

```
In [1]: results = plots.curve_of_growth(462, 377, data, genplot=False)
xc=462.438220       yc=377.038640

at (x,y)=462,377
radii:[1 2 3 4 5 6 7 8]
flux:[406.65712375514534, 1288.8955810496341, 1634.0235081082126, 1684.5579429185905, 1718.118845192796,
↪ 1785.265260722455, 1801.8561084128257, 1823.21222063562]

In [2]: type(results)
Out[2]: tuple

In [3]: results
Out[3]:
(array([1, 2, 3, 4, 5, 6, 7, 8]),
 [406.65712375514534,
  1288.8955810496341,
  1634.0235081082126,
  1684.5579429185905,
  1718.118845192796,
  1785.265260722455,
  1801.8561084128257,
  1823.21222063562])

 # the typle can be separated into it's parts
 radius, flux = results
```

Return the histogram information as a tuple of values and bin edges:

```
In [1]: counts, bins = plots.histogram(462, 377, data, genplot=False)

In [2]: counts
Out[2]:
array([372,   7,   1,   1,   1,   0,   1,   3,   1,   2,   1,   2,   0,
         0,   0,   1,   0,   0,   1,   0,   0,   0,   2,   0,   0,   0,
         0,   1,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0]

In [3]: bins
Out [3]:
 array()[   0.58091092,    4.66380756,    8.7467042 ,   12.82960084,
          16.91249748,   20.99539412,   25.07829076,   29.1611874 ,
          33.24408404,   37.32698068,   41.40987732,   45.49277396,
          49.5756706 ,   53.65856725,   57.74146389,   61.82436053,
          65.90725717,   69.99015381,   74.07305045,   78.15594709,
          82.23884373,   86.32174037,   90.40463701,   94.48753365,
          98.57043029,  102.65332693,  106.73622357,  110.81912021,
         114.90201685,  118.98491349,  123.06781013,  127.15070677,
         131.23360341,  135.31650005,  139.39939669,  143.48229333,
         147.56518997,  151.64808661,  155.73098325,  159.81387989,
         163.89677653,  167.97967317,  172.06256981,  176.14546645,
         180.22836309,  184.31125973,  188.39415637,  192.47705302,
         196.55994966,  200.6428463 ,  204.72574294,  208.80863958,
         212.89153622,  216.97443286,  221.0573295 ,  225.14022614,
         229.22312278,  233.30601942,  237.38891606,  241.4718127 ,
         245.55470934,  249.63760598,  253.72050262,  257.80339926,
         261.8862959 ,  265.96919254,  270.05208918,  274.13498582,
         278.21788246,  282.3007791 ,  286.38367574,  290.46657238,
         294.54946902,  298.63236566,  302.7152623 ,  306.79815894,
         310.88105558,  314.96395222,  319.04684886,  323.1297455 ,
         327.21264215,  331.29553879,  335.37843543,  339.46133207,
         343.54422871,  347.62712535,  351.71002199,  355.79291863,
         359.87581527,  363.95871191,  368.04160855,  372.12450519,
         376.20740183,  380.29029847,  384.37319511,  388.45609175,
         392.53898839,  396.62188503,  400.70478167,  404.78767831,
         408.87057495])
```

CHAPTER 9

# Software Dependencies

- Astropy (for some analysis functions)
- photutils (for photometry)
- matplotlib (for plotting)
- DS9 (image display - optional) * XPA: https://github.com/ericmandel/xpa
- Ginga (image display - optional )

astropy >= 1.0

python >= 2.7

numpy >= 1.7.0

**photutils > 0.2**
This must be installed to enable the photometry options for imexam() but it is not required

**Ginga**
This must be installed in order to use the Ginga displays instead of DS9. Windows users who install from source should also install Ginga if they wish to use an image viewer since the DS9 and XPA compiles will be disabled. It's possible to compile and install the XPA and DS9 from source, but not with typical default software.

Using ginga has the advantage that the imexam() loop is now event driven.

You can issue the viewer.imexam() command to print out the available examination command keys. The user can then press the "i" key while the mouse is in the graphics window, all subsequent key-presses will be grabbed without blocking your terminal command line. If you wish to turn of the imexam keys you can press either the "i" key a second time or the "q" key. A notification message will appear on screen that imexam mode has either started or stopped.

If you are using the Ginga HTML5 widget under python3 in the Jupyter notebook you should also install Pillow to get the correct image viewing.

CHAPTER 10

# IRAF imexamine capabilities

These are the capabilities of the IRAF version of the imexam task, called with **imexamine [input [frame]]**, which lives in images.tv.imexamine. The following are imexamines input options:

- **input** is an optional list of images to be examined. If specified, images are examined in turn, displaying them automatically. If no images are specfied the images currently loaded into the image display are examined.
- **output** contains the rootname for output images created with the "t" key. If no name is specified then the name of the input image is used. A three digit numver is appended to the rootname, such as ".001", starting with 1 until no image is found with that name. Successive output images are numbered sequentially
- **ncoutput** and **nloutput** are the size of the output image created when the "t" key is pressed, where the output image is centered on the cursor location
- **frame** specifies which frame should be used
- **logfile** is the filename which records output of the commands producing text, if no filename is given no logfile will be produced
- **defkey** is the default key for cursor x-y input list. This key is applied to input cursor lists which do not have a cursor key specified. It is used to repetitively apply a cursor command to a list of positions typically obtained from another task
- **allframes**, if true then images from an input list are loaded by cycling through the available frames, otherwise the last frame loaded is reused
- **nframes** is then number of display frames to use when automatically loading images. It should not exceed the number of frames provided by the display device. If the number of frames is set to 0 then the task will query the display device to determine how many frames are currently allocated. New frames may be allocated during program execution by displaying images with the 'd' key.
- **ncstat, nlstat** correlate with the statistics command which computes values from a box centered on the specified cursor position with the number of columns and lines given by these parameters.

The following is a list of available cursor and colon commands while imexamine is active in the display, many but not all are available in this python imexam package:

```
                            -- IMEXAMINE COMMANDS --

                           CURSOR KEY COMMAND SUMMARY

? Help              h Histogram         p Previous frame    x Coordinates
a Aperture Sum      i Image cursor      q Quit              y Set origin
b Box coords        j Line gauss fit    r Radial plot       z Print grid
c Column plot       k Col gauss fit     s Surface plot      , Quick phot
d Load display      l Line plot         t Output image      . Quick prof fit
e Contour plot      m Statistics        u Vector plot
f Redraw            n Next frame        v Vector plot
g Graphics cursor   o Overplot          w Toggle logfile


                             COLON COMMAND SUMMARY

allframes    ceiling      iterations   naverage     pointmode    width
angh         center       label        nbins        radius       x
angv         constant     logfile      ncolumns     round        xformat
autoredraw   dashpat      logx         ncontours    rplot        xlabel
autoscale    defkey       logy         ncoutput     select       xorder
background   eparam       magzero      ncstat       szmarker     y
banner       fill         majrx        nhi          ticklabel    yformat
beta         fitplot      majry        nlines       title        ylabel
boundary     fittype      marker       nloutput     top_closed   yorder
box          floor        minrx        nlstat       unlearn      z1,z2
buffer       interval     minry        output       wcs          zero


                           OUTPUT OF 'a' AND 'r' KEYS

The 'a' key and logfile output has column labels and each object has one
line of measurements in the logfile and two lines on the terminal.  The 'r'
key shows only the second line on the status line and the information from
the first line is in the graph title.  The first line contains the x and y
center coordinates and optional world coordinates.  The second line
contains the aperture magnitude and flux, the estimated background sky, the
profile fit peak, the ellipticity and position angle from the moment
analysis, and four estimates of the profile width.  The four estimates are
from the moment analysis, the full-width enclosing half the flux, the
profile fit, and a direct estimate of the full width at half-maximum.


                              CURSOR KEY COMMANDS

?        Print help
a        Aperture radial photometry measurement (see above for output)
b        Box coordinates for two cursor positions - c1 c2 l1 l2
c        Column plot
d        Load the image display
e        Contour plot
f        Redraw the last graph
g        Graphics cursor
h        Histogram plot
i        Image cursor
j        Fit 1D gaussian to image lines
k        Fit 1D gaussian to image columns
l        Line plot
```

```
m       Statistics
           image[section] npixels mean median stddev min max
n       Next frame or image
o       Overplot
p       Previous frame or image
q       Quit
r       Radial profile plot (see above for output)
s       Surface plot
t       Output image centered on cursor (parameters output, ncoutput, nloutput)
u       Centered vector plot from two cursor positions
v       Vector plot between two cursor positions
w       Toggle write to logfile
x       Print coordinates
           col line pixval [xorign yorigin dx dy r theta]
y       Set origin for relative positions
z       Print grid of pixel values - 10 x 10 grid
,       Quick profile photometry measurement (Gaussian or Moffat)
.       Quick radial profile plot and fit (Gaussian or Moffat)

                               COLON COMMANDS

Explicit image coordinates may be entered using the colon command syntax:

        :column line key

where column and line are the image coordinates and the key is one
of the cursor keys.  A special syntax for line or column plots is also
available as :c# or :l# where # is a column or line and no space is
allowed.

Other colon commands set or show parameters governing the plots and other
features of the task.  Each graph type has it's own set of parameters.
When a parameter applies to more than one graph the current graph is assumed.
If the current graph is not applicable then a warning is given.  The
"eparam" and "unlearn" commands may be used to change many parameters and
without an argument the current graph parameters are modified while with
the graph key as an argument the appropriate parameter set is modified.
In the list below the graph key(s) to which a parameter applies are shown.

allframes               Cycle through all display frames to display images
angh        s           Horizontal angle for surface plot
angv        s           Vertical angle for surface plot
autoredraw  cehlrsuv.   Automatically redraw graph after colon command?
autoscale   h           Adjust number of histogram bins to avoid aliasing
axes        s           Draw axes in surface plot?
background  jkr.        Subtract background for radial plot and photometry?
banner      cehjklrsuv. Include standard banner on plots?
beta        ar                Moffat beta parameter (INDEF to fit or value to fix)
boundary    uv          Boundary extension type for vector plots
box         cehjklruv.  Draw box around graph?
buffer      r.          Buffer distance for background subtraction
ceiling     es          Data ceiling for contour and surface plots
center      jkr.        Find center for radial plot and photometry?
constant    uv          Constant value for boundry extension in vector plots
dashpat     e           Dash pattern for contour plot
eparam      cehjklrsuv. Edit parameters
fill        e           Fill viewport vs enforce unity aspect ratio?
fitplot     r           Overplot profile fit on data?
```

```
fittype     ar          Profile fitting type (gaussian|moffat)
floor       es          Data floor for contour and surface plots
interval    e           Contour interval (0 for default)
iterations  ar          Iterations on fitting radius
label       e           Draw axis labels for contour plot?
logfile                 Log file name
logx        chjklruv.   Plot x axis logrithmically?
logy        chjklruv.   Plot y axis logrithmically?
magzero     r.          Magnitude zero for photometry
majrx       cehjklruv.  Number of major tick marks on x axis
majry       cehjklruv.  Number of major tick marks on y axis
marker      chjklruv.   Marker type for graph
minrx       cehjklruv.  Number of minor tick marks on x axis
minry       cehjklruv.  Number of minor tick marks on y axis
naverage    cjkluv      Number of columns, lines, vectors to average
nbins       h           Number of histogram bins
ncolumns    ehs         Number of columns in contour, histogram, or surface plot
ncontours   e           Number of contours (0 for default)
ncoutput                Number of columns in output image
ncstat                  Number of columns in statistics box
nhi         e           hi/low marking option for contours
nlines      ehs         Number of lines in contour, histogram, or surface plot
nloutput                Number of lines in output image
nlstat                  Number of lines in statistics box
output                        Output image root name
pointmode   chjkluv     Plot points instead of lines?
radius      r.          Radius of object aperture for radial plot and photmetry
round       cehjklruv.  Round axes to nice values?
rplot       jkr.        Radius to plot in 1D and radial profile plots
select                  Select image or display frame
sigma       jk          Initial sigma for 1D gaussian fits
szmarker    chjklruv.   Size of marks for point mode
ticklabels  cehjklruv.  Label ticks?
title       cehjklrsuv. Optional title for graph
top_closed  h           Close last bin of histogram
unlearn     cehjklrsuv. Unlearn parameters to default values
wcs                     World coordinate system for axis labels and readback
width       jkr.        Width of background region
x [min max] chjklruv.   Range of x to be plotted (no values for autoscaling)
xformat                        Coordinate format for column world coordinates
xlabel      cehjklrsuv. Optional label for x axis
xorder      jkr.        X order of surface for background subtraction
y [min max] chjklruv.   Range of y to be plotted (no values for autoscaling)
yformat                        Coordinate format for line world coordinates
ylabel      cehjklrsuv. Optional label for y axis
yorder      r.          Y order of surface for background subtraction
z1          h           Lower intensity value limit of histogram
z2          h           Upper intensity value limit of histogram
zero        e           Zero level for contour plot
```

CHAPTER 11

# Comparison with the IRAF verison of imexamine

The following is a comparison of the outputs, returned values and user options for this module versus imexamine in IRAF

- All plot types are replicated between the codes, though they may be rendered differently. The images below are representative of the basic plots from each package.
- The same user plot options, as in the rimexam, cimexam etc. type IRAF param files are replicated to their useful extent using python dictionaries for each imexamine key.
- Colormaps and point styles for the matplotlib plots may be changed by the user through the `imexam` key default dictionaries
- In `imexam`, once the plot is displayed on the screen, you can zoom in and out using the controls in the plotting window.
- `imexam` allows users to register their own analysis functions
- `imexam` does not attempt to replicate the colon command interaction, if users wish to change the plot settings they should exit the imexam() method, reset the options and call it again.
- all of the imexam() functions in `imexam` can be called by themselves if you supply an x,y coordinate

How do the numerical results compare with the IRAF version? This is a little harder to judge with cursor centering. Visual comparison of the resulting plots shows good agreement, as well as some random checks of the photometry and statistical return methods.

## 11.1 Statistical returns

IRAF "m" key:

::

–> imexam

# SECTION NPIX MEAN MEDIAN STDDEV MIN MAX

[584:588,697:701] 25 46533. 51314. 10281. 21215. 56186.

imexam “m” key (with cursor location flooring):

```
[583:588,695:700] median: 51458.000000
```

imexam only shows one statistic at a time. The same function call may be used to show the results from *any* valid numpy function, it will return an attribute error for invalid functions. For example, if you edit the defaults dictionary for the “m” key:

```
viewer.mimexam()

    {'function': ['report_stat'],
     'region_size': [5, 'region size in pixels to use'],
     'stat': ['median',
     'which numpy stat to return [median,min,max...must map to a numpy func]']}

viewer.exam.report_stat_pars["stat"][0] = "max"  <---- will report np.max for the array

    [584:589,695:700] amax: 56186.000000

viewer.exam.report_stat_pars["stat"][0] = "mean"  <---- will report np.mean for the array

    [583:588,694:699] mean: 45412.878906

viewer.exam.report_stat_pars["stat"][0]="std"

    [583:588,694:699] std: 10706.179688
```

## 11.2 Aperture Photometry

IRAF “a” key:

```
#   COL    LINE   COORDINATES
#     R    MAG    FLUX     SKY    PEAK    E   PA BETA ENCLOSED   MOFFAT DIRECT
585.81  698.16 585.81 698.16
17.51   8.86 2.858E6  10840.  45443. 0.03  -64 8.32     5.23     7.10   5.83
```

imexam “a” key (using the defaults):

```
xc=586.138728   yc=697.990516
x       y       radius  flux     mag(zpt=25.00)  sky     fwhm
586.14  697.99  5       1508664.63      9.55    11160.89        6.03
```

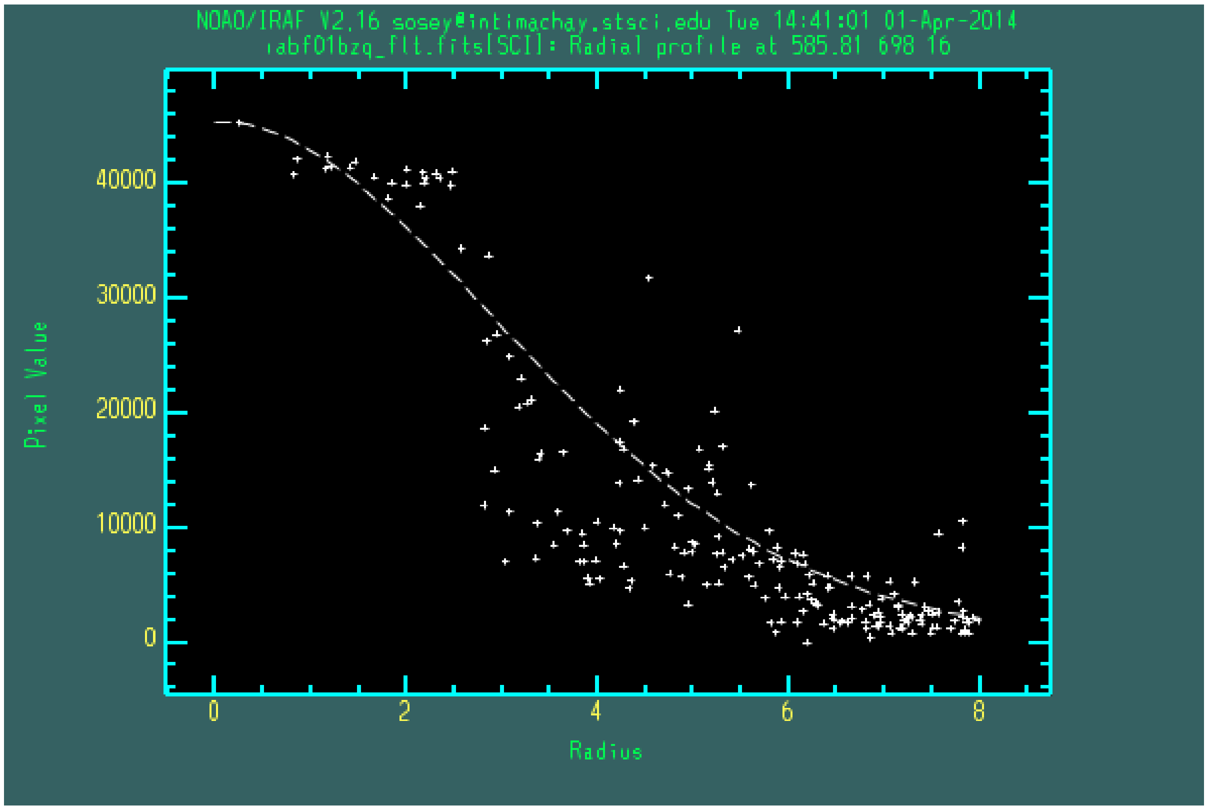
The “xc” and “yc” returns are the gaussian fit centers, as well as the FWHM from the fit. If we set the values to be similar to what IRAF.imexamine used, we can see the numbers are closer, the radius for the apertures are floored though before being sent to photutil:

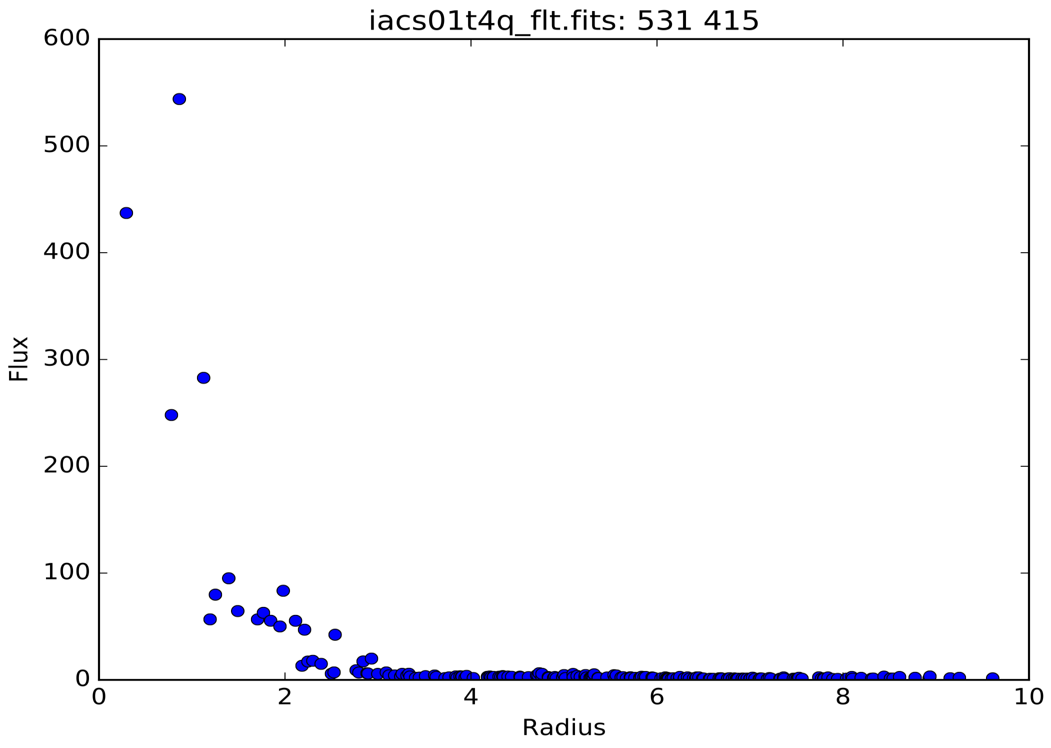
```
viewer.aimexam()

{'center': [True, 'Center the object location using a 2d gaussian fit'],
 'function': ['aperphot'],
 'radius': [5, 'Radius of aperture for star flux'],
 'skyrad': [15, 'Distance to start sky annulus is pixels'],
 'subsky': [True, 'Subtract a sky background?'],
 'width': [5, 'Width of sky annulus in pixels'],
 'zmag': [25.0, 'zeropoint for the magnitude calculation']}
```

```
viewer.exam.aperphot_pars["radius"][0]=17.5

xc=586.213790    yc=697.501845
x        y        radius   flux      mag(zpt=25.00)   sky      fwhm
586.21   697.50   17       2565167.11      8.98     11162.83        6.02
```

## 11.3 Radial Profile Plot

The fit profile of the star out to the specified radius. Users can look at the fit profile of the star using the 1D gaussian option. By default, `imexam` prints the data point values to the screen.



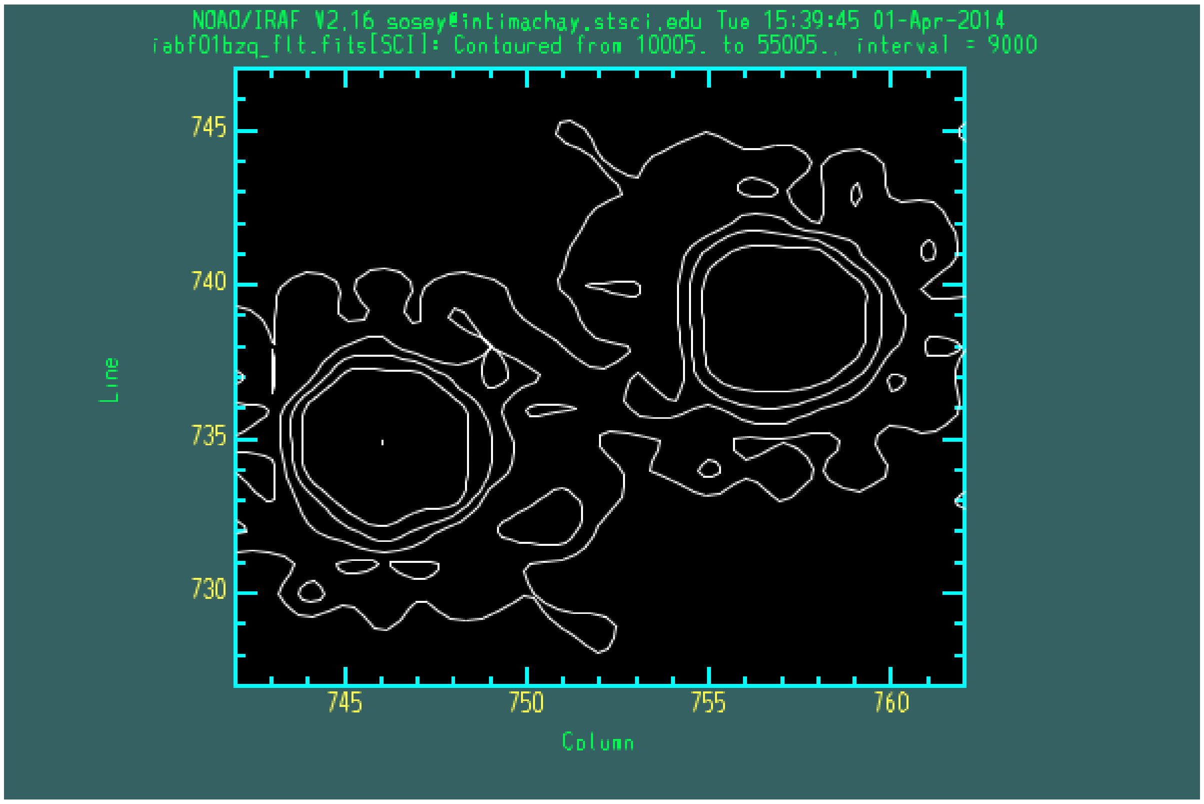


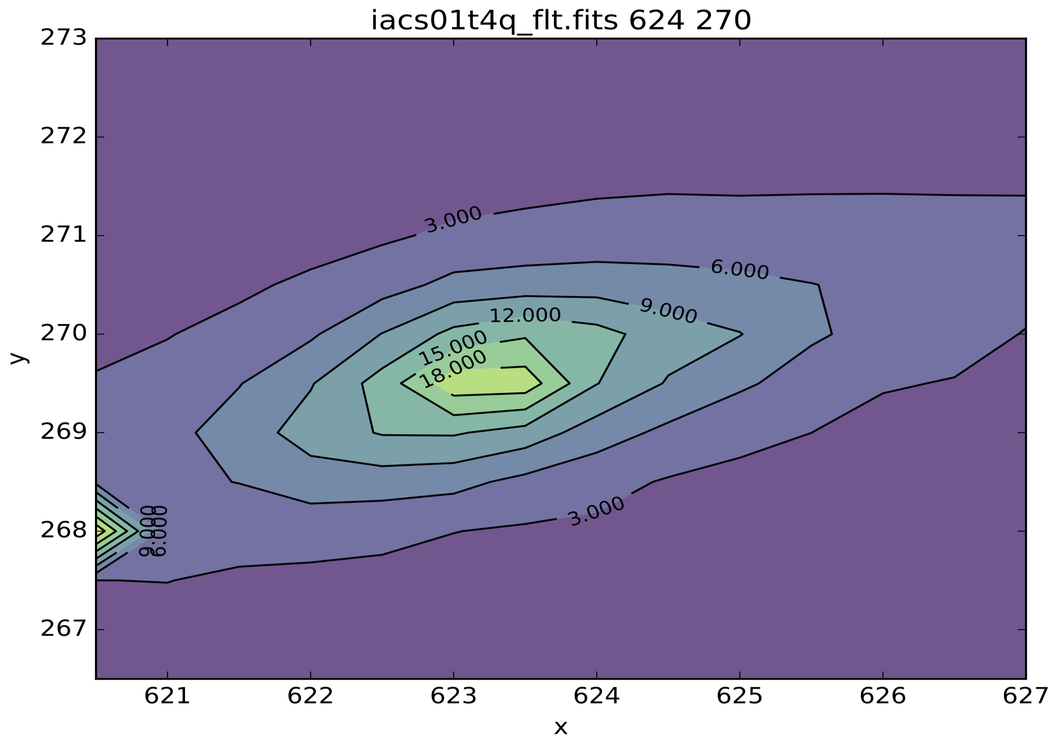
imexam prints the plotted data to the screen

```
pressed: r
xc=655.659205   yc=698.937124
Sky per pixel: 0.7021602984249302 using(rad=10.0->15.0)

at (x,y)=655,698
radii:[0 1 2 3 4 5 6 7 8 9]
flux:[  74.23025852  153.66757441   60.17693806    9.7988813     7.10537578
    9.08464076    3.1673068     2.92777784    0.26435121    0.18440688]
```

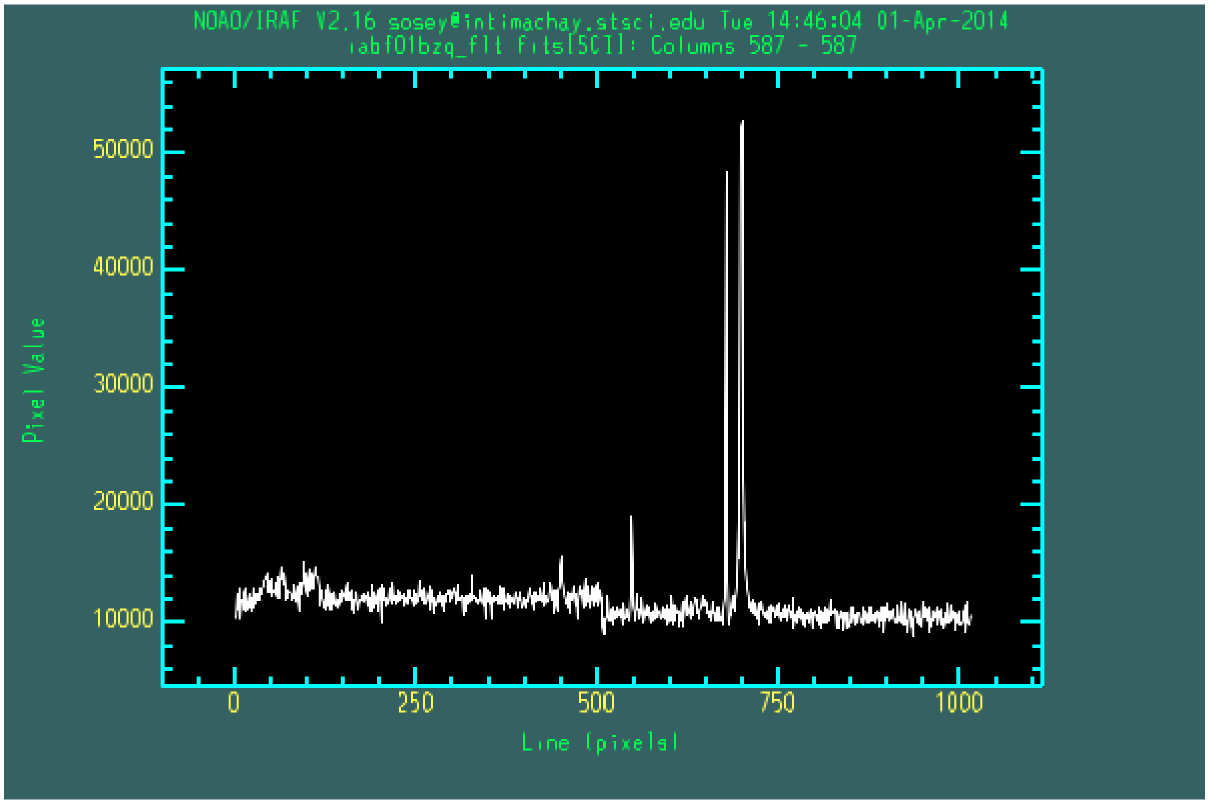
## 11.4 Contour plot

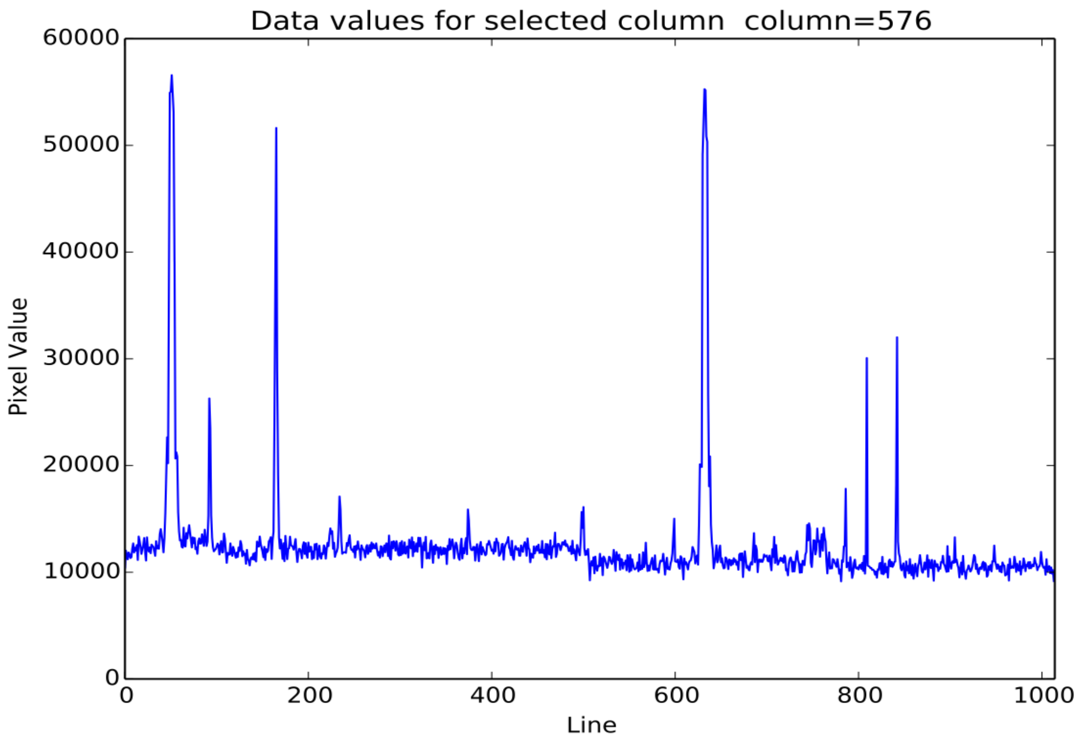
Note the added availability in this package for labeling the contours


NOAO/IRAF V2.16 sosey@intimachay.stsci.edu Tue 15:39:45 01-Apr-2014
iabf01bzq_flt.fits[SCI]: Contoured from 10005. to 55005., interval = 9000
Line
745
740
735
730
745
750
755
760
Column

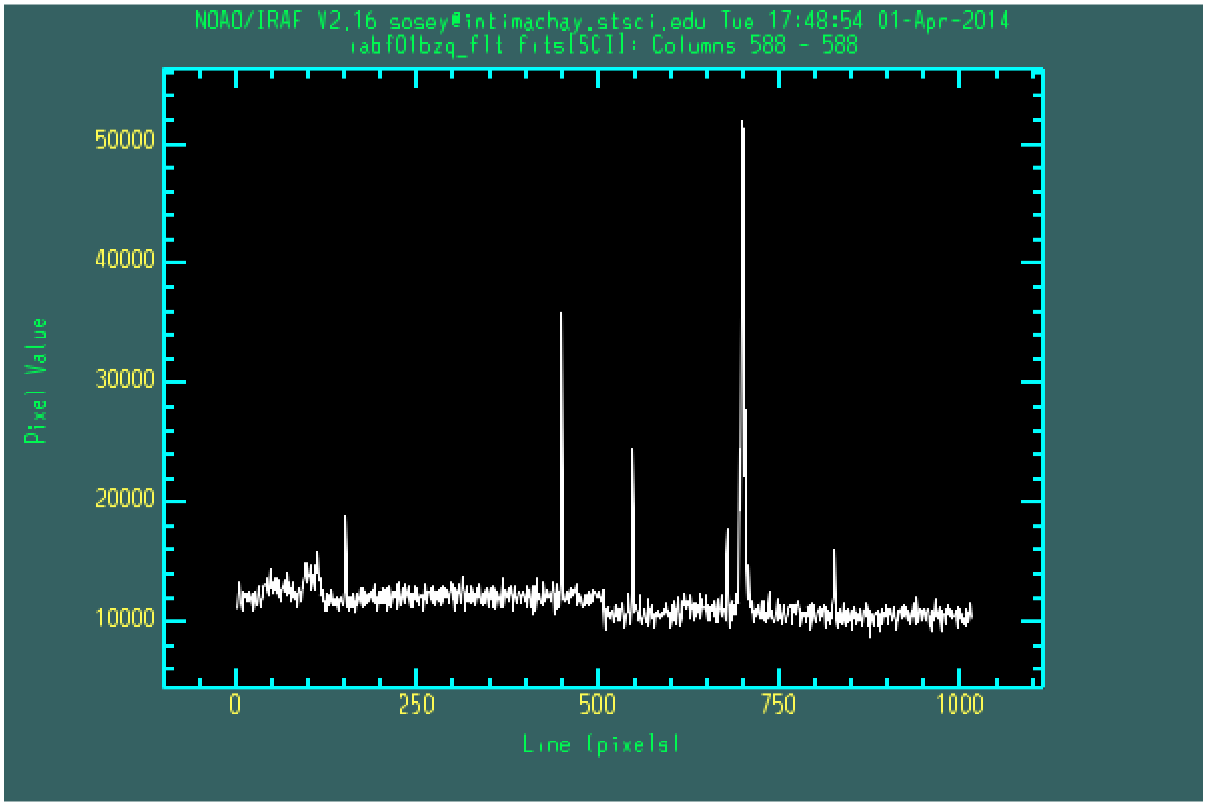

iacs01t4q_flt.fits 624 270
273
272
271
270
269
268
267
y
621
622
623
624
625
626
627
x
3.000
6.000
9.000
12.000
15.000
18.000
3.000
9.000
6.000

## 11.5 Column and Line plots



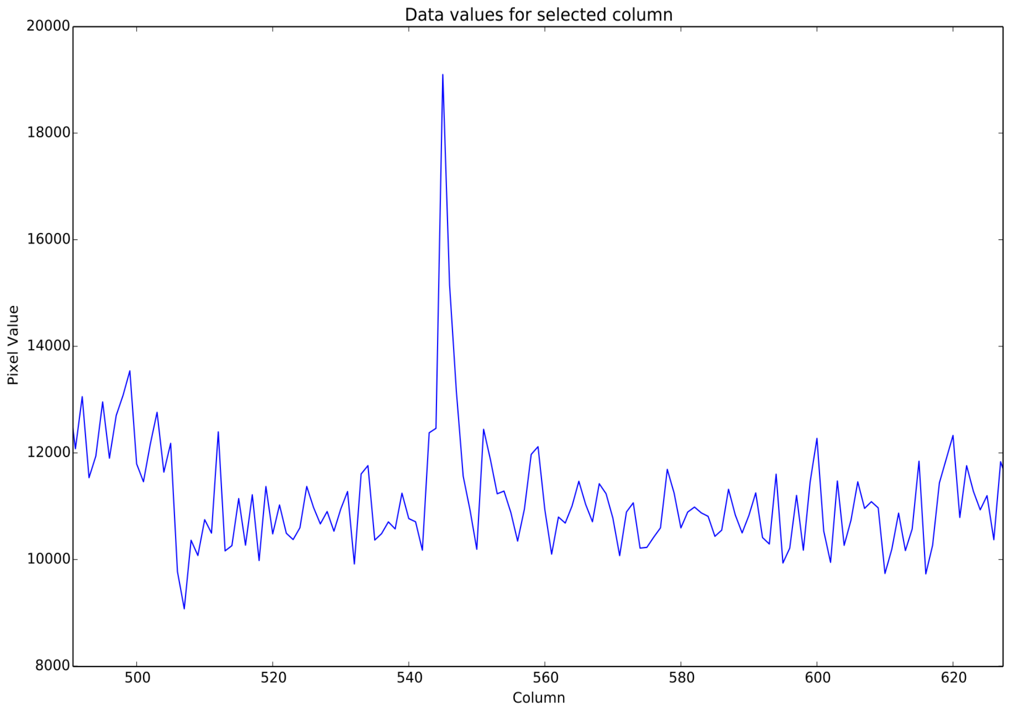


Keep in mind that python is 0-index and IRAF returns 1-index arrays, so the equivalent IRAF plot of 587 is really 588:

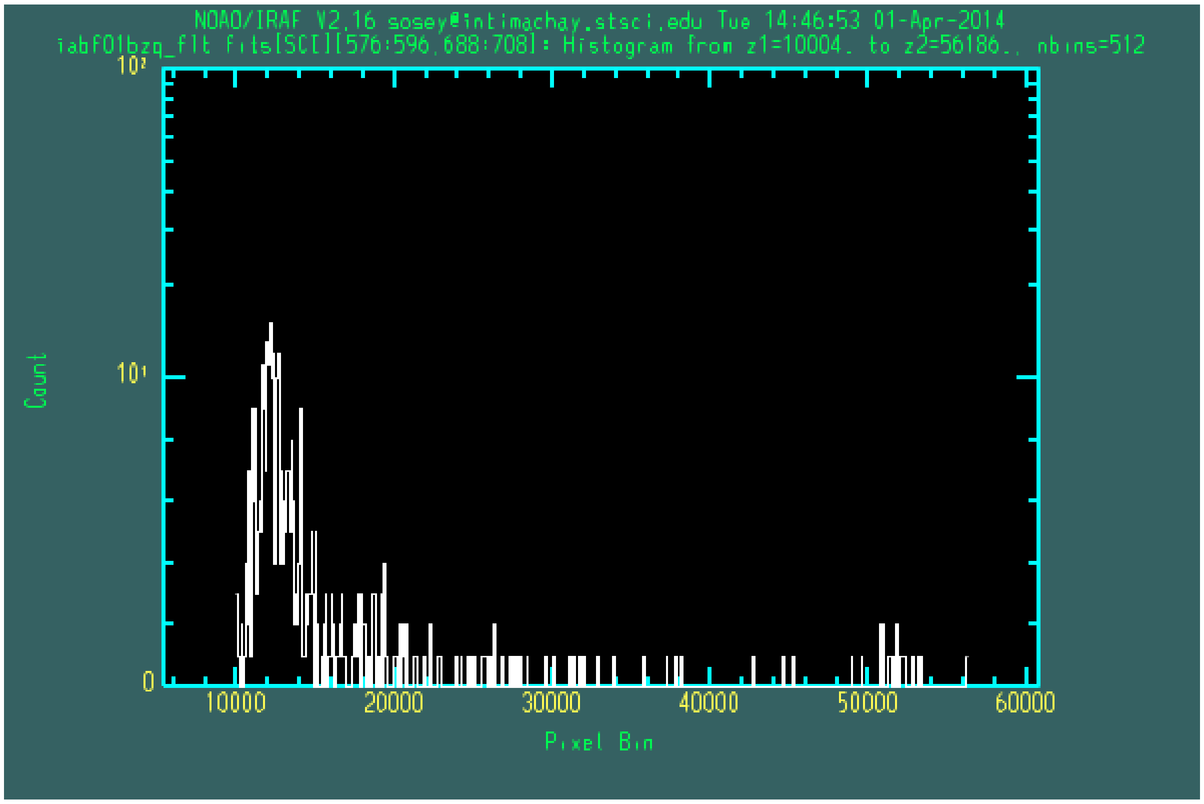


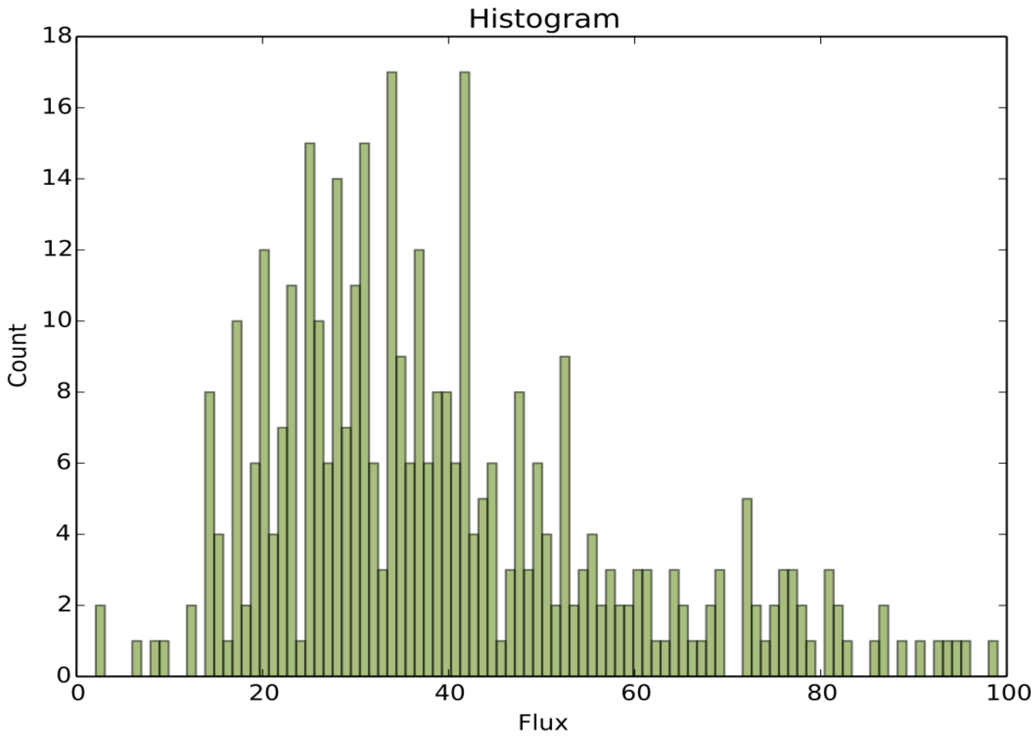
.

An added benefit in the python package is that you can zoom in and out of the plots using the window controls, below is a zoomed in area of the column plot as it appears in the window:


Data values for selected column
Pixel Value
Column
20000
18000
16000
14000
12000
10000
8000
500
520
540
560
580
600
620

## 11.6 Histogram plots



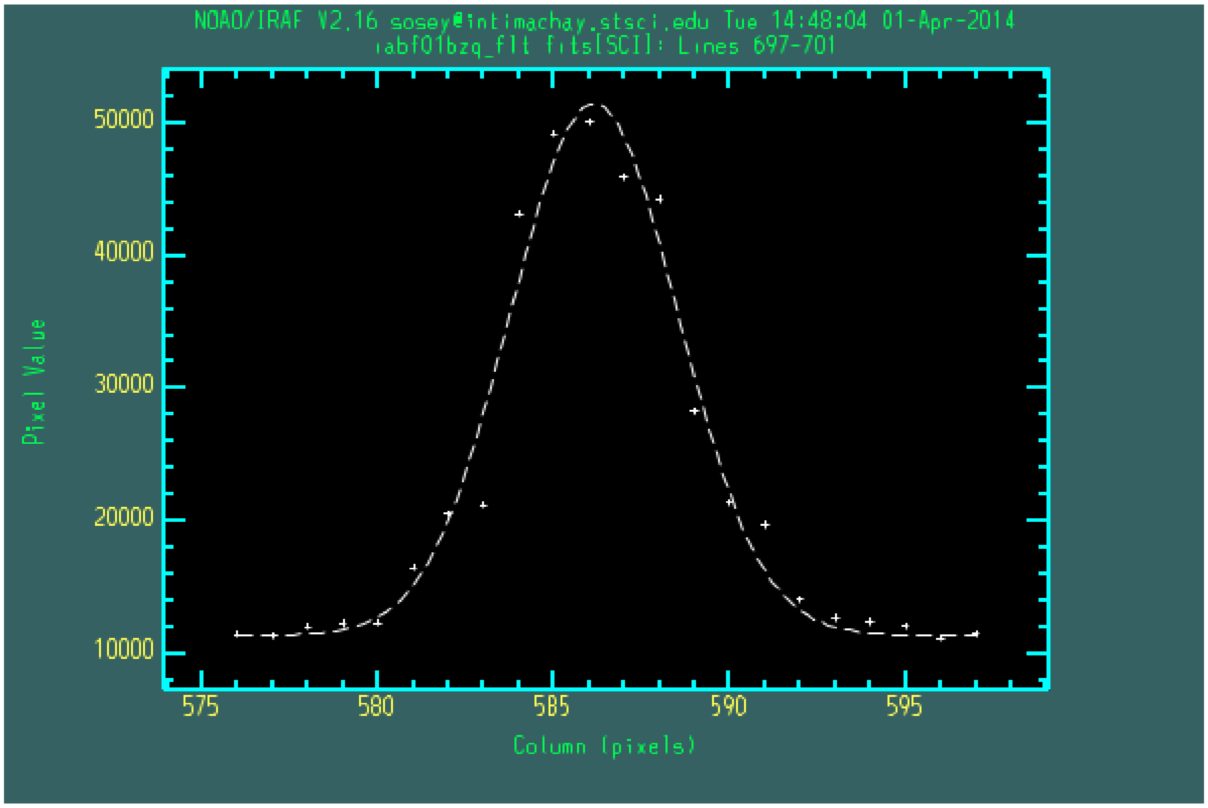


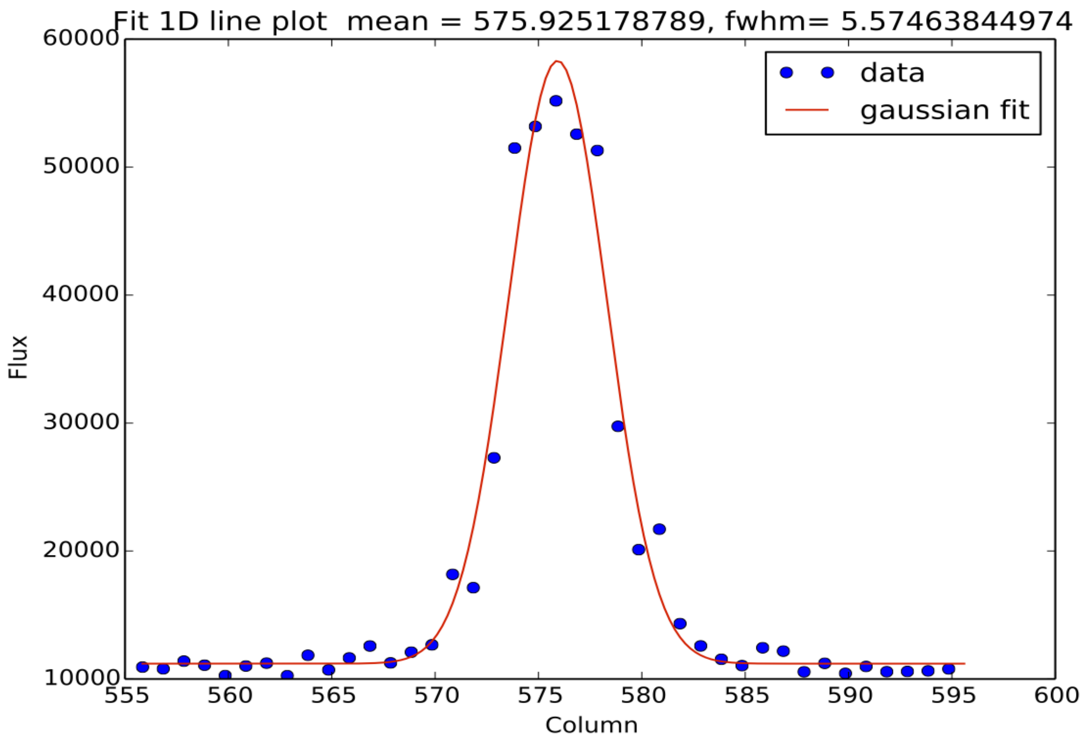
imexam prints bin information to the screen

```
100 bins
```

## 11.7 1D Gaussian plots

These plots are representative for both the column and line versions



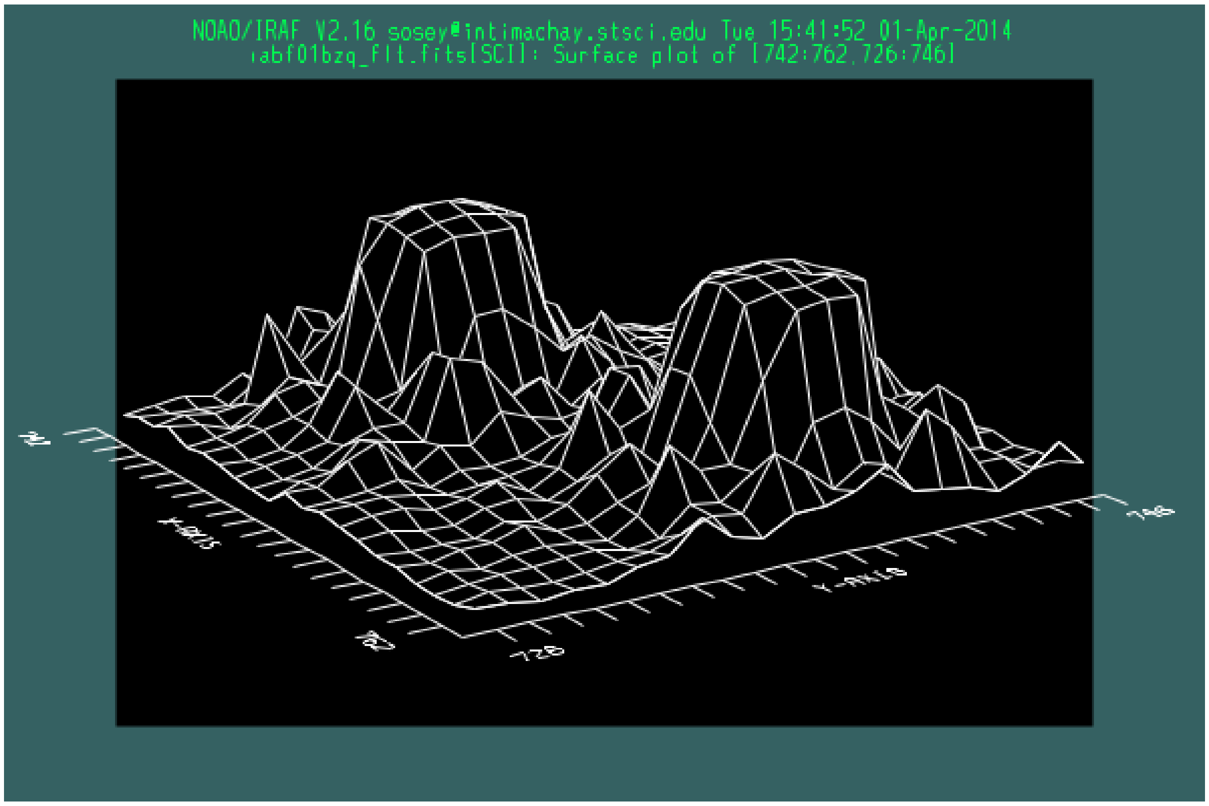


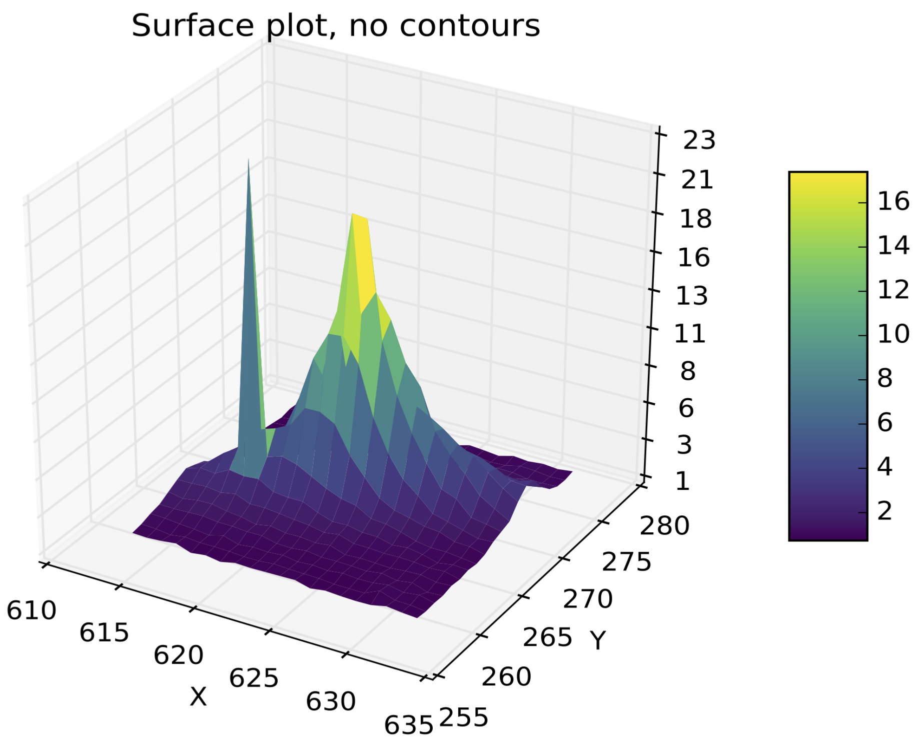
imexam prints the fit information to the screen

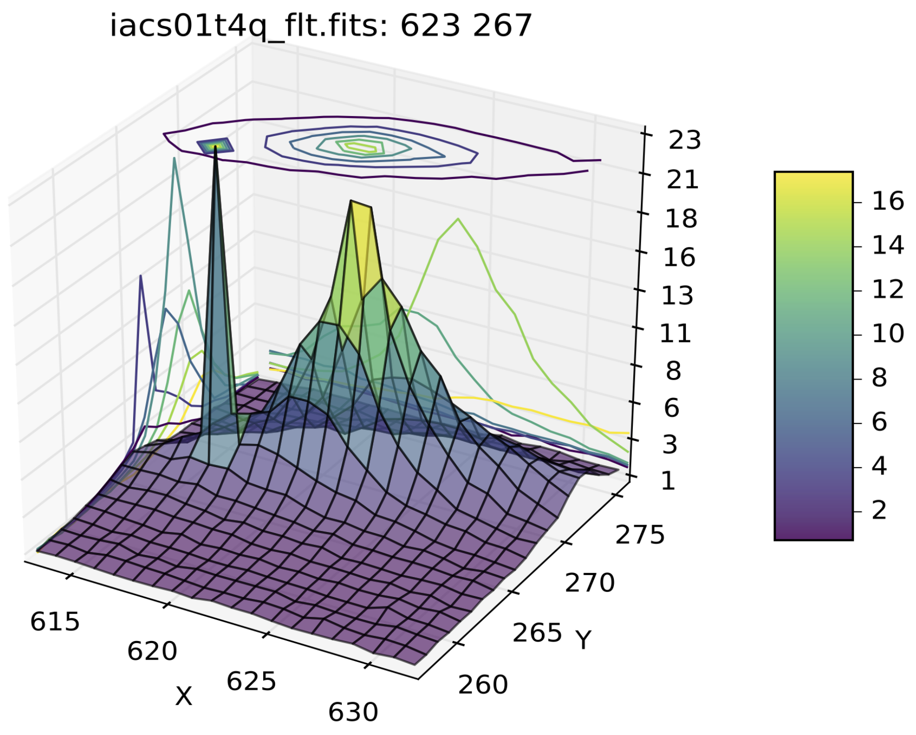
```
xc=585.660034       yc=697.499370
(585,697) mean=585.900, fwhm=5.653
```

## 11.8 Surface plots

The default viewing angle for this package was set to that the axis are easiest to read, the user may choose a different azimuthal value as well. The most fancy imexam surface plot is displayed, the user can alter it with the available options.


NOAO/IRAF V2.16 sosey@intimachay.stsci.edu Tue 15:41:52 01-Apr-2014
iabf01bzq_flt.fits[SCI]: Surface plot of [742:762,726:746]
X-AXIS
Y-AXIS
742
762
726
746


Surface plot, no contours
X
Y
610
615
620
625
630
635
255
260
265
270
275
280
1
3
6
8
11
13
16
18
21
23
2
4
6
8
10
12
14
16


iacs01t4q_flt.fits: 623 267
615
620
625
630
X
260
265
270
275
Y
1
3
6
8
11
13
16
18
21
23
2
4
6
8
10
12
14
16

# Part VI

# Reporting Issues

If you have found a bug in `imexam` please report it by creating a new issue on the `imexam` GitHub issue tracker.

Please include an example that demonstrates the issue sufficiently so that the developers can reproduce and fix the problem. You may also be asked to provide information about your operating system and a full Python stack trace. The developers will walk you through obtaining a stack trace if it is necessary.

`imexam` uses a package of utilities called astropy-helpers during building and installation. If you have any build or installation issue mentioning the `astropy_helpers` or `ah_bootstrap` modules please send a report to the astropy-helpers issue tracker. If you are unsure, then it's fine to report to the main `imexam` issue tracker.

# Part VII

# Contributing

Like the Astropy project, `imexam` is made both by and for its users. We accept contributions at all levels, spanning the gamut from fixing a typo in the documentation to developing a major new feature. We welcome contributors who will abide by the Python Software Foundation Code of Conduct.

`imexam` follows the same workflow and coding guidelines as Astropy. The following pages will help you get started with contributing fixes, code, or documentation (no git or GitHub experience necessary):

- How to make a code contribution
- Coding Guidelines
- Try the development version
- Developer Documentation

For the complete list of contributors please see the imexam contributors page on Github.

# Part VIII

# Reference API

CHAPTER 12

# imexam.connect Module

This is the main controlling class, it allows the user to connect to the viewer and the imexamine classes

## 12.1 Classes

| | |
|---|---|
| `Connect`([target, path, viewer, wait_time, ...]) | Connect to a display device to look at and examine images. |

### 12.1.1 Connect

**class** `imexam.connect.`**`Connect`**(*target=None*, *path=None*, *viewer='ds9'*, *wait_time=10*, *quit_window=True*, *port=None*)

Bases: `object`

Connect to a display device to look at and examine images.

The control features below are a basic set that should be available in all display tools.

The class for the display tool should override them and add it's own extra features.

**Parameters**

**target: string, optional**

the viewer target name or id (default is to start a new instance of a DS9 window)

**path** : string, optional

absolute path to the viewers executable

**viewer: string, optional**

The name of the image viewer you want to use, DS9 is the default

**wait_time: int, optional**

The time to wait for a connection to be eastablished before quitting

### Attributes

| | |
|---|---|
| **window: a pointer to an object** | controls the viewers functions |
| **imexam: a pointer to an object** | controls the imexamine functions and options |

Initialize the imexam control object.

### Methods Summary

| | |
|---|---|
| aimexam([get_name]) | Show the current parameters for the 'a' key. |
| alignwcs(**kwargs) | Align frames with wcs. |
| blink(**kwargs) | Blink windows if available. |
| cimexam([get_name]) | Show the current parameters for the 'c' key. |
| clear_contour() | Clear contours on window if available. |
| close() | Close the window and end connection. |
| cmap(**kwargs) | Set the color map table to something else. |
| colorbar(**kwargs) | Turn the colorbar on the screen on and off. |
| contour(**kwargs) | Show contours on the window. |
| contour_load(*args) | Load contours from a file. |
| crosshair(**kwargs) | Control the current position of the crosshair in the current frame. |
| cursor(**kwargs) | Move the cursor in the current frame to the specified image pixel. |
| disp_header(**kwargs) | Display the header of the current image to a window. |
| eimexam([get_name]) | Show the current parameters for the 'e' key, returns dict. |
| embed(**kwargs) | |
| frame(*args, **kwargs) | Move to a different frame. |
| get_data() | Return a numpy array of the data in the current window. |
| get_data_filename() | Return the filename for the data in the current window. |
| get_frame_info() | Return explicit information about the data displayed. |
| get_header(**kwargs) | Return the current fits header as a string. |
| get_image() | Return the full image object, not just the numpy array. |
| get_viewer_info() | Return a dictionary with information about all frames with data. |
| gimexam([get_name]) | Show the current parameters for curve of growth plots, returns dict. |
| grab() | Display a snapshop of the current image in the browser window. |
| grid(*args, **kwargs) | Convenience method to turn the grid on and off. |
| hideme() | Lower the precedence of the display window. |
| himexam([get_name]) | Show the current parameters for 'h' key, returns dict. |
| imexam() | Run imexamine loop with user interaction. |
| jimexam([get_name]) | Show the current parameters for 1D fit line plots, returns dict. |
| kimexam([get_name]) | Show the current parameters for 1D fit column plots, returns dict. |
| limexam([get_name]) | Show the current parameters for line plots, returns dict. |
| load_fits(*args, **kwargs) | Convenience function to load fits image to current frame. |
| load_mef_as_cube(*args, **kwargs) | Load a Mult-Extension-Fits image one frame as a cube. |

Continued on next page

Table 12.2 – continued from previous page

| | |
|---|---|
| load_mef_as_multi(*args, **kwargs) | Load a Mult-Extension-Fits image into multiple frames. |
| load_region(*args, **kwargs) | Load regions from a file which uses standard formatting. |
| load_rgb(*args, **kwargs) | Load three images into a frame, each one for a different color. |
| make_region(*args, **kwargs) | Make an input reg file with [x,y,comment] to a standard reg file. |
| mark_region_from_array(*args, **kwargs) | Mark regions on the viewer with a list of tuples as input. |
| match(**kwargs) | Match all other frames to the current frame. |
| mimexam([get_name]) | Show the current parameters for statistical regions, returns dict. |
| nancolor(**kwargs) | Set the not-a-number (NaN) color. |
| panto_image(*args, **kwargs) | Convenience function to change to x,y images coordinates. |
| panto_wcs(*args, **kwargs) | Pan to wcs coordinates in image. |
| plotname([filename]) | Change or show the default save plotname for imexamine. |
| readcursor() | Return the image coordinate postion and key pressed. |
| reopen() | Reopen a display window closed by the user but not exited. |
| rimexam([get_name]) | Show the current parameters for curve of growth plots, returns dict. |
| rotate(*args, **kwargs) | Rotate the current frame (in degrees). |
| save_header(*args, **kwargs) | Save the header of the current image to a file. |
| save_regions(*args, **kwargs) | Save the regions on the current window to a file. |
| save_rgb(*args, **kwargs) | Save an rgb image frame that is displayed as an MEF fits file. |
| scale(*args, **kwargs) | Scale the image on display. |
| set_plot_pars([key, item, value]) | Set the chosen plot parameter with the provided value. |
| set_region(*args, **kwargs) | Display a region using the specifications in region_string. |
| setlog([filename, on, level]) | Turn on and off logging to a logfile or the screen. |
| show_window_commands() | Print the available commands for the selected display window. |
| showme() | Raise the precedence of the display window. |
| showpix(*args, **kwargs) | Display the pixel value table, close window when done. |
| snapsave(*args, **kwargs) | Create a snap shot of the current window. |
| timexam([get_name]) | Show current parameters for image cutouts,returns dict. |
| unlearn() | Unlearn all the imexam parameters and reset to default. |
| valid_data_in_viewer() | Return True if a valid file or array is loaded. |
| view(*args, **kwargs) | Display numpy or nddata image array. |
| wimexam([get_name]) | Show the current parameters for surface plots, returns dict. |
| zoom(*args, **kwargs) | Zoom to parameter which can be any recognized string. |
| zoomtofit() | Zoom the image to fit the display. |

### Methods Documentation

**aimexam**(*get_name=False*)

Show the current parameters for the 'a' key.

Either returns the name of the function associated with the keyname Or it returns the dictionary of plotting parameters for that key

**alignwcs**(***kwargs*)
Align frames with wcs.

**blink**(***kwargs*)
Blink windows if available.

**cimexam**(*get_name=False*)
Show the current parameters for the 'c' key.

Either returns the name of the function associated with the keyname Or it returns the dictionary of plotting parameters for that key

**clear_contour**()
Clear contours on window if available.

**close**()
Close the window and end connection.

**cmap**(***kwargs*)
Set the color map table to something else.

Should verify with a defined list of options

**colorbar**(***kwargs*)
Turn the colorbar on the screen on and off.

**contour**(***kwargs*)
Show contours on the window.

**contour_load**(**args*)
Load contours from a file.

**crosshair**(***kwargs*)
Control the current position of the crosshair in the current frame.

crosshair mode is turned on by default

**cursor**(***kwargs*)
Move the cursor in the current frame to the specified image pixel.

it will also move selected regions

**disp_header**(***kwargs*)
Display the header of the current image to a window.

**eimexam**(*get_name=False*)
Show the current parameters for the 'e' key, returns dict.

**embed**(***kwargs*)

**frame**(**args*, ***kwargs*)
Move to a different frame.

**get_data**()
Return a numpy array of the data in the current window.

**get_data_filename**()
Return the filename for the data in the current window.

**get_frame_info**()
Return explicit information about the data displayed.

**get_header**(***kwargs*)
Return the current fits header as a string.

None is returned if there's a problem

**get_image**()
Return the full image object, not just the numpy array.

**get_viewer_info**()
Return a dictionary with information about all frames with data.

**gimexam**(*get_name=False*)
Show the current parameters for curve of growth plots, returns dict.

Either returns the name of the function associated with the keyname Or it returns the dictionary of plotting parameters for that key

**grab**()
Display a snapshop of the current image in the browser window.

**grid**(**args*, ***kwargs*)
Convenience method to turn the grid on and off.

grid can be flushed with many more options

**hideme**()
Lower the precedence of the display window.

**himexam**(*get_name=False*)
Show the current parameters for 'h' key, returns dict.

Either returns the name of the function associated with the keyname Or it returns the dictionary of plotting parameters for that key

**imexam**()
Run imexamine loop with user interaction.

At a minimum it requires a copy of the data array

**jimexam**(*get_name=False*)
Show the current parameters for 1D fit line plots, returns dict.

Either returns the name of the function associated with the keyname Or it returns the dictionary of plotting parameters for that key

**kimexam**(*get_name=False*)
Show the current parameters for 1D fit column plots, returns dict.

Either returns the name of the function associated with the keyname Or it returns the dictionary of plotting parameters for that key

**limexam**(*get_name=False*)
Show the current parameters for line plots, returns dict.

Either returns the name of the function associated with the keyname Or it returns the dictionary of plotting parameters for that key

**load_fits**(**args*, ***kwargs*)
Convenience function to load fits image to current frame.

**load_mef_as_cube**(**args*, ***kwargs*)
Load a Mult-Extension-Fits image one frame as a cube.

**load_mef_as_multi**(**args*, ***kwargs*)
Load a Mult-Extension-Fits image into multiple frames.

**load_region**(**args*, ***kwargs*)
Load regions from a file which uses standard formatting.

**load_rgb**(**args*, ***kwargs*)
Load three images into a frame, each one for a different color.

**make_region**(**args*, ***kwargs*)
Make an input reg file with [x,y,comment] to a standard reg file.

the input file should contains lines with x,y,comment

**mark_region_from_array**(**args*, ***kwargs*)
Mark regions on the viewer with a list of tuples as input.

**match**(***kwargs*)
Match all other frames to the current frame.

**mimexam**(*get_name=False*)
Show the current parameters for statistical regions, returns dict.

Either returns the name of the function associated with the keyname Or it returns the dictionary of plotting parameters for that key

**nancolor**(***kwargs*)
Set the not-a-number (NaN) color.

**panto_image**(**args*, ***kwargs*)
Convenience function to change to x,y images coordinates.

using ra,dec, x, y in image coord

**panto_wcs**(**args*, ***kwargs*)
Pan to wcs coordinates in image.

**plotname**(*filename=None*)
Change or show the default save plotname for imexamine.

**readcursor**()
Return the image coordinate postion and key pressed.

in the form of x,y,str with array offset

**reopen**()
Reopen a display window closed by the user but not exited.

**rimexam**(*get_name=False*)
Show the current parameters for curve of growth plots, returns dict.

Either returns the name of the function associated with the keyname Or it returns the dictionary of plotting parameters for that key

**rotate**(**args*, ***kwargs*)
Rotate the current frame (in degrees).

**save_header**(**args*, ***kwargs*)
Save the header of the current image to a file.

**save_regions**(**args*, ***kwargs*)
Save the regions on the current window to a file.

**save_rgb**(**args*, ***kwargs*)
Save an rgb image frame that is displayed as an MEF fits file.

`scale`(**args*, ***kwargs*)
Scale the image on display.

The default zscale is the most widely used option

`set_plot_pars`(*key=None*, *item=None*, *value=None*)
Set the chosen plot parameter with the provided value.

**Parameters**

**key: String**

The value of the option key, should be a single letter or number

**item: string**

The value of the parameter in the dictionary

**value: float, string, int, bool**

What the parameters value should be set to

## Examples

set_plot_par('j','func','MexicanHat1D')

where j is the key value during imexam func is the parameter you want to edit MexicanHat1D is the name of the astropy function to use

`set_region`(**args*, ***kwargs*)
Display a region using the specifications in region_string.

`setlog`(*filename=None*, *on=True*, *level=20*)
Turn on and off logging to a logfile or the screen.

**Parameters**

**filename: str, optional**

Name of the output file to record log information

**on: bool, optional**

True by default, turn the logging on or off

**level: logging class, optional**

set the level for logging messages, turn off screen messages by setting to logging.CRITICAL

`show_window_commands`()
Print the available commands for the selected display window.

`showme`()
Raise the precedence of the display window.

`showpix`(**args*, ***kwargs*)
Display the pixel value table, close window when done.

`snapsave`(**args*, ***kwargs*)
Create a snap shot of the current window.

save in the specified format. If no format is specified the filename extension is used

**timexam**(*get_name=False*)
Show current parameters for image cutouts,returns dict.

Either returns the name of the function associated with the keyname Or it returns the dictionary of plotting parameters for that key

**unlearn**()
Unlearn all the imexam parameters and reset to default.

**valid_data_in_viewer**()
Return True if a valid file or array is loaded.

**view**(**args*, ***kwargs*)
Display numpy or nddata image array.

If an astropy NDData object is passed without a reference to the data one will be added. I haven't tested this yet for multiarray data

**wimexam**(*get_name=False*)
Show the current parameters for surface plots, returns dict.

Either returns the name of the function associated with the keyname Or it returns the dictionary of plotting parameters for that key

**zoom**(**args*, ***kwargs*)
Zoom to parameter which can be any recognized string.

**zoomtofit**()
Zoom the image to fit the display.

# CHAPTER 13

# imexam.imexamine Module

Licensed under a 3-clause BSD style license - see LICENSE.rst.

This class implements IRAF/imexamine type capabilities for providing powerful diagnostic quick-look tools.

However, the power of this python tool is that it is essentially a library of plotting and analysis routines which can be directed towards any viewer. It can also be used without connecting to any viewer since the calls take only data,x,y information. This means that given a data array and a list of x,y positions you can creates plots without havin to interact with the viewers.

Users can also register a custom function with the class and have it available for use in either case.

The plots which are made are fully customizable

## 13.1 Classes

| | |
|---|---|
| `Imexamine`() | The imexamine class controls plotting and analysis functions. |

### 13.1.1 Imexamine

**class** `imexam.imexamine.`**`Imexamine`**

Bases: `object`

The imexamine class controls plotting and analysis functions.

do imexamine like routines on the current frame.

read the returned cursor key value to decide what to do

region_size is the default radius or side of the square for stat info

### Methods Summary

| | |
|---|---|
| aper_phot(x, y[, data]) | Perform aperture photometry. |
| close() | For use with the Imexamine object standalone. |
| column_fit(x, y[, data, form, genplot, fig]) | Compute the 1d fit to the column of data. |
| contour(x, y[, data, fig]) | plot contours in a region around the specified location. |
| curve_of_growth(x, y[, data, genplot, fig]) | Display a curve of growth plot. |
| cutout(x, y[, data, size, fig]) | Make a fits cutout around the pointer location without wcs. |
| do_option(x, y, key) | Run the imexam option. |
| gauss_center(x, y[, data, delta]) | Return the 2d gaussian fit center of the data. |
| get_options() | Return the imexam options as a key list. |
| get_plot_name() | return the default plot name. |
| histogram(x, y[, data, genplot, fig]) | Calulate a histogram of the data values. |
| line_fit(x, y[, data, form, genplot, fig]) | compute the 1D fit to the line of data using the specified form. |
| new_plot_window(x, y[, data]) | make the next plot in a new plot window. |
| option_descrip(key[, field]) | Return the looked up dictionary of options. |
| plot_column(x, y[, data, fig]) | column plot of data at point y. |
| plot_line(x, y[, data, fig]) | line plot of data at point x. |
| print_options() | Print the imexam options to screen. |
| radial_profile(x, y[, data, form, genplot, fig]) | Display the radial profile plot (intensity vs radius) for the object. |
| register(user_funcs) | register a new imexamine function made by the user as an option. |
| report_stat(x, y[, data]) | report the statisic of values in a box with side region_size. |
| reset_defpars() | set all pars to their defaults. |
| save([filename, fig]) | Save to file the figure that's currently displayed. |
| save_figure(x, y[, data, fig]) | Save to file the figure that's currently displayed. |
| set_aper_phot_pars([user_dict]) | the user may supply a dictionary of par settings. |
| set_colplot_pars() | set parameters for column plots. |
| set_column_fit_pars() | set parameters for 1D line fit plots. |
| set_contour_pars() | set parameters for contour plots. |
| set_curve_pars() | set parameters for curve of growth plots. |
| set_cutout_pars() | set parameters for cutout images. |
| set_data([data, dtype]) | initialize the data that imexamine uses. |
| set_histogram_pars() | set parameters for histogram plots. |
| set_line_fit_pars() | set parameters for 1D line fit plots. |
| set_lineplot_pars() | set parameters for line plots. |
| set_option_funcs() | Define the dictionary which maps imexam keys to their functions. |
| set_plot_name([filename]) | set the default plot name for the "s" key. |
| set_radial_pars() | set parameters for radial profile plots. |
| set_surface_pars() | set parameters for surface plots. |
| setlog([filename, on, level]) | Turn on and off logging to a logfile or the screen. |
| show_fit_models() | Print the available astropy models for plot fits. |
| show_xy_coords(x, y[, data]) | print the x,y,value to the screen. |
| showplt() | Show the plot. |
| surface(x, y[, data, fig]) | plot a surface around the specified location. |
| unlearn_all() | reset the default parameters for all functions. |

### Methods Documentation

`aper_phot`(*x*, *y*, *data=None*)

Perform aperture photometry.

uses photutils functions, photutils must be available

**Parameters**

**x: int**

The x location of the object

**y: int**

The y location of the object

**data: numpy array**

The data array to work on

`close`()

For use with the Imexamine object standalone.

`column_fit`(*x*, *y*, *data=None*, *form=None*, *genplot=True*, *fig=None*)

Compute the 1d fit to the column of data.

**Parameters**

**x: int**

The x location of the object

**y: int**

The y location of the object

**data: numpy array**

The data array to work on

**form: string**

This is the functional form specified in the column fit parameters

**genplot: int**

produce the plot or return the fit model

**fig: figure name for redirect**

Used for interaction with the ginga GUI

#### Notes

delta is the range of data values to use around the x,y location

The background is currently ignored

if centering is True, then the center is fit with a 2d gaussian, but this is currently not done for Polynomial1D

`contour`(*x*, *y*, *data=None*, *fig=None*)

plot contours in a region around the specified location.

**Parameters**

**x: int**

The x location of the object

**y: int**

The y location of the object

**data: numpy array**

The data array to work on

**fig: figure for redirect**

Used for interaction with the ginga GUI

`curve_of_growth`(*x*, *y*, *data=None*, *genplot=True*, *fig=None*)
Display a curve of growth plot.

**Parameters**

**x: int**

The x location of the object

**y: int**

The y location of the object

**data: numpy array**

The data array to work on

**fig: figure name for redirect**

Used for interaction with the ginga GUI

## Notes

the object photometry is taken from photutils

`cutout`(*x*, *y*, *data=None*, *size=None*, *fig=None*)
Make a fits cutout around the pointer location without wcs.

**Parameters**

**x: int**

The x location of the object

**y: int**

The y location of the object

**data: numpy array**

The data array to work on

**size: int**

The radius of the cutout region

**fig: figure for redirect**

Used for interaction with the ginga GUI

`do_option`(*x*, *y*, *key*)
Run the imexam option.

**Parameters**

**x: int**

The x location of the cursor or data point

**y: int**

The y location of the cursor or data point

**key: string**

The key which was pressed

**gauss_center**(*x*, *y*, *data=None*, *delta=10*)
Return the 2d gaussian fit center of the data.

**Parameters**
**x: int**

The x location of the object

**y: int**

The y location of the object

**data: numpy array**

The data array to work on

**delta: int**

The range of data values (bounding box) to use around the x,y location for calculating the center

**get_options**()
Return the imexam options as a key list.

**get_plot_name**()
return the default plot name.

**histogram**(*x*, *y*, *data=None*, *genplot=True*, *fig=None*)
Calulate a histogram of the data values.

**Parameters**
**x: int, required**

The x location of the object

**y: int, required**

The y location of the object

**data: numpy array, optional**

The data array to work on

**genplot: boolean, optional**

If false, returns the hist, bin_edges tuple

**fig: figure name for redirect**

Used for interaction with the ginga GUI

## Notes

This functional originally used the pylab histogram routine for plotting. In order to accomodate returning just the histogram data, this was changed to the numpy histogram, with a subsequent plot if genplot is True.

Does not yet support numpy v1.11 strings for bin estimation.

**line_fit**(*x*, *y*, *data=None*, *form=None*, *genplot=True*, *fig=None*)
compute the 1D fit to the line of data using the specified form.

**Parameters**

**x: int**

The x location of the object

**y: int**

The y location of the object

**data: numpy array**

The data array to work on

**form: string**

This is the functional form specified in the line fit parameters Currently Gaussian1D, Moffat1D, MexicanHat1D, Polynomial1D

**genplot: bool**

produce the plot or return the fit

**fig: figure for redirect**

Used for interaction with the ginga GUI

### Notes

The background is currently ignored

If centering is True in the parameter set, then the center is fit with a 2d gaussian, not performed for Polynomial1D

**new_plot_window**(*x*, *y*, *data=None*)
make the next plot in a new plot window.

### Notes

x,y, data, are not used here, but the calls are setup to take them for all imexam options. Is there a better way to do the calls in general? Once the new plotting window is open all plots will be directed towards it. The old window cannot be used again.

**option_descrip**(*key*, *field=1*)
Return the looked up dictionary of options.

**Parameters**

**key: string**

The key which was pressed, it relates to the function to call

**field: int**

This tells where in the option dictionary the function name can be found

**plot_column**(*x*, *y*, *data=None*, *fig=None*)
column plot of data at point y.

**Parameters**

**x: int**

The x location of the object

**y: int**

The y location of the object

**data: numpy array**

The data array to work on

**fig: figure name for redirect**

Used for interaction with the ginga GUI

`plot_line`(*x*, *y*, *data=None*, *fig=None*)
line plot of data at point x.

**Parameters**
**x: int**

The x location of the object

**y: int**

The y location of the object

**data: numpy array**

The data array to work on

**fig: figure object for redirect**

Used for interaction with the ginga GUI

`print_options`()
Print the imexam options to screen.

`radial_profile`(*x*, *y*, *data=None*, *form=None*, *genplot=True*, *fig=None*)
Display the radial profile plot (intensity vs radius) for the object.

From the parameters Dictionary: If pixel is True, then every pixel at each radius is plotted. If pixel is False, then the sum of all pixels at each radius is plotted.

Background may be subtracted and centering can be done with a 2D Gaussian fit. These options are read from the plot parameters dict.

**Parameters**
**x: int**

The x location of the object

**y: int**

The y location of the object

**data: numpy array**

The data array to work on

**form: string**

The string name of the form of the fit to use

**genplot: bool**

Generate the plot if True, else return the fit data

`register`(*user_funcs*)
register a new imexamine function made by the user as an option.

**Parameters**

**user_funcs: dict**

Contains a dictionary where each key is the binding for the (function,description) tuple

## Notes

The new binding will be added to the dictionary of imexamine functions as long as the key is unique. The new functions do not have to have default dictionaries associated with them.

`report_stat`(*x*, *y*, *data=None*)

report the statisic of values in a box with side region_size.

The statistic can be any numpy function

**Parameters**

**x: int**

The x location of the object

**y: int**

The y location of the object

**data: numpy array**

The data array to work on

`reset_defpars`()

set all pars to their defaults.

`save`(*filename=None*, *fig=None*)

Save to file the figure that's currently displayed.

this is used for the standalone plotting

**Parameters**

**filename: string**

Name of the file the plot will be saved to. The extension on the filename determines the filetype

**fig: figure name for redirect**

Used for interaction with the ginga GUI

`save_figure`(*x*, *y*, *data=None*, *fig=None*)

Save to file the figure that's currently displayed.

this is used for the imexam loop, because there is a standard api for the loop

**Parameters**

**x: int**

The x location of the object

**y: int**

The y location of the object

**data: numpy array**

The data array to work on

**fig: figure for redirect**

Used for interaction with the ginga GUI

**set_aper_phot_pars**(*user_dict=None*)
the user may supply a dictionary of par settings.

**set_colplot_pars**()
set parameters for column plots.

**set_column_fit_pars**()
set parameters for 1D line fit plots.

**set_contour_pars**()
set parameters for contour plots.

**set_curve_pars**()
set parameters for curve of growth plots.

**set_cutout_pars**()
set parameters for cutout images.

**set_data**(*data=array([], dtype=float64)*)
initialize the data that imexamine uses.

**set_histogram_pars**()
set parameters for histogram plots.

**set_line_fit_pars**()
set parameters for 1D line fit plots.

**set_lineplot_pars**()
set parameters for line plots.

**set_option_funcs**()
Define the dictionary which maps imexam keys to their functions.

### Notes

The user can modify this dictionary to add or change options, the first item in the tuple is the associated function the second item in the tuple is the description of what the function does when that key is pressed

**set_plot_name**(*filename=None*)
set the default plot name for the "s" key.

**Parameters**
**filename: string**

The name which is used to save the current plotting window to a file The extension on the name decides which file type is used

**set_radial_pars**()
set parameters for radial profile plots.

**set_surface_pars**()
set parameters for surface plots.

**setlog**(*filename=None*, *on=True*, *level=20*)
Turn on and off logging to a logfile or the screen.

**Parameters**
**filename: str, optional**

Name of the output file to record log information

**on: bool, optional**

True by default, turn the logging on or off

**level: logging class, optional**

set the level for logging messages, turn off screen messages by setting to logging.CRITICAL

`show_fit_models()`

Print the available astropy models for plot fits.

`show_xy_coords`(*x*, *y*, *data=None*)

print the x,y,value to the screen.

**Parameters**

**x: int**

The x location of the object

**y: int**

The y location of the object

**data: numpy array**

The data array to work on

`showplt()`

Show the plot.

`surface`(*x*, *y*, *data=None*, *fig=None*)

plot a surface around the specified location.

**Parameters**

**x: int**

The x location of the object

**y: int**

The y location of the object

**data: numpy array**

The data array to work on

**fig: figure for redirect**

Used for interaction with the ginga GUI

`unlearn_all()`

reset the default parameters for all functions.

# CHAPTER 14

# imexam.ds9_viewer Module

Licensed under a 3-clause BSD style license - see LICENSE.rst

This class supports communication with DS9 through the XPA

Some code in this class was adapted from pysao, which can be found at https://github.com/leejjoon/pysao. Specifically this package used the existing Cython implementation to the XPA and extended the calls to the other available XPA executables so that more functionality is added. The API information is available here:

http://hea-www.harvard.edu/RD/xpa/client.html#xpaopen

Using Cython will allow for broader development of the code and produce faster runtimes for large datasets with repeated calls to the display manager.

XPA is licensed under MIT, help can be found here: http://hea-www.cfa.harvard.edu/saord/xpa/help.html

The current XPA can be downloaded from here: http://hea-www.harvard.edu/saord/xpa/

## 14.1 Classes

| | |
|---|---|
| `ds9`([target, path, wait_time, quit_ds9_on_del]) | Control all interactions between the user and the DS9 window. |

### 14.1.1 ds9

**class** `imexam.ds9_viewer.`**`ds9`**(*target=''*, *path=''*, *wait_time=5*, *quit_ds9_on_del=True*)

Bases: `object`

Control all interactions between the user and the DS9 window.

The ds9() contructor takes a ds9 target as its main argument. If none is given, then a new window and process will be started.

DS9's xpa access points are documented in the reference manual, but the can also be returned to the user with a call to xpaset.

http://hea-www.harvard.edu/saord/ds9/ref/xpa.html

**Parameters**

**target: string, optional**

the ds9 target name or id. If None or empty string, a new ds9 instance is created.

**path** : string, optional

path of the ds9. Used only if a new ds9 is requested.

**wait_time** : float, optional

waiting time before error is raised

**quit_ds9_on_del** : boolean, optional

If True, try to quit ds9 when this instance is deleted.

## Attributes

| | |
|---|---|
| **wait_time: float** | The waiting time before error is raised |
| **path: string** | The path to the DS9 executable |
| **_xpa_name: string** | The value in XPA_METHOD, the name of the DS9 window |
| **_quit_ds9_on_del: boolean** | Determine whether to quit ds9 when object goes out of scope. |
| **_ds9_unix_name: string** | The full path filename to the unix socket, only if unix sockets are being used with local |
| **_need_to_purge: boolean** | whenever there are unix socket directories which need to be purged when the object goes out of scope |
| **_tmpd_name: string** | The full path name to the unix socket file on the local system |
| **_filename: string** | The name of the image that's currently loaded into DS9 |
| **_ext: int** | Extension of the current image that is loaded. If one extension of an MEF file is loaded this will be 1 regardless of the extension that was specified (because DS9 and the XPA now see it as a single image and header) |
| **_extname: string** | If available, the EXTNAME of the MEF extension that is loaded, taken from the current data header |
| **_extver: int** | If available, the EXTVER of the MEF extension that is loaded, taken from the current data header |
| **_ds9_process: pointer** | Points to the ds9 process id on the system, returned by Popen, whenever this module starts DS9 |
| **_mef_file: boolean** | The file is a multi-extension fits file |
| **_iscube: bookean** | The file is a multiextension fits file, and one of the extensions contains at least 1 additional extension (3D or more) |
| **_numaxis: int** | number of image planes, this is NAXIS |
| **_naxis: tuple** | specific image plane displayed, defaulted to 1 image plane, most relevant to cube fits files |

starter.

### Notes

I think this is a quirk in the XPA communication. The xpans, and XPA prefer to have all connections be of the same type. DS9 defaults to creating an INET connection. In some cases, if no IP address can be found for the computer, the startup can hang. In these cases, a local connection is preferred, which uses a unix filename for the socket.

The problem arises that if the user already has DS9 windows running, that were started by default, the nameserver is only listening for the default socket type (inet) and not local. There are also cases where the machine running this code does not have xpa installed, so there is no xpans (nameserver) to run and keep track of the open connections. In that case, the user needs to provide this init with the name of the socket in their window (in XPA_METHOD) in order to create the connection.

### Methods Summary

| | |
|---|---|
| `alignwcs`([on]) | align wcs. |
| `blink`([blink, interval]) | Blink frames. |
| `clear_contour`() | clear contours from the screen. |
| `close`() | close the window and end connection. |
| `cmap`([color, load, invert, save, filename]) | Set the color map table, using a defined list of options. |
| `colorbar`([on]) | turn the colorbar on the bottom of the window on and off. |
| `contour`([on, construct]) | show contours on the window. |
| `contour_load`(filename) | load a contour file into the window. |
| `crosshair`([x, y, coordsys, skyframe, ...]) | Control the position of the crosshair in the current frame. |
| `cursor`([x, y]) | move the cursor in the current frame to the specified image pixel. |
| `disp_header`() | Display the header of the current image to a DS9 window. |
| `embed`() | Embed the viewer in a notebook. |
| `frame`([n]) | convenience function to change or report frames. |
| `get`(param) | XPA get method to ds9 instance which returns received string. |
| `get_data`() | return a numpy array of the data displayed in the current frame. |
| `get_filename`() | return the filename currently on display. |
| `get_frame_info`() | return more explicit information about the data displayed. |
| `get_header`([fitsobj]) | Return the current fits header. |
| `get_image`() | return the full image object instead of just the data array. |
| `get_slice_info`() | return the slice tuple that is currently displayed. |
| `get_viewer_info`() | Return a dictionary of information. |
| `grab`() | Make a copy of the image view. |
| `grid`([on, param]) | convenience to turn the grid on and off. |
| `hideme`() | lower the ds9 window. |
| `iscube`() | return whether a cube image is displayed in the current frame. |
| `load_fits`([fname, extver, mecube]) | convenience function to load fits image to current frame. |
| `load_mef_as_cube`([filename]) | Load a Mult-Extension-Fits image into one frame as an image cube. |

Continued on next page

Table 14.2 – continued from previous page

| | |
|---|---|
| `load_mef_as_multi`([filename]) | Load a Mult-Extension-Fits image into multiple frames. |
| `load_region`(filename) | Load regions from a file which uses ds9 standard formatting. |
| `load_rgb`(red, green, blue[, scale, lockwcs]) | load 3 images into an RGBimage frame. |
| `make_region`(infile[, labels, header, ...]) | make an input reg file with [x,y,comment] to a DS9 reg file. |
| `mark_region_from_array`(input_points[, ...]) | mark ds9 regions regions given an input list of tuples. |
| `match`([coordsys, frame, crop, fslice, ...]) | match all other frames to the current frame. |
| `nancolor`([color]) | set the not-a-number color, default is red. |
| `panto_image`(x, y) | convenience function to change to x,y physical image coordinates. |
| `panto_wcs`(x, y[, system]) | pan to wcs location coordinates in image. |
| `readcursor`() | Returns the image coordinate postion and key pressed. |
| `reopen`() | Reopen a closed window. |
| `rotate`([value, to]) | rotate the current frame (in degrees). |
| `run_inet_ds9`() | start a new ds9 window using an inet socket connection. |
| `save_regions`([filename]) | save the regions in the current window to a DS9 style regions file. |
| `save_rgb`([filename]) | save an rgbimage frame as an MEF fits file. |
| `scale`([scale]) | The default zscale is the most widely used option. |
| `set`(param[, buf]) | XPA set method to ds9 instance. |
| `set_iraf_display`() | Set the environemnt variable IMTDEV to the current display. |
| `set_region`([region_string]) | display a region using the specifications in region_string. |
| `show_xpa_commands`() | Print the available XPA commands. |
| `showme`() | raise the ds9 window. |
| `showpix`([close]) | display the pixel value table, close window when done. |
| `snapsave`([filename, format, resolution]) | Create a snap shot of the current window, save in specified format. |
| `valid_data_in_viewer`() | return bool if valid file or array is loaded into the viewer. |
| `view`(img) | Display numpy image array to current frame. |
| `zoom`([par]) | Zoom using the specified command. |
| `zoomtofit`() | Zoom to fit the image to the viewer. |

## Methods Documentation

**alignwcs**(*on=True*)

align wcs.

**Parameters**

**on: bool**

Align the images using the WCS in their headers

**blink**(*blink=True*, *interval=None*)

Blink frames.

**Parameters**

**blink: bool, optional**

Set to True to start blinking the frames which are open

**interval: int**

Set interval equal to the length of pause for blinking

### Notes

**blink_syntax=**
Syntax: blink [true|false] [interval <value>]

`clear_contour()`
clear contours from the screen.

`close()`
close the window and end connection.

`cmap`(*color=None*, *load=None*, *invert=False*, *save=False*, *filename='colormap.ds9'*)
Set the color map table, using a defined list of options.

**Parameters**
**color: string**

color must be set to one of the available DS9 color map names

**load: string, optional**

set to the filename which is a valid colormap lookup table valid contrast values are from 0 to 10, and valid bias values are from 0 to 1

**invert: bool, optional**

invert the colormap

**save: bool, optional**

save the current colormap as a file

**filename: string, optional**

the name of the file to save the colormap to

`colorbar`(*on=True*)
turn the colorbar on the bottom of the window on and off.

**Parameters**
**on: bool**

Set to True to turn on the colorbar, False to turn it off

`contour`(*on=True*, *construct=True*)
show contours on the window.

**Parameters**
**on: bool**

Set to true to turn on contours

**construct: bool, optional**

Will open the contour dialog box which has more options

`contour_load`(*filename*)
load a contour file into the window.

**Parameters**
**filename: string**

The name of the file with the contour level defined

`crosshair`(*x=None*, *y=None*, *coordsys='physical'*, *skyframe='wcs'*, *skyformat='fk5'*, *match=False*, *lock=False*)

Control the position of the crosshair in the current frame.

crosshair mode is turned on automatically

**Parameters**

**x: string or int**

The value of x is converted to a string for the call to XPA, use a value here appropriate for the skyformat you choose

**y: string or int**

The value of y is converted to a string for the call to XPA, use a value here appropriate for the skyformat you choose

**coordsys: string, optional**

The coordinate system your x and y are defined in

**skyframe: string, optional**

If skyframe has "wcs" in it then skyformat is also sent to the XPA

**skyformat: string, optional**

Used with skyframe, specifies the format of the coordinate which were given in x and y

**match: bool, optional**

If set to True, then the wcs is matched for the frames

**lock: bool, optional**

If set to True, then the frame is locked in wcs

`cursor`(*x=None*, *y=None*)

move the cursor in the current frame to the specified image pixel.

selected regions will also be moved

**Parameters**

**x: int**

pixel location x coordinate to move to

**y: int**

pixel location y coordinate to move to

**x and y are converted to strings for the call**

`disp_header`()

Display the header of the current image to a DS9 window.

`embed`()

Embed the viewer in a notebook.

`frame`(*n=None*)

convenience function to change or report frames.

**Parameters**

**n: int, string, optional**

The frame number to open or change to. If the number specified doesn't exist, a new frame will be opened If nothing is specified, then the current frame number will be returned. The value of n is converted to a string before passing to the XPA

### Examples

frame(1) sets the current frame to 1 frame("last") set the current frame to the last frame frame() returns the number of the current frame frame("new") opens a new frame frame(3) opens frame 3 if it doesn't exist already, otherwise goes to frame 3

`get`(*param*)
XPA get method to ds9 instance which returns received string.

**Parameters**
**param** : parameter string (eg. "fits" "regions")

### Notes

This function is linked with the Cython implementation get(param)

`get_data`()
return a numpy array of the data displayed in the current frame.

### Notes

This is the data array that the imexam() function from connect() uses for analysis

astropy.io.fits stores data in row-major format. So a 4d image would be [NAXIS4, NAXIS3, NAXIS2, NAXIS1] just the one image is retured in the case of multidimensional data, not the cube

`get_filename`()
return the filename currently on display.

This function will check if there is already a filename saved. It's possible that the user can connect to a ds9 window with no file loaded and then ask for the data file name after loading one through the ds9 menu options. This will poll the private filename and then try and set one if it's empty.

`get_frame_info`()
return more explicit information about the data displayed.

`get_header`(*fitsobj=False*)
Return the current fits header.

The return value is the string or None if there's a problem If fits is True then a fits header object is returned instead

`get_image`()
return the full image object instead of just the data array.

`get_slice_info`()
return the slice tuple that is currently displayed.

`get_viewer_info`()
Return a dictionary of information.

The dictionary contains information about all frames which are loaded with data

### Notes

Consider adding a loop to verify that all the frames still exist and the user has not deleted any through the gui.

`grab()`
Make a copy of the image view.

`grid(on=True, param=False)`
convenience to turn the grid on and off.

grid can be flushed with many more options

**Parameters**
**on: bool, optional**

Will turn the grid overlay on in the current frame

**param: bool, optional**

Will open the parameter dialog in DS9

`hideme()`
lower the ds9 window.

`iscube()`
return whether a cube image is displayed in the current frame.

`load_fits(fname=None, extver=None, mecube=False)`
convenience function to load fits image to current frame.

**Parameters**
**fname: string, optional**

The name of the file to be loaded. You can specify the full extension in the name, such as filename_flt.fits or filename_flt.fits[1]

**extver: int, optional**

The extension to load (EXTVER in the header)

**mecube: bool, optional**

If mecube is True, load the fits file as a cube into ds9

### Notes

To tell ds9 to open a file whose name or path includes spaces, surround the path with "{...}", e.g.

% xpaset -p ds9 file "{foo bar/my image.fits}"

This is assuming that the image loads into the current or next new frame, watch the internal file and ext values because the user can switch frames through DS9 app itself

XPA needs to have the absolute path to the filename so that if the DS9 window was started in another directory it can still find the file to load.

`load_mef_as_cube(filename=None)`
Load a Mult-Extension-Fits image into one frame as an image cube.

`load_mef_as_multi(filename=None)`
Load a Mult-Extension-Fits image into multiple frames.

`load_region`(*filename*)
Load regions from a file which uses ds9 standard formatting.

**Parameters**
**filename: string**

The file containing the DS9 style region description

`load_rgb`(*red*, *green*, *blue*, *scale=False*, *lockwcs=False*)
load 3 images into an RGBimage frame.

**Parameters**
**red: string**

The name of the fits file loaded into the red channel

**green: string**

The name of the fits file loaded into the green channel

**blue: string**

The name of the fits file loaded into the blue channel

**scale: bool**

If True, then each image will be scaled with zscale() after loading

**lockwcs: bool**

If True, then the image positions will be locked to each other using the WCS information in their headers

`make_region`(*infile*, *labels=False*, *header=0*, *textoff=10*, *size=5*)
make an input reg file with [x,y,comment] to a DS9 reg file.

the input file should contain lines specifying x,y,comment

**Parameters**
**infile: str**

input filename

**labels: bool**

add labels to the regions

**header: int**

number of header lines in text file to skip

**textoff: int**

offset in pixels for labels

**size: int**

size of the region type

### Notes

only circular regions are supported currently

`mark_region_from_array`(*input_points*, *ptype='image'*, *textoff=10*, *size=4*)
mark ds9 regions regions given an input list of tuples.

a convienence function, you can also use set_region

**Parameters**

**input_points: iterator, a tuple, or list of tuples**

or a string: (x,y,comment),

**ptype: string**

the reference system for the point locations, image|physical|fk5

**size: int**

the size of the region marker

**textoff: string**

the offset for the comment text, if comment is empty it will not show

## Notes

only circular regions are supported currently

**match**(*coordsys='wcs'*, *frame=True*, *crop=False*, *fslice=False*, *scale=False*, *bin=False*, *colorbar=False*, *smooth=False*, *crosshair=False*)

match all other frames to the current frame.

**Parameters**

**coordsys: string, optional**

The coordinate system to use

**frame: bool, optional**

Match all other frames to the current frame, using the set coordsys

**crop: bool, optional**

Set the current image display area, using the set coordsys

**fslice: bool, optional**

Match current slice in all frames

**scale: bool, optional**

Match to the current scale for all frames

**bin: bool, optional**

Match to the current binning for all frames

**colorbar: bool, optional**

Match to the current colorbar for all frames

**smooth: bool, optional**

Match to the current smoothing for all frames

**crosshair: bool, optional**

Match the crosshair in all frames, using the current coordsys

### Notes

You can only choose one of the options at a time, and the logic will select the first True option so set frame=False and something else in addition to your choice if you don't want the default option.

`nancolor`(*color='red'*)
set the not-a-number color, default is red.

**Parameters**
**color: string**

The color to use for NAN pixels

`panto_image`(*x*, *y*)
convenience function to change to x,y physical image coordinates.

**Parameters**
**x: float**

X location in physical coords to pan to

**y: float**

Y location in physical coords to pan to

`panto_wcs`(*x*, *y*, *system='fk5'*)
pan to wcs location coordinates in image.

**Parameters**
**x: string**

The x location to move to, specified using the given system

**y: string**

The y location to move to

**system: string**

The reference system that x and y were specified in, they should be understood by DS9

`readcursor`()
Returns the image coordinate postion and key pressed.

### Notes

XPA returns strings of the form: u a 257.5 239

`reopen`()
Reopen a closed window.

`rotate`(*value=None*, *to=False*)
rotate the current frame (in degrees).

the current rotation is printed with no params

**Parameters**
**value: float [degrees]**

Rotate the current frame {value} degrees If value is 0, then the current rotation is printed

**to: bool**

Rotate the current frame to the specified value

**run_inet_ds9**()
start a new ds9 window using an inet socket connection.

## Notes

It is given a unique title so it can be identified later.

**save_regions**(*filename=None*)
save the regions in the current window to a DS9 style regions file.

**Parameters**
**filename: string**

The nameof th file to which the regions displayed in the current window are saved. If no filename is provided then it will try and save the regions to the name of the file in the current display with _regions.txt appended

If a file of that name already exists on disk it will no attempt to overwrite it

**save_rgb**(*filename=None*)
save an rgbimage frame as an MEF fits file.

**Parameters**
**filename: string**

The name of the output fits image

**scale**(*scale='zscale'*)
The default zscale is the most widely used option.

**Parameters**
**scale: string**

The scale for ds9 to use, these are set strings of [linear|log|pow|sqrt|squared|asinh|sinh|histequ]

## Notes

The xpa doesn't return an error if you set an unknown scale, it just doesn't do anything, this is true for all the xpa calls

**set**(*param*, *buf=None*)
XPA set method to ds9 instance.

## Notes

This function is linked with the Cython implementation

set(param, buf=None) param : parameter string (eg. "fits" "regions") buf : aux data string (sometime string needed to be ended with CR)

**set_iraf_display**()
Set the environemnt variable IMTDEV to the current display.

the socket address of the current imexam.ds9 instance is used Notes —– For example, your pyraf commands will use this ds9 for display.

TODO: Not sure this is still needed. Stop using IRAF.

**set_region**(*region_string=''*)
display a region using the specifications in region_string.

**Parameters**
**region_string: string**

Should take the form of a region string that DS9 is expecting

## Examples

set_region("physical ruler 200 300 200 400") set_region("line 0 400 3 400 #color=red")

**show_xpa_commands**()
Print the available XPA commands.

**showme**()
raise the ds9 window.

**showpix**(*close=False*)
display the pixel value table, close window when done.

**Parameters**
**close: bool, optional**

If set to True, then the pixel table dialog window is closed

**snapsave**(*filename=None*, *format=None*, *resolution=100*)
Create a snap shot of the current window, save in specified format.

This function bypasses the XPA calling routines to avoid a bug with the X11/XPA interface. Instead is uses the os import function which takes a snapshot of the specified x11 window.

**Parameters**
**filename: str, optional**

filename of output image, the extension in the filename can also be used to specify the format. If no filename is specified, then the filename will be constructed from the name of the image displayed image with _snap.png appended.

**format: str, optional**

available formats are fits, eps, gif, tiff, jpeg, png If no format is specified the filename extension is used

**resolution: int, optional**

1 to 100, for jpeg images

**valid_data_in_viewer**()
return bool if valid file or array is loaded into the viewer.

**view**(*img*)
Display numpy image array to current frame.

**Parameters**
**img: numpy array**

The array containing data, it will be forced to numpy.array()

**zoom**(*par='to fit'*)
Zoom using the specified command.

**Parameters**

**par: string**

- **it can be a number (ranging 0 to 8 effectively), and successive**
  calls continue zooming in the same direction
- it can be two numbers '4 2', which specify zoom on different axis
- if can be to a specific value 'to 8' or 'to fit'
- it can be 'open' to open the dialog box
- **it can be 'close' to close the dialog box (only valid if the box**
  is already open)

### Examples

zoom('0.1')

**`zoomtofit`**()

Zoom to fit the image to the viewer.

CHAPTER 15

# imexam.ginga_viewer Module

Licensed under a 3-clause BSD style license - see LICENSE.rst

This class supports communication with a Ginga-based viewer. For default key and mouse shortcuts in a Ginga window, see: https://ginga.readthedocs.org/en/latest/quickref.html

## 15.1 Classes

| | |
|---|---|
| `ginga`([exam, close_on_del, logger, port, ...]) | A ginga-based viewer that displays to an HTML5 widget in a browser. |
| `ginga_general`([exam, close_on_del, logger, port]) | A base class which controls all interactions between the user and the ginga widget. |

### 15.1.1 ginga

**class** `imexam.ginga_viewer.`**`ginga`**(*exam=None*, *close_on_del=True*, *logger=None*, *port=None*, *host='localhost'*, *use_opencv=False*)

Bases: `imexam.ginga_viewer.ginga_general`

A ginga-based viewer that displays to an HTML5 widget in a browser.

This is compatible with the Jupyter notebook and can be run from a server.

This kind of viewer has slower performance than if we choose some widget back ends, but the advantage is that it works so long as the user has a working browser.

All the rendering is done on the server side and the browser only acts as a display front end. Using this you could create an analysis type environment on a server and view it via a browser or from a Jupyter notebook.

#### Methods Summary

| | |
|---|---|
| `close()` | Close the viewing window. |
| `reopen()` | Reopen the viewer window if the user closes it accidentally. |

### Methods Documentation

**`close()`**
Close the viewing window.

**`reopen()`**
Reopen the viewer window if the user closes it accidentally.

## 15.1.2 ginga_general

**class** `imexam.ginga_viewer.`**`ginga_general`**(*exam=None*, *close_on_del=True*, *logger=None*, *port=None*)
Bases: `object`

A base class which controls all interactions between the user and the ginga widget.

The ginga contructor creates a new window using the ginga backend.

**Parameters**
**close_on_del** : boolean, optional

If True, try to close the window when this instance is deleted.

### Attributes

| | |
|---|---|
| **view: Ginga view object** | The object instantiated from a Ginga view class |
| **exam: imexamine object** | |

initialize a general ginga viewer object.

**Parameters**
**exam: imexam object**

This is the imexamine object which contains the examination functions

**close_on_del: bool**

If True, the window connection shuts down when the object is deleted

**logger: logger object**

Ginga viewers all need a logger, if none is provided it will create one

**port: int**

This is used as the communication port for the HTML5 viewer. The user can choose to have multiple windows open at the same time as long as they have different port designations. If no port is specified, this class will choose an open port.

### Methods Summary

| | |
|---|---|
| `blink()` | Blink multiple frames. |

Continued on next page

Table 15.3 – continued from previous page

| | |
|---|---|
| close() | Close the window. |
| cmap([color, load, invert, save, filename]) | Set the color map table to something else, using a defined list of options. |
| contour_load() | Load a file with contour information. |
| crosshair(**kwargs) | Control the current position of the crosshair in the frame. |
| cursor(**kwargs) | Move the cursor in the current frame to the specified image pixel. |
| disp_header() | Display the fits header for the current data. |
| embed([width, height]) | Embed the current window into the notebook. |
| frame() | Convenience function to report frames. |
| get_data() | Return a numpy array of the data displayed in the current frame |
| get_filename() | Return the filename currently associated with the data |
| get_frame_info() | Return more explicit information about the data in current frame. |
| get_header() | Return current fits header as string, None if there's a problem. |
| get_image() | Return the AstroImage instance for the data in the viewer |
| get_slice_info() | Return the slice tuple that is currently displayed. |
| get_viewer_info() | Return a dictionary of information about all frames with data |
| grab() | |
| grid(*args, **kwargs) | Turn the grid display on and off. |
| hideme() | Lower the display window in prededence. |
| iscube() | Return whether a cube image is displayed in the current frame. |
| load_fits([fname, extver]) | Load fits image to current frame. |
| load_mef_as_cube(*args, **kwargs) | Load a Mult-Extension-Fits image one frame as a cube. |
| load_mef_as_multi(*args, **kwargs) | Load a Mult-Extension-Fits image into multiple frames. |
| load_region(*args, **kwargs) | Load regions from a file which uses standard formatting. |
| load_rgb(*args, **kwargs) | Load three images into a frame, each one for a different color. |
| make_region(*args, **kwargs) | make an input reg file with [x,y,comment] to a standard reg file. |
| mark_region_from_array(*args, **kwargs) | Mark regions on the viewer with a list of tuples as input. |
| match(**kwargs) | Match all other frames to the current frame. |
| nancolor(**kwargs) | Set the not-a-number (Nan) color. |
| panto_image(x, y) | Change to x,y physical image coordinates. |
| panto_wcs(x, y[, system]) | Pan to wcs location coordinates in image |
| readcursor() | Returns image coordinate postion and key pressed. |
| rotate([value]) | Rotate the current frame (in degrees). |
| save_header(*args, **kwargs) | Save the header of the current image to a file. |
| save_regions(*args, **kwargs) | Save the displayed regions on the current window to a file. |
| save_rgb(*args, **kwargs) | Save an rgb image frame that is displayed as an MEF fits file. |
| scale([scale]) | Scale the image intensity, zscale is used as the default. |

Continued on next page

Table 15.3 – continued from previous page

| | |
|---|---|
| `set_region`(*args, **kwargs) | Display a region using the specifications in region_string. |
| `show_window_commands`() | Print the available commands for the selected display. |
| `showme`() | Raise the precendence of the display window. |
| `showpix`(*args, **kwargs) | Display the pixel value table, closing the window when done. |
| `snapsave`() | Save a frame display as a PNG file. |
| `start_event_loop`() | |
| `transform`([flipx, flipy, swapxy]) | Transform the frame. |
| `valid_data_in_viewer`() | Return bool if a valid file or array is loaded into the viewer |
| `view`(img) | Display numpy image array in current frame |
| `zoom`(zoomlevel) | Zoom the image using the specified zoomlevel. |
| `zoomtofit`() | Zoom the image to fit the display. |

## Methods Documentation

**blink**()
Blink multiple frames.

**close**()
Close the window.

**cmap**(*color=None*, *load=None*, *invert=False*, *save=False*, *filename='colormap.ds9'*)
Set the color map table to something else, using a defined list of options.

**Parameters**

**color: string**

color must be set to one of the available color map names

**load: string, optional**

set to the filename which is a valid colormap lookup table valid contrast values are from 0 to 10, and valid bias values are from 0 to 1

**invert: bool, optional**

invert the colormap

**save: bool, optional**

save the current colormap as a file

**filename: string, optional**

the name of the file to save the colormap to

**contour_load**()
Load a file with contour information.

**crosshair**(***kwargs*)
Control the current position of the crosshair in the frame.

crosshair mode is turned on.

**cursor**(***kwargs*)
Move the cursor in the current frame to the specified image pixel.

it will also move selected regions

**disp_header**()
Display the fits header for the current data.

**embed**(*width=600*, *height=650*)
Embed the current window into the notebook.

**frame**()
Convenience function to report frames.

currently only 1 frame is supported per calling object in HTML5 display

**get_data**()
Return a numpy array of the data displayed in the current frame

### Notes

This is the data array that the imexam() function from connect() uses for analysis

astropy.io.fits stores data in row-major format. So a 4d image would be [NAXIS4, NAXIS3, NAXIS2, NAXIS1] just the one image is retured in the case of multidimensional data, not the cube

**get_filename**()
Return the filename currently associated with the data

**get_frame_info**()
Return more explicit information about the data in current frame.

**get_header**()
Return current fits header as string, None if there's a problem.

**get_image**()
Return the AstroImage instance for the data in the viewer

**get_slice_info**()
Return the slice tuple that is currently displayed.

**get_viewer_info**()
Return a dictionary of information about all frames with data

**grab**()

**grid**(**args*, ***kwargs*)
Turn the grid display on and off.

grid can be flushed with many more options

**hideme**()
Lower the display window in prededence.

**iscube**()
Return whether a cube image is displayed in the current frame.

**load_fits**(*fname=''*, *extver=None*)
Load fits image to current frame.

**Parameters**
**fname: string, optional**

The name of the file to be loaded. You can specify the full extension in the name, such as filename_flt.fits[sci,1] or filename_flt.fits[1]

**extver: int, optional**

The extension to load (EXTVER in the header)

### Notes

Extname isn't used here, ginga wants the absolute extension number, not the version number associated with a name

**load_mef_as_cube**(**args*, ***kwargs*)
Load a Mult-Extension-Fits image one frame as a cube.

**load_mef_as_multi**(**args*, ***kwargs*)
Load a Mult-Extension-Fits image into multiple frames.

**load_region**(**args*, ***kwargs*)
Load regions from a file which uses standard formatting.

**load_rgb**(**args*, ***kwargs*)
Load three images into a frame, each one for a different color.

**make_region**(**args*, ***kwargs*)
make an input reg file with [x,y,comment] to a standard reg file.

the input file should contains lines with x,y,comment

**mark_region_from_array**(**args*, ***kwargs*)
Mark regions on the viewer with a list of tuples as input.

**match**(***kwargs*)
Match all other frames to the current frame.

**nancolor**(***kwargs*)
Set the not-a-number (Nan) color.

**panto_image**(*x*, *y*)
Change to x,y physical image coordinates.

**Parameters**

**x: float**

X location in physical coords to pan to

**y: float**

Y location in physical coords to pan to

**panto_wcs**(*x*, *y*, *system='fk5'*)
Pan to wcs location coordinates in image

**Parameters**

**x: string**

The x location to move to, specified using the given system

**y: string**

The y location to move to

**system: string**

The reference system that x and y were specified in, they should be understood by DS9

**readcursor**()
Returns image coordinate postion and key pressed.

**rotate**(*value=None*)
Rotate the current frame (in degrees).

the current rotation is printed with no params

**Parameters**
**value: float [degrees]**

Rotate the current frame {value} degrees If value is None, then the current rotation is printed

**save_header**(**args*, ***kwargs*)
Save the header of the current image to a file.

**save_regions**(**args*, ***kwargs*)
Save the displayed regions on the current window to a file.

**save_rgb**(**args*, ***kwargs*)
Save an rgb image frame that is displayed as an MEF fits file.

**scale**(*scale='zscale'*)
Scale the image intensity, zscale is used as the default.

**Parameters**
**scale: string**

The scale for ds9 to use, these are set strings of [linear|log|pow|sqrt|squared|asinh|sinh|histequ]

**set_region**(**args*, ***kwargs*)
Display a region using the specifications in region_string.

**show_window_commands**()
Print the available commands for the selected display.

**showme**()
Raise the precendence of the display window.

**showpix**(**args*, ***kwargs*)
Display the pixel value table, closing the window when done.

**snapsave**()
Save a frame display as a PNG file.

**Parameters**
**filename: string**

The name of the output PNG image

**start_event_loop**()

**transform**(*flipx=None*, *flipy=None*, *swapxy=None*)
Transform the frame.

**Parameters**
**flipx: boolean**

if True flip the X axis, if False don't, if None leave current

**flipy: boolean**

if True flip the Y axis, if False don't, if None leave current

**swapxy: boolean**

if True swap the X and Y axes, if False don't, if None leave current

`valid_data_in_viewer()`
Return bool if a valid file or array is loaded into the viewer

`view`(*img*)
Display numpy image array in current frame

**Parameters**
**img: numpy array**

The array containing data, it will be forced to numpy.array()

### Examples

view(np.random.rand(100,100))

`zoom`(*zoomlevel*)
Zoom the image using the specified zoomlevel.

**Parameters**
**zoomlevel: integer**

### Examples

zoom(6) zoom(-3)

`zoomtofit()`
Zoom the image to fit the display.

# Python Module Index

# Index