stginga: Ginga Plugins for Data Analysis and Quality Assurance of HST and JWST Science Data

Pey Lian Lim,[1] and Eric Jeschke[2]

[1]*STScI, Baltimore, MD, USA;* `lim@stsci.edu`

[2]*NAOJ, Hilo, HI, USA*

Abstract.

`stginga`[1] is an image visualization package to assist in data analysis and quality assurance of science data from Hubble Space Telescope (HST) and James Webb Space Telescope (JWST). It is based on the Ginga[2] toolkit for building scientific viewers. In this article, we will describe the main plugins developed with `stginga`. We also discuss the basic outline of writing a Ginga plugin, with pointers to documentation and examples.

1. Introduction

Ginga (Jeschke et al. 2015) is a Python package that implements a toolkit for building scientific viewers. It provides a *reference viewer*, which features a plugin architecture in which nearly every graphical feature of the program is implemented by a Python plugin. By implementing some new plugins for the HST and JWST data analysis and quality assurance tasks, and combining these with a curated selection of the distributed "stock" plugins, we were able to fairly quickly develop a tool for use in the HST and JWST community.

The reference viewer separates image data into virtual holding pens called *channels*, named and organized by the user. Plugins are categorized as *global* or *local*. A global plugin applies to all images across all channels: only one instance can be opened in the whole Ginga session, whereas a local plugin is associated with the channel it is started from: one instance can be opened per channel and different instances can be configured separately in the same Ginga session.

2. BackgroundSub

BackgroundSub (see Figure 1) is used to calculate and subtract background value. User draws a shape (e.g., annulus) to define the region from which background is calculated. As user modifies the region or changes the parameters in the "Attributes" box, background value would be recalculated accordingly. Optionally, if a data quality (DQ)

[1]`https://github.com/spacetelescope/stginga` (STScI)

[2]`https://github.com/ejeschke/ginga` (NAOJ)

extension is available, pixels marked as "not good" also can be excluded from calculations. Subtraction parameters can be saved to a JSON file, which then can be reloaded.

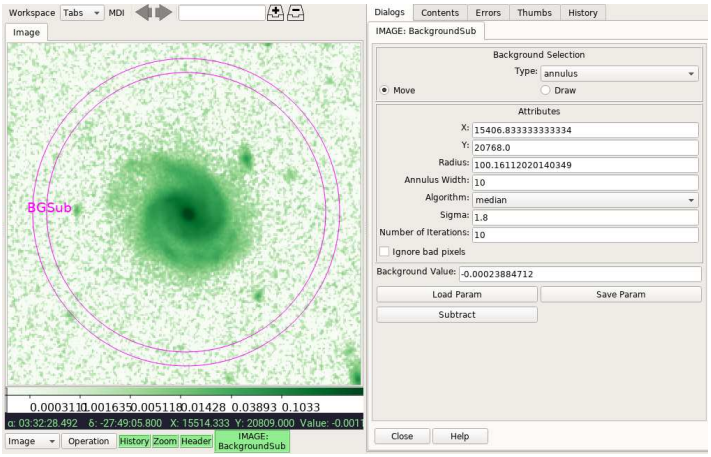

Figure 1. BackgroundSub plugin for background subtraction.

Then, the calculated background can be subtracted off the displayed image in Ginga. However, the subtracted image only exists in an in-memory cache in Ginga; if the cache fills up Ginga will eject the image if it is not being viewed. To save the subtracted image out to a different file, use the *SaveImage* plugin in Ginga. As of this writing, *BackgroundSub* only handles constant background, therefore unsuitable for when background has a gradient or a pattern.

3. BadPixCorr

BadPixCorr[3] is a plugin for performing interactive bad pixel correction on an image.[4] Currently, it only handles fixing a single bad pixel or bad pixels within a circular region. The bad pixel(s) can be filled either by a user-defined constant, a constant calculated from an annulus (not unlike *BackgroundSub*), or Scipy `griddata` interpolation using the annulus. If DQ extension is present, the corresponding DQ flags will also be set to the given new flag value (default is 0 for "good").

4. DQInspect

DQInspect (see Figure 2) is used to visualize the associated DQ array stored as an HDU within an image. It shows the different DQ flags (top table) that went into a selected pixel (marked by a red "x") and also the overall mask of the selected DQ flag(s) (blue/covered pixels; bottom table). For overall mask, when multiple flags are selected, each flag is assigned a different mask color at a reduced opacity for each. User has the option to customize flag definitions for different instruments.

---

[3]Figure not shown here but available in the corresponding poster.

[4]See *BackgroundSub* for comments on JSON support and in-memory cache handling of corrected image.

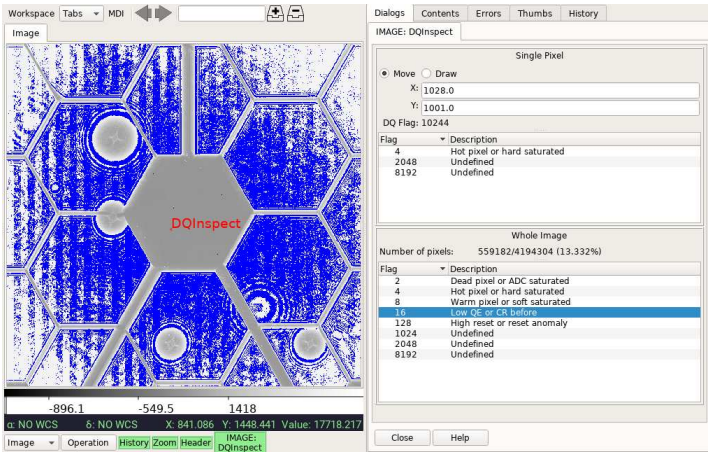

Figure 2. DQInspect plugin for data quality inspection.

5. SNRCalc

SNRCalc[3] is used to calculate Signal-to-Noise Ratio (SNR) and Surface Background Ratio (SBR) on an image. Given the selected science (S) and background (B) regions, SBR is defined by Ball Aerospace (Acton 2015) as the median of S divided by the standard deviation of B. If the image has an accompanying error (E) extension, SNR can also be calculated by dividing S by E over the same region and then computing its minimum, maximum, and mean.

While SNR is more popular, SBR is useful for an image without existing or reliable error values. User may define a minimum limit for SBR check, so that the GUI can provide a quick visual indication on whether the selected region achieves the desired SBR or not. As part of the statistics, mean background value is also provided albeit not used in SBR nor SNR calculations. Optionally, if DQ extension is available, pixels marked as "not good" can be excluded from calculations as well. Calculated values can be saved in the image header using the "Update HDR" button.[4]

6. Writing a Ginga plugin

Instructions for writing a plugin is available at `https://bit.ly/writeplugins`. Existing plugins in Ginga and `stginga` code repositories can be used as examples. It is recommended that you play with the existing ones and choose one that is the closest to your desired functionality as a starting point.

6.1. Local plugins

A local plugin at it's simplest is just a Python class defined in a file. The class should inherit from *ginga.GingaPlugin.LocalPlugin* and provide *__init__()*, *build_gui()*, *start()*, and *stop()* methods. These methods are used to initialize the plugin, build the user interface, and to do any necessary tasks at the start and stop of the plugin, respectively. Typically you would also want to implement the *redo()* method, which is called when there is new data loaded into the viewer to which the running plugin should respond.

Inside the file, any modules that are available in the user's Python environment may be imported and used, allowing huge flexibility in the kinds of things a plugin can do. It can open files, connect to sockets or other communication frameworks, or call a myriad of astronomical Python packages. It also has a reference to the viewer with which it is associated so it can access the viewer data (as a Numpy array) and can manipulate canvas overlays with graphics on the viewer (as shown in the sections above) or manipulate the viewer settings (e.g., panning, scale, color map).

6.2. Global plugins

Writing a global plugin is similiar to the process for writing a local one. The difference is that the plugin ostensibly must be able to update it's state when the user switches channels, since there only one instance of the plugin is allowed to be open; There are callbacks for which you can register to be alerted of these events. Otherwise, the API is quite similar to that of a local plugin.

6.3. Distributing plugins

When you want to distribute your plugin(s) the best way is to probably use the *ginga-plugin-template*.[5] This template allows one or more plugins to be installed as a separate package, and be discovered by the reference viewer when it starts up. If you want more control over the layout of the viewer and the set of included plugins, you can follow the path blazed by `stginga` and make your own startup script to for the reference viewer with a curated mix of the stock plugins with your own.

7. Conclusion

`stginga` utilizes Ginga plugins to support HST and JWST data analysis, which includes background subtraction, bad pixel correction, DQ flags inspection, and signal-to-noise calculations.

Writing Ginga plugins can be an expedient way to develop graphical data analysis and quality assurance tasks, by leveraging the combination of Python, a lean Ginga plugin API, and the burgeoning number of open-source astronomical Python modules.

Both `stginga` and `ginga` are installable via `pip`. Alternately, if you use `conda`, they are also available on AstroConda,[6] in addition to `ginga` being in `conda-forge` too.

References

Acton, S. 2015, Image Pre-Processor SBR, Private communications

Jeschke, E., Inagaki, T., & Kackley, R. 2015, in Astronomical Data Analysis Software and Systems XXIV, edited by A. R. Taylor, & E. Rosolowsky (ASP). Vol. 495

[5] `https://github.com/ejeschke/ginga-plugin-template` (NAOJ)

[6] `https://astroconda.readthedocs.io` (STScI)