JWST Astronomy Data Analysis Tools Roadmap Documentation

Release 0.1

Henry Ferguson, Perry Greenfield, and Alberto Conti

October 28, 2016

Contents

1 Executive Summary **3**

2 The vision **5**

3 Guiding principles **7**
3.1 Open Source Software . . . 7
3.2 Easy to install . . . 8
3.3 Well documented . . . 8
3.4 Easy to extend . . . 9
3.5 Multiple interfaces . . . 9
3.6 Stable, widely adopted languages . . . 10
3.7 Stable, widely adopted libraries . . . 10
3.8 Leverage existing codes & algorithms . . . 11

4 Why do we need new tools? **13**
4.1 Make science more efficient . . . 13
4.2 Remove scientist dependency on IRAF . . . 13
4.3 Modern programming language . . . 14
4.4 Make better use of community code . . . 14
4.5 Modern algorithms where relevant . . . 15
4.6 Leverage advances in computer hardware . . . 15

5 Science Use Cases **17**
5.1 Faint Galaxies . . . 17
5.2 Infrared Slit Spectroscopy of Galactic Objects . . . 17
5.3 Imaging Young Stellar Objects in the Magellanic Clouds . . . 20
5.4 Need more use cases . . . 20

6 Technologies and Infrastructure **21**
6.1 Data Formats . . . 21
6.2 Data Abstraction . . . 22
6.3 Parameter Handling . . . 23
6.4 Scientific and Numerical Libraries . . . 23
6.5 Physical Units and Constants . . . 24
6.6 Interprocess Communications . . . 24
6.7 Multiprocessing . . . 24
6.8 Special-Purpose Hardware . . . 25
6.9 GUI Frameworks . . . 25
6.10 Software Distribution . . . 25

6.11 Documentation . . . 26
6.12 Testing . . . 26
6.13 Graphics and Image Displays . . . 27

7 Architecture **29**

8 The Computational Toolbox **31**
8.1 General-purpose multi-dimensional Array Analysis tools . . . 32
8.2 Imaging . . . 34
8.3 Spectra and Spectral Extraction . . . 35
8.4 3D Spectroscopy . . . 38
8.5 Source extraction, morphology and photometry . . . 42
8.6 Simulation . . . 43
8.7 Other tools . . . 44

9 Graphics and Visualization **47**
9.1 2D image display . . . 47
9.2 3D image display . . . 47
9.3 Interactive 2D & 3D graphics . . . 48
9.4 Publication-quality graphics . . . 49
9.5 Easy-to-construct widgets . . . 49
9.6 Easy-to-construct web graphics . . . 49

10 Development Timeline **51**

11 Indices and tables **53**

abstract

author Henry Ferguson, Perry Greenfield, and Alberto Conti

date 30 June 2013

The purpose of this document is to provide a roadmap for developing the software tools that astronomers need for going from pipeline-reduced data to scientific publications. The document looks at the process of analyzing data for several different science cases, and looks at existing tools in IRAF to identify the highest priorities for new tool development or for porting of existing algorithms to a modern python environment. For the most part, the numerical computations will be coded in Python and incorporated into astropy. Many of the tools will have broad applicability beyond JWST, so it makes sense for the code to be open source and the development to be a broad community effort. This roadmap, while geared toward JWST, is intended to foster a community dialog for the evolution of these software tools.

Contents:

CHAPTER 1

Executive Summary

This is a brief overview of the document.

CHAPTER 2

The vision

The overall vision of the data analysis tools effort is the following:

- To ensure that astronomers are equipped with software tools to analyze and interpret JWST data efficiently right from the start of the mission.
- To reduce the necessity for astronomers to write data-analysis software.
- To make it easier to do so when necessary.
- To provide a rich set of modular software tools upn which to build
- To make it easy to share & re-use code.
- To improve repeatability of scientific results.

The expected users of these tools are HST and JWST observers, and astronomers working on similar kinds of data from other facilities. While there is some overlap with tools developed for X-ray and radio astronomy, the primary focus is on UV, Optical and IR observations. The main focus is on "post-pipeline" analysis of already-calibrated data, but the essential building blocks of most standard calibration pipelines are included.

CHAPTER 3

Guiding principles

The following are guiding principles for developing this suite of software tools. In this section, we describe both principle, the rationale for it, and the specific choice that we have made (or are considering) to respond to that guiding principle. Briefly, the guiding principles are:

1. Open source software
2. Easy to install
3. Well documented
4. Easy to extend
5. Multiple interfaces
6. Built on stable, widely adopted languages
7. Built on stable, widely adopted code libraries
8. Leverage existing codes and algorithms.

These principles are by and large what distinguish code intended to serve a broad community from code written by most individual astronomers for their own day-to-day research.

We shall consider each principle in turn.

3.1 Open Source Software

To ensure the repeatibility of modern scientific results, it is important that the main computational code (not just the description of the algorithms) be available to researchers. Repeatability should not mean re-investing hundreds of man-years of effort to reproduce a chain of analysis, when the code could be inspected and tested by independent researchers. Sometimes the error in a scientific finding is not the result of a deep error of scientific judgment, but rather a simple typo in computer code. *The guiding principle for the Data Analysis Tools is that all of the source code for the core numerical computations should be open source.* It is less important that all of the code for GUI interfaces, image displays, and interprocess communication be open source. Where practical, however, we will adopt open-source solutions in preference to closed-source solutions.

The current plan is to closely link the development of the tools described here to the open-source project astropy. The concept is that of the tools will become a part of the astropy core distribution, and work seamlessly with code contributed to astropy from the rest of the community.

3.1.1 Licensing

The term *open source* can mean different things to different people. In the context of this roadmap, it means the following:

- Access to the source code;
- The right to make copies and distribute those copies without unreasonable restrictions;
- The right to make changes and distribute those changes without unreasonable restrictions.

The commonly agreed-upon *reasonable restriction* is that the source code and any binary redistribution must retain the copyright notice and the original liability disclaimer.

There are a variety of open-source software licenses, with various pros and cons. The current plan is to adopt the 3-clause BSD-style license used for astropy. This license is compatible with most GPL-style licenses.

3.2 Easy to install

With many astronomers having to do their own system administration, it is important to make the software easy to install on the most commonly used computing platforms. This suggests the following high-level requirements:

1. A personal installation should be possible without system-administrator privileges
2. A multi-user shared installation should be possible with system-administrator privileges
3. The installation package should include all of the major library dependencies
   (a) The installation should not break or overwrite previously-installed versions of these libraries
4. The software should have the ability to check if is up to date

It is worth commenting that ease of installation does not preclude setting up virtual machines, either on the user's own platform or in the cloud. Indeed, if the installation procedure fulfills the goals above, setting up a virtual machine would simply layer some steps on top to provide all of the OS and programming-language infrastructure. While virtual machines are a very interesting way of making computing available to the community, we do not envision that as our primary method for providing access to the software.

3.3 Well documented

Excellent documentation is essential if the code is to fulfill the vision of being easy to use, share, and extend. The following are specific forms of documentation:

1. User guides
2. Cookbooks and tutorials
3. Help command or help buttons
4. Coding reference guide or API documentation
5. Comments in the source code

In general, documentation should be available in the browser either on the web or by pointing the browser to the local documentation repository, and the same documentation should be available in pdf. There have been significant advances in the past 5 years in the ease of generating user and API documentation and presenting it in multiple formats. Our current plan is to use the Sphinx documentation generator to accomplish this.

The use of docstrings in python makes providing help easy, and the same text is easily incorporated into the user guide. This is especially helpful for the Application Programming Interface (API) documentation.

User guides provide a brief summary of each tool, describing the computational algorithm and how to use it. Tutorials and cookbooks provide worked examples, often chaining together different tasks with discussion of the rationale behind each step. Tutorials can be effective in the form of videos or ipython notebooks.

3.4 Easy to extend

A typical workflow for an astronomer often starts with running a sequence of tasks one-by-one on a specific data set, tuning the adjustable parameters of each task to perfect the data processing. Once that is accomplished, the astronomer will often want to chain these tasks together, possibly with some extra code to set some of the parameters or follow some branch based on some property of the specific data. It is extremely important that the tools be developed in a way to facilitate this kind of work flow. This implies the following goals for the data-analysis tools:

1. They should be modular
2. They should provide consistent APIs
3. They should provide easy-to-use libraries

Being modular is somewhat hard to achieve. If the tasks are too primitive, then there are too many steps needed to chain them together. If they are too complex, then one ends up with a massive user guide to describe all of the adjustable parameters or various kinds of I/O. An example of non-modular code (from the perspective of most users) is the popular faint-galaxy photometry package SExtractor, which (for example) requires the user to re-run the source detection every time one changes any input parameter even if the change has nothing to do with source detection step. The user guide is excellent, and there is even an excellent SExtractor for dummies guide. But most astronomers won't get in and modify the code itself because it is too intimidating. The C code itself is reasonably modular, but the individual steps have not been exposed to be tweaked and chained together in different ways. A more modular approach is exemplified by the relatively new scikit-image, which provides a set of lower-level image-segmentation and image-morphology tasks and a way to link then together in a pipeline.

The guiding principle for the JWST data analysis tools is to build the complex tasks in a transparent, well-documented way around the more primitive modules, and make it easy for astronomers to modify a module or chain the modules together in a different way.

Providing consistent APIs means that modules that do similar things should have a similar calling sequence. This is also somewhat hard to achieve. Generally speaking it means adopting some common naming conventions for parameters, and common method naming within classes that have similar purposes.

If the code is modular and has consistent APIs, then providing easy-to-use libraries means packaging the different modules in a sensible way and providing good documentation. This is also somewhat of a challenge to achieve, especially if the libraries incorporate community-generated code.

3.5 Multiple interfaces

In developing the data analysis tools, a guiding principle is that the basic numerical algorithms should be available through multiple interfaces:

1. Via the command line
2. Via a scripting interface

And that the scripting interface should support calling the same routines from:

1. Various GUI interfaces
2. Client-server interfaces

Most astronomers are comfortable working with tasks at the command-line level (even if that command line is within an environment like IRAF or IDL). Most astronomers are also accustomed to chaining tasks together in a scripting language. The software will use python as the scripting language.

As the number of parameters grows or the level of interactivity grows, it is important to have graphical user interfaces (GUIs). These usually interact with other processes via events, and event-handlers which trigger calls to a specific subroutine. A goal for the JWST data analysis tools is to segregate the numerical computations from the GUI so that multiple GUIs can access the same code, and so that the same code that is feeding the GUIs can be used by astronomers in their scripts. (This was not the case, for example, with the Java-based HST Exposure Time Calculators.) The challenge to enabling multiple GUIs is to develop a well-documented GUI abstraction layer.

It is becoming increasingly common for astronomers or astronomical institutions to provide web services that involve some computation. A goal is to make it easy to use the data analysis tools to build such services on the *server side*. This should be straightforward.

It is *not* a goal make the numerical-computation portion of the software compatible with client-side computing in the browser. Most of the computations are too complex for this to be practical, and the numerical libraries to support this are generally lacking. However, the concept of developing GUIs within the browser is very appealing.

3.6 Stable, widely adopted languages

There is a strong tension between the desire for a stable computing platforming and the desire for a platform that incorporates the latest cutting-edge computational developments. This roadmap aims for a happy medium by choosing languages that are widely used in science – not just astronomy – and are also widely used outside of the sciences.

1. C
2. Python
3. Javascript (for browser interfaces but not for heavy numerical computation)

All three languages are in the top 10 of the TIOBE index of programming language popularity and are considered mainstream. All three are very popular outside of science based on metrics such as the number job advertisements or searches for language tutorials, popularity on GITHub and Stack Overflow, or popularity in a variety of other metrics. This is not to say these languages are better or worse than others, simply that they are safe choices and are likely to have a large developer community for at least the next decade. In contrast, the scripting languages used by R, IRAF, IDL, and MATLAB are unique to those environments, making for a smaller communities of software developers, less open-source code, and less community-based online support.

The general strategy will be to code as much as possible in Python, moving to C when it offers significant performances advantages, and using Javascript only for browser interfaces.

3.7 Stable, widely adopted libraries

The numerical and scientific libraries available for Python and C are substantial. For python, the code will leverage the many developer-years invested in the following packages:

1. numpy – The standard python array-manipulation package
2. scipy – Scientific, numerical and statistical libraries
3. matplotlib – 2-D (and some 3D) graphics
4. astropy – Astronomy-oriented tasks
5. ipython and ipython notebook – user interfaces
6. ConfigObj – configuration file handling

For C, we plan to adopt the following libraries:

1. CFITSIO?
2. What else?

There are a variety of other libraries under development, which may or may not be useful for the Astronomy Tools development. These will be discussed under Technologies and Infrastructure.

We expect that many if not most of the Data Analysis tools developed in this roadmap will be part of astropy.

3.8 Leverage existing codes & algorithms

Where practical, we will re-use existing code or algorithms. For IDL and IRAF, this generally means trying to draw from the algorithms rather than the code. For python code, the general strategy will be to try to get it incorporated into astropy and up to the astropy standards for configuration control and documentation.

CHAPTER 4

Why do we need new tools?

There are a variety of reasons why a development effort is needed for data analysis tools for JWST. First and foremost, it is important to reduce the amount of time between receipt of the data and achieving the scientific result. Shortening this cycle is incredibly important because of JWST's limited lifetime.

4.1 Make science more efficient

Most astronomers spend large amounts of time writing computer code (usually scripts) to manipulate, view, and analyze data. A set of modern, well-documented tools can reduce this coding burden and speed up the process of converting data into scientific knowledge. If a piece of code is re-used by more than one astronomer, there is savings of a factor of two in development cost per astronomer-hour, minus the time spent training the second astronomer in how to use the code. Since the funding for astronomers and science-software development come from the same source, this is an instant savings, and basically translates to more science per dollar. The data-analysis tools described here will be useful to hundreds of astronomers, so even counting time spent writing documentation and training people in the use, the investment will be worthwhile – provided that the tools are actually tools that astronomers want to use (hence the importance of community engagement in their development).

There also is a huge advantage to be gained from providing tools that work well with a modern scripting language. This reduces the time time wasted by astronomers when performing repetitive tasks with tools that don't allow easy automation.

There are specific tools that are either non-existent in the community, or nowhere near at the state of maturity needed for JWST – e.g. tools for support of 3D spectroscopy, tools for dealing with JWST error arrays and WCS information, tools for simulating JWST data, and tools for simultaneously using data from HST, JWST, ALMA, and other facilities in a seamless and statistically rigorous fashion.

4.2 Remove scientist dependency on IRAF

It is widely recognized that continuing to develop software based on core IRAF libraries, which are largely written in SPP and CL, two languages that have no community outside of IRAF (and barebones support within the community), would be inefficient and costly in the long run. We need a graceful way to retire CL, SPP, and dependencies on core IRAF. (The extensive set of high-level analysis tools written in SPP, together with the relatively large number of astronomers that still use the CL for IRAF scripting make this a challenge.)

In a sense, the astronomical community is encumbered with a technical debt in IRAF. Forcing the JWST scientific community to rely on that platform for day-to-day their data analysis and visualization coding makes no long-term economic sense. Indeed, IRAF core development has been starved of resources over the past few years, making it even more obsolete relative to other options. Paying down the debt requires investment (in real dollars, or FTE), but will pay off in the long run in greater science efficiency.

STScI developed PyRAF as a step in this evolution. Pyraf brought more powerful scripting, error handling, and array-manipulation capabilities, better interoperability with other languages, and access to a growing body of open-source scientific libraries. Partly as a result of this, a growing fraction of astronomers entering the field are using the python language as a central part of their software toolbox. For that generation, the transition away from IRAF entirely should be relatively painless once some of the core functionality is replaced.

4.3 Modern programming language

Programming languages continue to evolve. Modern languages have evolved to keep up with changes in computer architectures and changes in coding paradigms. Choosing popular languages like C and Python helps ensure that the language will keep up with the changes in computer hardware. For IRAF, the burden of making those underlying changes falls entirely on the astronomical community, for IDL or MATLAB it largely falls on a single corporation with its own strategic goals, creating a significant risk if the corporation changes its focus or fails to survive.

Python in particular offers the following modern features, which make it particularly attractive for writing most of the code:

- It is a high-level, interpreted language
- It has a clean, readable syntax
- It is very portable
- It supports both object-oriented programming, and procedure-oriented programming
- It supports 64-bit architectures and threading multiple CPUs
- It has extensive error handling capabilities
- It is free and open source
- It can be extended with other languages (e.g. C)
- It has an enormous open-source development community supporting the base language and libraries (both general and scientific).

The main practical advantage relative to the IRAF CL is that it is possible to do efficient array arithmetic like in IDL and MATLAB, and there are extensive libraries to support scientific computations. Another significant advantage is Python's error trapping, which makes debugging code much easier than for the CL. The advantages relative to IDL, Matlab, and R include the language syntax, its popularity, and the fact that it is free. Furthermore, those environments are particularly geared toward scientific and statistical analysis and can be cumbersome when dealing with many other common operations that often confront astronomers in their work (e.g. text processing, web development, gui development, etc.). In terms of speed for most numerical computations Python+numpy is comparable to IDL and Matlab – faster for some operations, slower for others.

Python takes about 1/6 the number of lines of code as C, which at least anecdotally translates into a significant decrease in the amount of time it takes an experience programmer to write their program. Since coding is Python is relatively quick, one can develop the functionality in Python first and address any performance bottlenecks on the second iteration.

The downside of adopting a state-of-the-art language, of course, is that it is evolving. There is an ongoing cost for code maintenance to handle compatibility problems as the languages evolve.

4.4 Make better use of community code

The overall cost of data analysis tool development could be reduced if there were more sharing and less duplication of effort. Therefore an important part of building a new analysis-tool infrastructure is to encourage such code shar-

ing. This has become more common and easier now with the rise in popularity of tools like github for open-source development.

The concept in this roadmap is to build most of the tools into **'astropy <http://www.astropy.org>.'_** The Astropy Project is a community effort to develop a single core package for Astronomy in Python and foster interoperability between Python astronomy packages. Astropy includes afffilated packages that are not part of the core source code but can be found from the astropy website. Astropy does not depend on these packages, but the packages may depend on astropy. These must be downloaded and installed separately from astropy itself.

Strategies that can help encourage sharing of community code include providing:

- Rubust, well-documented libraries in upon which to build.
- Easy-to-follow standards for source-code style and documentation.
- Templates for source code.
- Clear instructions on how to package the code so that it can be installed on multiple platforms, along with expert assistance when necessary or feasible.
- An active developers' forum.

4.5 Modern algorithms where relevant

There has been considerable evolution over the past few decades in the techniques and algorithms used to analyze data, not only in astronomy but also in other fields. This includes advances in statistical techniques such as resampling and Monte-Carlo Markov Chains (MCMC), different ways to approach the challenges of model fitting or optimization (e.g. genetic algorithms), and different ways to compress or describe data via basis functions (e.g. wavelets, shapelets, Karhunen-Loeve decompositions, etc.). Implementations of many of these exist already in the python/C ecosystem, so including them in the astronomer's toolbox in many cases may just be an issue of documentation and packaging.

4.6 Leverage advances in computer hardware

Computer hardware has evolved significantly since IRAF was developed. A significant fraction of the IRAF infrastructure, for example, deals with operating-system abstraction (allowing inter-operability with VAX/VMS and Unix) with network abstraction, and with magnetic tape I/O. In the meantime, standard desktop and laptop computer hardware now offers featuers such as full 64-bit addressing, multiple cores, hardware graphics acceleration and computation in graphics processing units (GPUs). A lot of infrastructure for using these hardware advances already exists for Python and C, although some of it is evolving rapidly (e.g. for GPUs).

Section *** discusses this infrastructure in more detail, with an indication of which current libraries and packages are under consideration.

CHAPTER 5

Science Use Cases

Before diving into more detailed concepts for specific tools, it is useful to have some examples of how the tools might be used. Because the tools are meant to be general-purpose and because the science program on JWST diverse, these use cases are meant to be illustrative rather than exhaustive. We focus in particular on JWST, but many of the tasks are similar for other observatories.

To illustrate the use cases, flow digrams are shown for different types of JWST science. These flow diagrams start where the JWST pipeline leaves off. The data have been calibrated, rectified, co-added, and extracted according to the standard pipeline algorithms. We assume that these calibrations are sufficient, or that the astronomer can re-run the pipeline if needed. The steps shown here are much more difficult to support in a general-purpose pipeline, because they often depend on the specific science goals and require inspection and judgement from the investigator at various steps. They may also combine data from other missions, which is beyond the scope of the JWST pipeline.

5.1 Faint Galaxies

A core science area for JWST involves imaging of faint galaxies. Observations will generally be in multiple bands, which may or may not be observed with the same guide stars. The different bands will have different point-spread functions. The investigator will often want to combine the JWST images with other data sets from other observatories. The flow diagram below shows various tasks that an astronomer would be expected to combine the data, perform custom measurements, derive physical quantities for individual galaxies, and construct statistical distributions for comparison to theory.

5.2 Infrared Slit Spectroscopy of Galactic Objects

For long-slit or multi-slit spectroscopy, the JWST pipeline will present the astronomer with extracted one-dimensional spectra that are wavelength and flux calibrated. Other observatories may or may not have an automated pipeline which does that. Even for JWST, the cautious astronomer is going to want to carefully check the results in the first year or two of JWST operations.

Further analysis will involve identifying spectral features (or cross-correlating with models), and determining parameters such as fluxes, equivalent widths, or line profiles. Analysis often includes sophisticated modeling. In some cases, the models are compared to compared to these derived quantities (which must have well-understood uncertainties). In other cases, the models are used to generate simulated spectra (with as high fidelity as possible) and are compared directly to the data.

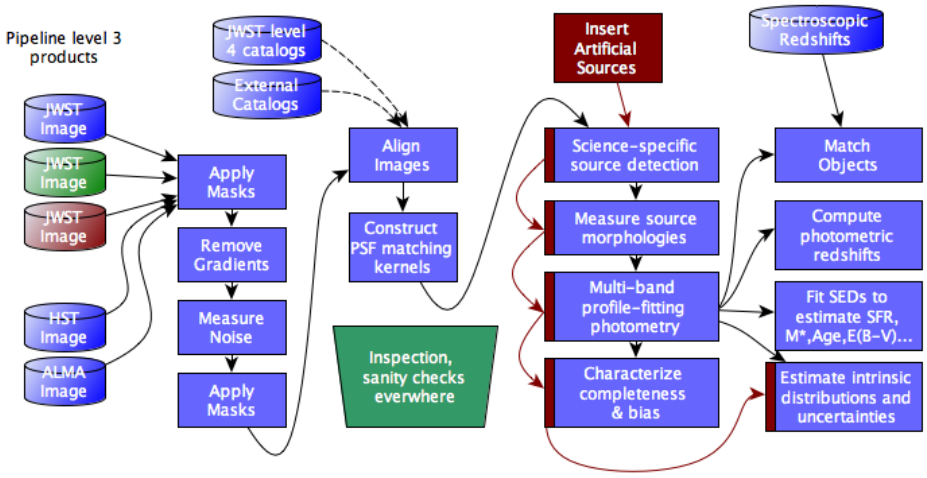
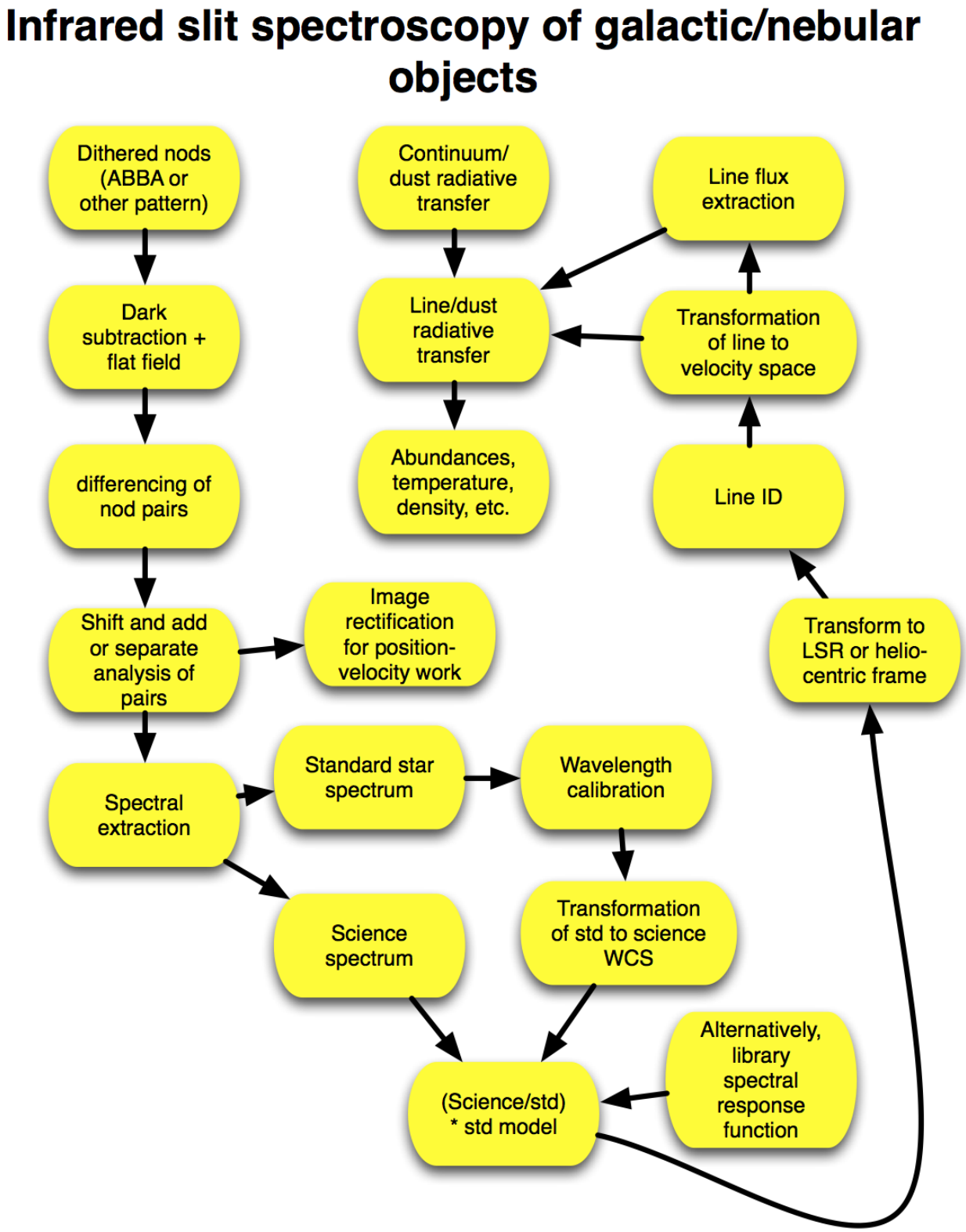
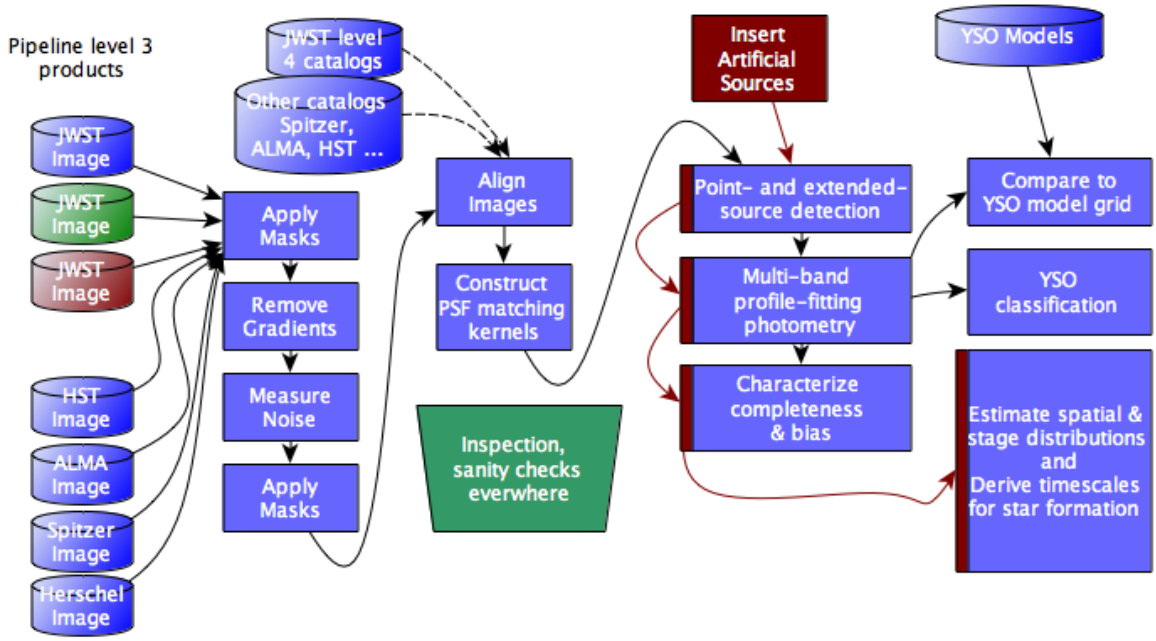


Fig. 5.1: Illustration of the data flow for a multi-wavelength survey of faint galaxies. The JWST inputs are the individual images from different bands and the JWST level 4 catalogs. The first steps involve inspecting and cleaning up the data, which might have scattered light from surrounding bright sources or might be slightly misaligned due to the use of different guide stars. Any mis-alignments between the JWST images and comparison images from other observatories will also need to be measured and corrected. Detailed photometric comparisons will require efforts to account for the different spatial resolution – which generally involves constructing PSF kernels to allow a higher resolution image to be convolved to lower resolution. Those psf kernels are then applied in various ways to detect source, measure their morphologies, and measure their fluxes. To characterize incompleteness and measurement biases, these steps are also generally carried out on artificial sources that were inserrted into the real images. The resulting catalogs are often matched to existing catalogs (e.g. of measured redshifts), and various physical quantities and statistical distributions are derived from the measurements.


Infrared slit spectroscopy of galactic/nebular objects
Dithered nods (ABBA or other pattern)
Dark subtraction + flat field
differencing of nod pairs
Shift and add or separate analysis of pairs
Image rectification for position-velocity work
Spectral extraction
Standard star spectrum
Wavelength calibration
Science spectrum
Transformation of std to science WCS
(Science/std) * std model
Alternatively, library spectral response function
Transform to LSR or helio-centric frame
Line ID
Transformation of line to velocity space
Line flux extraction
Continuum/ dust radiative transfer
Line/dust radiative transfer
Abundances, temperature, density, etc.

5.3 Imaging Young Stellar Objects in the Magellanic Clouds

The flow for multi-wavelength imaging of nearby objects shares many of the steps needed for faint-galaxy studies, including image alignment, registration and PSF-matching to other data sets (images and catalogs), source detection and characterization, and artificial source injection.



5.4 Need more use cases

Need more use cases!

CHAPTER 6

Technologies and Infrastructure

One of the major challenges of developing software is the rapidly changing landscape of software infrastructure. Much of this infrastructure is independent of astronomy, and it is often better to rely on existing solutions than invent new ones. Ideally one would like to select building blocks that are stable and well tested, without being on their way to obsolescence. It is silly for astronomers to build most of this infrastructure themselves, but some of it is astronomy specific.

In discussing infrastructure, it is important to consider the following:

- Building on infrastructure that is produced elsewhere creates a software dependency.
- Supporting multiple infrastructures that accomplish similar purposes (such as many different file formats or GUIs) can be costly.

In this section we discuss various items that can be categorized as infrastructure (because, ideally, they play no role in the numerical computations). In some cases, the roadmap presents choices that have already been made. In other cases, we discuss the current menu of choices to identify where further work needs to be done to make informed decisions.

6.1 Data Formats

Currently, most astronomical images and spectra are stored and transported in FITS format. A variety of formats are used for tabular data, including ASCII files with a variety of conventions for metadata, FITS tables, VO tables, and databases.

6.1.1 Relatively Certain

For the support of JWST observers, the data analysis tools should at a minimum support the following formats for "non-tabular" data:

- FITS

For the support of JWST observers, the data analysis tools should at a minimum support the following formats for tabular data:

- ASCII Tables - All of the following (already supported by astropy.io):
  - AASTex: AASTeX deluxetable used for AAS journals
  - Basic: basic table with customizable delimiters and header configurations
  - Cds: CDS format table (also Vizier and ApJ machine readable tables)

- CommentedHeader: column names given in a line that begins with the comment character
- Daophot: table from the IRAF DAOphot package
- FixedWidth: table with fixed-width columns (see also Fixed-width Gallery)
- Ipac: IPAC format table
- Latex: LaTeX table with datavalue in the tabular environment
- NoHeader: basic table with no header where columns are auto-named
- Rdb: tab-separated values with an extra line after the column definition line
- SExtractor: SExtractor format table
- Tab: tab-separated values

6.1.2 Under Consideration

FITS has fairly serious limitations for some types of metadata. ** Perry, add a few words here about what is under consideration as an alternative. **

HDF5 is a format capable of handling complex data. It is has seen limited use in astronomy, but is widely used outside of astronomy. It is of potential interest for all kinds of data. For tabular data in particular, the pytables package offers very efficient ways of querying and doing numerical computations on large datasets.

Relational databases. Python currently offers several ways to interface with relational databases, so nothing needs to be added to enable queries and database manipulation. One could imagine there would be interest in developing a simplified query interface to popular astronomical databases, but so far we have no specific use cases for this roadmap. Ureka is currently bundling the following python database libraries:

- MySQLdb
- PostgreSQL
- SQLite3

JSON is a lightweight data-interchange format that is increasingly used for transporting information from browswer queries. Libraries in python exist for reading and writing JSON. It would be worth having the the ability in astropy to import JSON tables, although the immediate pressure for this is low.

6.2 Data Abstraction

The concept in this roadmap is that most numerical calculations are done at root level by calls to numpy, (although when performance considerations arise, wrapping C code, using numba or bottleneck are under consideration). Regardless of the guts of the calculation, there is usually a need to pass metadata from one routine to the next, and it can be inefficient to read and write files from disk if the only purpose is to carry around metadata. Furthermore, it is often useful to have an array of measurements accompanied by other arrays such as uncertainties or flags, and to pass these associated arrays as one entity between different routines.

There is therefore the need to have a *data abstraction layer* that is distinct from the disk file format and that is optimized for in-memory operations.

There are several concepts for data abstraction that are likely to underpin the data analysis software developed in this roadmap:

- astropy.nddata provides a class and related tools to manage n-dimensional array-based data.

- astropy.table provides facilities for working with tables that are independent of the disk-storage format and handle missing data, descriptions, units and column formatting.
- stpipe.models provides a class and tools associated with model fitting.
- astropy.wcs provides utilities for managing world-coordinate-system transformations.

6.3 Parameter Handling

The current concept is to use configobj to handle file-oriented input of parameters.

6.4 Scientific and Numerical Libraries

Python is attractive as the main language for the data analysis tools in part because it already has a rich library of numerical, scientific and statistical tools.

6.4.1 Relatively Certain

The current plan is to build around the following libraries:

- numpy N-dimensional array processing.
- scipy Scientific computing library.
- astropy Astronomical utilities (The software in this roadmap will have strong dependencies on astropy, building on its core library. The vision is that most or all of the tools discribed here will become part of astropy itself.)

6.4.2 Under Consideration

There continues to be rapid evolution outside of astronomy in the numerical computing for python. Building around these libraries could reduce the coding effort needed for developing the analysis tools and could improve their performance. But each library creates a dependency, so we will have to choose carefully and probably not let all flowers bloom in this area. The following are under consideration:

- scikits These are add-on packages for scipy that are in various states of development. Two of these are already well developed and of particular interest: scikit-image, focusing primarily on image filtering, segmentation, and morphology, and scikit-learn, focusing on machine learning.
- cython Compiler for for adding C extensions within python code.
- numba Offers significant peformance enhancements to numpy using a just-in-time specializing compiler to LLVM, which is itself a compiler).
- bottleneck a collection of fast numpy array functions written in Cython.
- pytables a package for working with hdf5 files and efficiently manipulating extremely large amounts of data.
- pandas Focuses primarily on statistical analysis of tabular data, with similar basic functionality to the R statistical language (but without the huge package of contributed libraries).
- fftw Fast Fourier Transforms (faster than other options currently available in numpy or scipy).

Ureka is currently buldling the following additional libraries, not mentioned above, in its distribution. (Does any code in Ureka depend on these, other than scipy??):

- BLAS Basic linear algebra. Bundled with the standard scipy distribution.

- LAPACK Linear algebra package. Bundled with the standard scipy distribution.
- GSL The GNU Scientific Library.
- SymPy Symbolic mathematics library (Computer Algebra System).

6.5 Physical Units and Constants

The current concept is to continue to develop astropy.units to handle physical quantities, and to continue to develop astropy.time to handle time.

6.6 Interprocess Communications

The inter-process communication IPC infrastructure handles data transport between simultaneously-running, independent processes. Examples include passing cursor positions from a GUI to a task that might perform some analysis of the pixels near the cursor, or passing positions of sources to a routine that might mark them on an image display.

6.6.1 Relatively Certain

The following IPC protocols are likely to be supported:

- SAMP The Virtual Observatory messaging protocol.
- OMQ The protocol used by ipython.

6.6.2 Under Consideration

The following are under consideration:

- XPA The protocol used by the SAO ds9 image display. Since ds9 now can communicate using SAMP, it is unclear if XPA support is needed.
- REST A standard protocol for client-server communications, popular in web services.
- AMPQ another popular message-passing protocol. Quoting from the zeromq website AMQP and 0MQ have quite different goals. AMQP aims to commoditize existing enterprise messaging patterns, while 0MQ aims to create messaging patterns that can succeed at Internet scale...0MQ acts more like low-latency products.

6.7 Multiprocessing

Multiprocessing involves making use of either multiple cores on a single computer, or multiple computers on a network. For typical JWST tasks, it is likely that the work is embarassingly parallel in the sense that the tasks can be accomplished little or no communication between tasks running in parallel. There are numerous ways of dealing with such problems that do not require any astronomy-specific software development. Examples include:

- Condor a queuing system for distributed computing;
- Ipython parallel computing, which supports a variety of different parallel and distributed cmoputing models;
- The python multiprocessing module which can create pools of subprocesses that can inter-communicate.

Thread-based parallelism is not always efficient in python due to the Global Interpreter Lock. Many of the subtleties are addressed in this article by Jesse Noller.

For the purposes of this roadmap, the brief summary is that the data-analysis software in this roadmap will be amenable to multiprocessing using standard tools. In the initial implementation, it is unlikely that the tools themselves will use multiprocessing internally. However, multiprocessing may be one of the ways to address performance issues if re-coding in a faster language (e.g. C) doesn't solve the problem.

6.8 Special-Purpose Hardware

Examples of special-purpose hardware include Graphics Processing Units (GPUs), and Accelerated Processing Units (APUs). It is currently not envisioned that any the software in this roadmap will require the use of such hardware. It is conceivable that routines could be developed where the use of a GPU is an option if it is available, but that is not a near-term priority.

6.9 GUI Frameworks

The aim is to separate the computational layer from the GUI layer so that different user interfaces can be used to access the same functionality. There will nevertheless be default graphical user interfaces for inputting parameters or otherwise controlling task execution, and interacting with graphics and images. Standard libraries will be used for developing the interfaces and there will be a *gui abstraction layer* to allow different GUIs to interact with the software tools with a simple change of user preferences.

6.9.1 Relatively Certain

While not really a GUI per se, the *ipython notebook <http://ipython.org/notebook.html>* provides a full-featured environment for interactive scripting, and it will be a goal to make the data analysis tools work well in this environment and to use it for many of the tutorials.

6.9.2 Under Consideration

The GUI frameworks under consideration are:

- Qt A popular open-source cross-platform framework with python bindings.
- glue A python library for exploring relationships within and among related data sets.

6.10 Software Distribution

6.10.1 Under Consideration

STScI has recently developed Ureka as a unified distribution mechanism for STScI- and Gemini-developed python tools, as well as IRAF, pyraf, STSDAS, and a variety of supporting libraries. This is still in beta testing as of September 2013.

There are two other distribution mechanisms worthy of consideration:

- Anaconda the distribution mechanism used by Continuum Analytics. This distribution already includes many of the open-source scientific packages and libraries under consideration as dependencies for the data-analysis tools.

- Canopy the distribution mechanism used by Enthought. This distribution already includes many of the same libraries as Anaconda, and also includes astropy.

6.11 Documentation

6.11.1 Relatively Certain

The current plan is to use sphinx for documentation. This is already in use for astropy and STScI software and is widely used in the python community, as well as for other languages. It produces web pages as well as pdf, ebooks, and other formats and is able to import the same docstrings used to drive the *help* for the command-line interface to the data-analysis tools or for mouseovers in ipython notebooks. This roadmap has been written using sphinx.

It is likely that the documentation will use locally-developed and open-source extensions and styles for sphinx, such as numpydoc and sphinxcontrib-programoutput.

6.11.2 Under Consideration

The Ureka distribution currently includes the following in addition to sphinx and numpydoc, possibly for support of code that was written before sphinx appeared on the scene:

- Epydoc.
- Pygments.
- Docutils.
- Jinja2.

6.12 Testing

6.12.1 Relatively Certain

What's the plan? Nose?

6.12.2 Under Consideration

The Ureka distribution currently includes the following:

- Pandokia
- py
- PyTest
- unittest2
- shUnit2
- nose.

6.13 Graphics and Image Displays

6.13.1 Relatively Certain

For 2-D line graphics, the plan is to use matplotlib, a full-featured library that produces publication-quality graphics as well as providing the hooks for developing interactive GUIs or widgets and for animations. It also some 3D graphics and image display. It is probably not the platform for interactive image display as a replacement for ds9.

Independent of whether any new image-display tools are developed, ds9 will be supported. In particular, it will be possible to load and manipulate images from memory and interact with cursors and regions to perform data analysis.

6.13.2 Under Consideration

The following are under consideration:

- glue A python library for exploring relationships within and among related data sets.
- Mayavi A python library for 3D visualization built on the Visualization Toolkit (VTK).
- ginga A FITS image viewer under development at the Subaru observatory.
- APLpy, an astropy-affiliated package, is layered on top of matplotlib and provides routines for making full-color images and making various kinds of astronomical graphic overlays on images (e.g. coordinage grids and beam sizes).

The following are currently bundled with Ureka: - graphviz visualization software for network diagrams, flowcharts and other structural information. - PIL The Python Imaging Library, mostly useful for supporting standard non-FITS formats.

CHAPTER 7

Architecture

The *figure below* provides a high-level overview of the architecture of the data-analysis tools. The goal is to keep the computational tools lightweight and modular, and insulated from details of the data structure on disk and the details of the user interfaces. The tools will primarily work on data arrays stored in memory, with parameters passed as arguments. There will be a separate layer to read data in from disk and present the data to the tools via standard data models (which may in some cases be objects with methods). Reading and writing of data is accomplished in a separate layer, so that data can generally be passed in memory from task to task. This is different from IRAF, which tends to work on files, which must be read and written for each separate operation. Similarly, interaction with user-interfaces will be in a separate layer, which can be very lightweight: all it has to do is gather the parameters from the user-interface, call the routine with those parameters, and return the results. There will also be a separate event-driven interface to the same tools, which can be triggered by events such as mouse clicks in tools that are connected via some form of inter-process communication (IPC). To support multiple IPC protocols, there will need to be an IPC abstraction layer that makes these all look the same to the event-driven interface layer. Sometimes the tools will interact with image displays and graphics through IPC channels with two-way communication. Other times, the tools will send their output without setting up an event-driven interaction.

There are a few things to note about this architecture.

- It is not compatible with *client-side* computing in a web browser.
- The software is nevertheless compatible with *server-side* computing, where the computations are done either on the user's machine or on a remote server but the user interface is a web browser.
- The software is compatible with user interfaces that are not web browsers, which includes other types of graphical-user interfaces, along with editing of human-readable parameter files and invoking tools from a Unix command line or from within python.
- It is a goal to make it straightforward for a user to fire off a wide variety of computations via data selections in an image display or plot. This is handled by the event-driven interfaces which will then present these selections to the tools in the same way that a user would present them if the user had instead defined the selection mathematically in a script. (Indeed, one of the options in this layer should be to save the selections so that they can be edited and turned into scripts.)

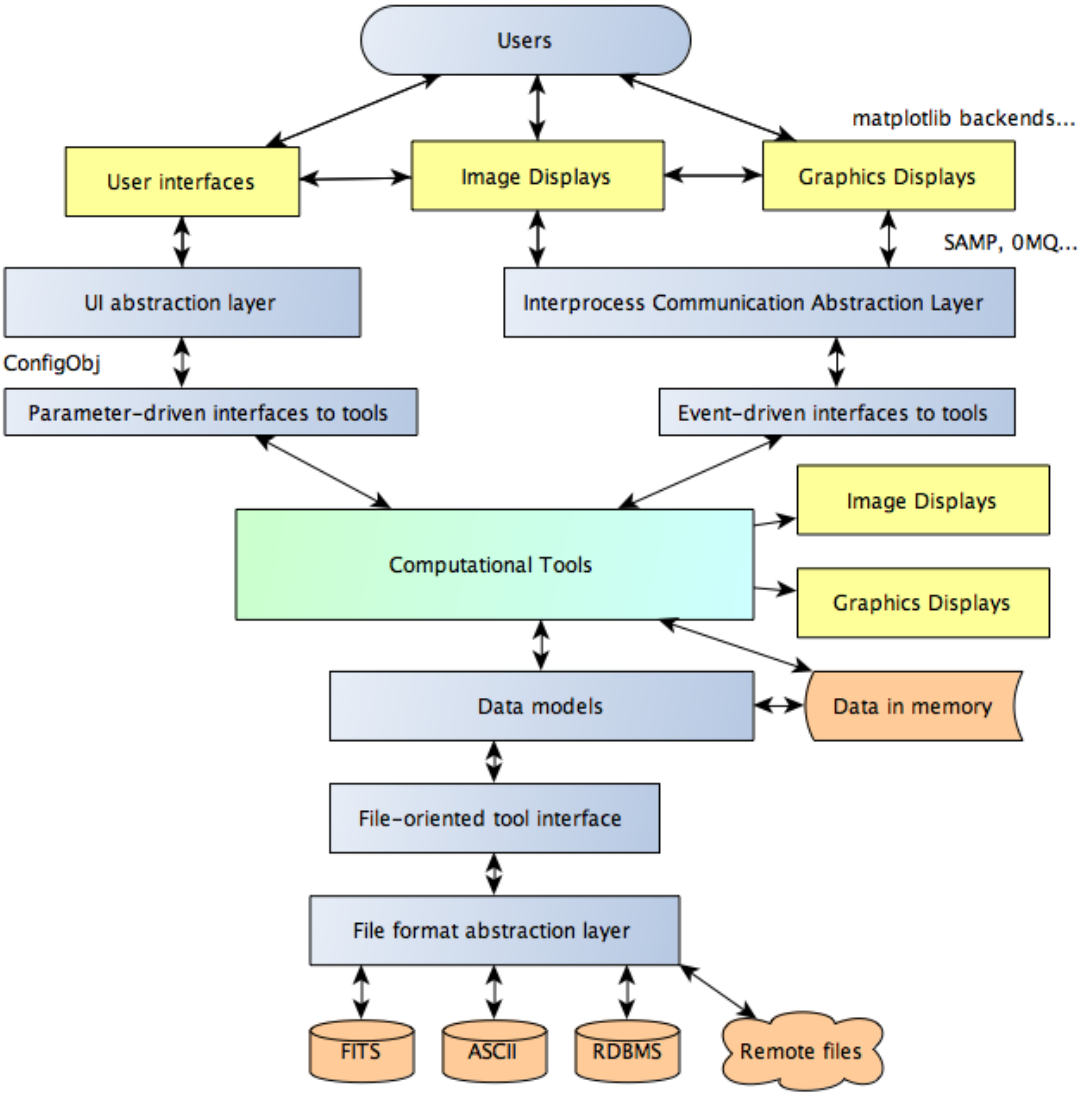


Fig. 7.1: A schematic view of the architecture of the data analysis tools.

CHAPTER 8

The Computational Toolbox

This section describes the specific software tools that are to be built. It should be viewed as a *concept document* not a *requirements document.* The discussion of each tool is brief, and comments are added about existing infrastructure that can be used to build each tool.

Many of the tools should work equally well with ground-based data or data from space observatories, but there is a need to include tools for standard kinds of ground-based data analysis. These are included, although the list may be less complete than for the JWST tools.

The figure below provides birds-eye view of the tools described in this section.

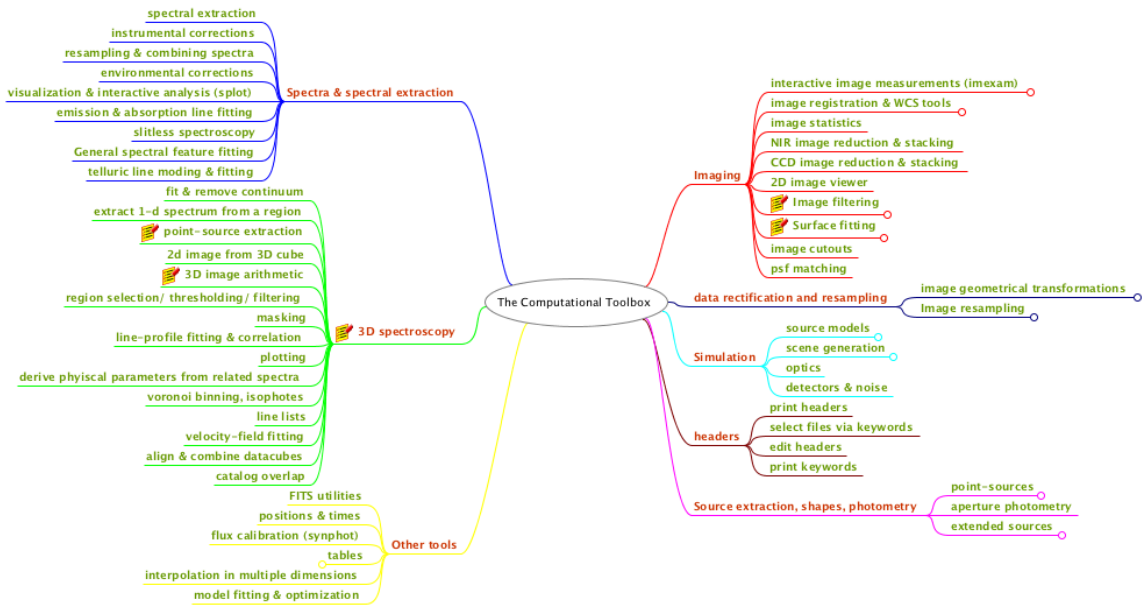


Fig. 8.1: A high-level categorization of the tasks described in this section. Many of the nodes in this map have sub-nodes and sub-sub-nodes that are not shown but would generally comprise separate tools. At the level of detail shown here there are ~50 categories of tools. In the end this will probably expand into of order 10^3 individual tools.

8.1 General-purpose multi-dimensional Array Analysis tools

The descriptions of the tools here are deliberately brief. The main goal is to highlight the basic functionality. In *many cases* basic functionality already exists within the various python scientific libraries. In those cases, the development effort for astronomers is generally focused on building interfaces to make them convenient to use on astronomical data sets. This includes:

- Consistent approaches to masking or rejecting data
- Consistent approaches to handling of undefined data and masks
- Consistent approaches to dealing with error arrays and data quality arrays
- Consistent approaches to dealing with metadata, including WCS information
- Consistent approaches to history and logging

8.1.1 Interactive Image Measurements

The new tools need to have the kinds of functionality currently supplied by IRAF's `imexamine` tool. This includes:

- centroids
- contours
- 3d surface plots
- 2d cuts
- localized histograms
- aperture photometry
- basic statistics

8.1.2 Basic Statistics

The includes basic statistics like mean, median, standard deviation, minimum and maximum, with optional masking, setting of a floor and ceiling on input values and rejection of outliers.

8.1.3 Filtering

There is already rich set of image filtering tasks in the scipy ndimage module, the scipy signal and in scikit-image. However, these do not always deal gracefully with image masks, undefined values, or error arrays. The astropy nddata module is aimed at addressing this issue. Filtering includes the following, all in N dimensions, where N is at least 3 for JWST data:

- smoothing
- convolution
- gradient detection
- wavelets

8.1.4 Surface Fitting

A very common operation for astronomical images is to fit a relatively smooth line, surface, or multi-dimensional manifold to a data set (with optional masking and iterative data rejection). This is closely related to smoothing, except that the surface is parametized somehow. The following parametrizations are common enough that they should be provided:

- orthogonal polynomials (e.g. chebyshev)
- splines

8.1.5 Interpolation

There are many, many applications that involve interpolation in dealing with astronomical images. A typical application involves estimating the sky background associated with sources in the image. A set of background estimates from "clean" regions in the image are usually interpolated to the positions of the sources. Another example is in applying geometrical transformations to images. The fluxes on a rectangular pixel grid are used to estimate the fluxes that would have been measured on a different pixel grid.

Relatively Certain

The following kinds of interpolation should be included:

- multi-dimensional linear and polynomial interpolation
- spline interpolation

Under Consideration

- Inverse distance-weighted interpolation, and variants that use local neighbors
- Natural neighbor interpolation
- Radial basis function) interpolation
- Gaussian processes regression, or Kriging, commonly used in geospatial modeling
- Band-limited interpolation (e.g. sinc or Lanczos)
- Drizzling

8.1.6 Geometric transformations and resampling

The toolbox will include a wide variety of ways to transform images. Many of these involve resampling, and interpolation. Most of the framework for doing this already exists in python numerical libraries. For geometric transformations, easy things should be easy to perform:

- Block averaging or block replicating
- Linear transformations (shift, magnify, rotate, transpose)

Harder things should be possible:

- Affine transformations (preserving straight lines and planes)
- Arbitrary geometric distortions

- Standard map projections (extensive support for various projections exists in matplotlib, but it is currently hard to extract the projected data for other uses).

It is would be useful to make making the different interpolation options easily accessible to the geometric-transformation tools.

8.2 Imaging

In this section, we focus on tools that are mainly needed for undispersed images of astronomical sources. Some of these tools apply to images that are spectrally dispersed as well, but most of those tools are discussed in the spectroscopy section.

8.2.1 Image Registration & WCS Tools

Even if JWST images are perfectly rectified by the standard pipeline, it will be frequently necessary to register other images to the JWST images. The geometric distortion properties of these images are sometimes unknown and must be derived using the data themselves. The various tasks involved (many of which are in the IRAF `immatch` package) include:

- image distortion fitting
- image rectification
- aligning images
- combining images to a common grid
- matching catalogs (triangle matching)
- image cross-correlation

8.2.2 NIR Image Reduction & Stacking

This would replace the functionality of the IRAF `irred` package.

8.2.3 CCD Image Reduction & Stacking

This would replace the functionality of the IRAF `ccdred` package.

8.2.4 Peak Finding

Peak finding is a typical operation on astronomical images. It is the first step of any source detection and photometry package, but is often useful by itself. Routines to do this in scipy and scikit-image would serve as a starting point.

8.2.5 Image Segmentation

Image segmentation is often the next step after finding peaks. Pixels are assigned to sources via some algorithm. The routines for image segmentation in scipy and scikit-image serve as a good starting point. They provide a rich set of options for tasks such as labeling pixels associated with HII regions or clusters within an individual galaxy.

8.2.6 Image Cutouts

Obtaining cutouts of selected sources from a database of larger images is a common operation, but one that has not been standardized. Most organizations serving out data now support the IVOA Simple Image Access Specification (SIAP). However this does not solve the problem of how to organize the data or determine footprints. The IVOA has a draft of a footprint specification open for comments. Many of the institutions serving large data sets support SIAP and are headed toward common standards for footprints. However, tools are needed to help individuals and small groups accomplish the same functions on their smaller datasets for their own use.

8.2.7 PSF Matching

The need to match the point-spread function between different images is very common, and a place where existing tools need improvement. Typically one wants to determine the convolution-kernel that, when applied to a high-resolution image, produces an image with the point-spread function of a lower-resolution image. Examples include matching the PSFs of a JWST NIRCam image to that of a MIRI image. Or matching a JWST image to a ground-based image. The PSF kernel may be spatially varying across the image. Sometimes one has PSF models for each instrument that can be used to construct the PSF kernel. Other times, one must rely on sources in the image, which may or may not be point sources.

A popular algorithm in ground-based astronomy is the Alard-Lupton method of decomposing the kernel into a set of Gaussian basis functions. This is widely used in the supernova-search community. It works very well for typical ground-based PSFs but the Gaussian basis-function may not be the best choice for PSFs with a lot of structure or broad wings. Becker et al. (2012) discuss other methods and promote a method using regularized delta functions. Regardless of the method, the toolbox needs to have efficient, flexible tools for selecting the sources to use for PSF matching, iteratively and perhaps interactively rejecting bad ones, evaluating the results, extracting the kernels, and applying them for photometry or image subtraction.

8.3 Spectra and Spectral Extraction

8.3.1 Spectral extraction

At the simplest level, spectral extraction means turning a two-dimensional dispersed image into a one-dimensional spectrum. This involves separable steps:

- Identifying the pixels to be used for source and background;
- Co-adding the pixels in the cross-dispersion direction according to a specified weighting (and masking) scheme and subtracting background.

Generally speaking, the task of spectral extraction does not include:

- Converting pixels to wavelength
- Converting counts to flux

Nevertheless, there need to be user-oriented tools layered on top of spectral extraction that apply the dispersion solution and flux calibration.

While the JWST pipeline will generally attempt to extract scientifically useful 1D spectra of the objects in the field, it is very likely that astronomers will want to control the extraction in different ways. Some of this can be done by tweaking pipeline parameters, but there also need to be general-purpose tools for doing this driven from parameter files or GUIs or regions on an image display.

This software can become quite complex, depending on the level of sophistication.

A simple approach involves selecting a spectrum on a rectified image (where all spectra are straight lines and where there is a linear wavelength scale per pixel), extracting a rectangular region from this image and summing along columns or rows. This is very useful for quick-look analysis, but often insufficient for the final scientific analysis.

The next level of sophistication selects a curved spectrum on a non-rectified image, where the wavelength scale may not be linear. Either the software traces the spectrum, or has a pre-defined map of the spectral traces so that it can extract the spectrum. It then applies the dispersion solution and either constructs a 1D array with a linear (or logarithmic) relation between wavelength and pixel number, or it constructs a table where each row has a wavelength.

There is also a further level of sophistication, which uses a rectified, co-added image from multiple exposures as a tool by which to identify source and background regions. However, these regions are then transformed back into the reference frame of the original individual (probably dithered) exposures and the spectral extraction is carried out there. This minimizes number of resampling steps carried out on the data and simplifies the propagation of uncertainties.

It is a goal to support all three methods of spectral extraction, with both GUI-specified extraction regions and regions specified via input parameters.

8.3.2 Resampling & combining spectra

Resampling a 1D spectrum can mean taking a spectrum on one wavelength scale and putting it on another scale. Or it can mean taking a table of fluxes and wavelengths and turning it into an array with a specified wavelength scale. Or it can mean taking a model spectrum and interpolating it to find its values at the wavelengths of an observed spectrum. Resampling involves interpolation and is therefore an approximation. There are many different ways to interpolate, each with its own strengths and weaknesses. Many interpolation methods are already supported by scipy and numpy, so the primary goal is to provide interfaces to these interpolation methods so that they can be easily applied to astronomy data sets with attention to metadata, error arrays and masks.

Resampling 2D spectra is more complicated and often requires a detailed specification of the image distortions and wavelength calibrations of the instruments used to obtain the data. At the lowest level it is just 2D interpolation. The main challenge is to develop a simple, yet general mechanism to specify the image geometries.

Combining spectra involves co-adding or finding some robust measure of the mean of data from several different sources. The weights often depend on the uncertainties and masks. The co-added spectra should have associated uncertainties when it is possible to compute them.

8.3.3 Visualization & interactive analysis (splot)

An important functionality for the Graphical User Interface(s) is to provide interactive viewing and manipulation of one-dimensional spectra. The type of functionality needed includes that available in the splot package, SPECVIEW, VOSPEC, the STARLINK SPLAT package or a variety of individually-maintained IDL tools.

Perhaps more important than the specific functionality is to develop a framework to make it easy for astronomers, who are generally not experienced GUI developers, to add functionality to the GUI for a particular science application.

8.3.4 Emission & absorption line fitting

The software tools need to include a variety of fitting functions and fitting methods. Fitting functions should include simple Gaussians and Lorentz and Voit profiles, as well as more elaborate functional forms. The software should allow for convolution with a wavelength-dependent line-spread function. It should provide a variety of minimization routines and ways of treating uncertainties and missing data. It should work well in regimes where the assumption of Gaussian uncertainties on the fluxes is not valid (e.g. for very low counts/pixel). It should have ways to convert the results into physical units. It should be straightfoward to use the same software to assess uncertainties via resampling or Monte-Carlo techniques.

8.3.5 General spectral fitting

In addition to fitting emission and absorption lines, there need to be tools for estimating or fitting continua (used in line-fitting). This includes allowing the user to interactively select regions to use for continuum fitting, as well as providing algorithms for iteratively masking emission and absorption features as part of the determination of the continuum spectrum.

There also need to be tools for fitting heuristic or physical models to spectra, e.g. along the lines of the contributed package specfit in STSDAS.

8.3.6 Line lists

Relatively Certain

The toolbox should contain convenient, well-documented lists of the wavelengths of lines commonly seen in astronomical objects.

Under Consideration

It is less certain how far to push into the astrophysics of spectral analysis. It might make sense to provide common line ratios (e.g. from Case-B recombination), and to provide simple interfaces to more sophisticated tools like CLOUDY.

8.3.7 Slitless spectroscopy

HST and JWST each have two slitless spectrographs. The JWST NIRSpec MOS can also mimic a slitless spectrograph. The lack of a slit creates several challenges for slitless spectroscopy:

- Spectra from several sources can overlap;
- Overlap can include a zeroth-order image or spectra from higher orders;
- Spectra taken at different rotations can be used to overcome some of the effects of blending;
- The wavelength calibration depends critically on knowing the source positions; and
- The convolution kernel for a spectral feature can be quite broad because it not truncated by a slit.

For HST, slitless spectroscopy has been supported by the AxE package. The 3D-HST Treasury Program developed a separate set of tools. Regardless of the ability of the JWST pipeline to meet all the calibration needs, tools for observers to inspect the original 2D spectra and extracted 1D spectra, manipulate and re-run the extraction, model the line-spread function, and deal with the source confusion are essential.

Searches for emission lines (such as Lyman-alpha lines from very distant galaxies) are hopeless without dealing with most of these issues. It would clearly be of interest to observers to have a tool that searches for emission lines in the data automatically, accounting for the difficulties of spectral overlaps and the fact that the S/N for a significant detection is dependent on the size of the line-emitting region and the underlying background due to sky and galaxy continuum.

8.3.8 Instrumental corrections

For JWST and HST, the instrumental corrections are the job of the pipeline. The pipeline parameters and tasks themselves can be modified and re-run by the user. These include:

- converting counts to calibrated fluxes in physical units; applying sensitivity, flat-field and illumination corrections

- converting pixel positions to wavelengths

To the extent that the assumptions for different instruments can be standardized, it would be useful to include the building blocks for applying instrumental corrections for other instruments in the general set of tools. However, because this depends on standardized formats for calibration information, it will require broad community involvement. One way forward would be to develop a prototype of applying insrumental corrections to a popular ground-based spectrograph, using the same calibration philosophy that is used for JWST, and document this as a template for other instruments.

More importantly, there are tasks involved in deriving these corrections that are in common to HST and JWST instrument scientists and the large community of astronomers needing to calibrate other instruments:

- deriving the instrumental sensitivity from standard-star observations;
- deriving the instrumental flat field;
- deriving instrumental illumination corrections;
- deriving the instrumental dispersion solution; and,
- deriving the absolute wavelength calibration

In the case of ground-based observations, some of these calibrations need to be applied night by night, while for the space instruments there is no need to worry about atmospheric variations and other time dependencies (e.g. thermal changes of the instrument) tend to be much slower.

8.3.9 Environmental corrections

It is useful to separate environmental corrections from instrumental corrections, with the distinction that the environmental corrections generally happen outside of the telescope. These include:

- Correcting spectra to heliocentric, galacto-centric or cosmic-flow corrected velocities;
- Shifting spectra to the rest frame;
- Correcting for interstellar reddening and extinction;
- Correcting for scattered light;
- Modeling and/or removing telluric features; and,
- Correcting for airmass and atmospheric extinction

The last couple items on this list are unique to ground-based instruments. The rest are important for JWST and HST, although some of these are done in the pipeline.

8.4 3D Spectroscopy

In 3D spectroscopy, instead of having a two-dimensional image, the scientist is presented with a three-dimensional image, with two dimensions of spatial information and one dimension of wavelength information. There are some unique challenges to these kinds of data. However, there is also a lot of commonality with 1D and 2D spectroscopy (for example in extracting 1D spectra or fitting spectral features).

For JWST 3D spectra (in NIRSpec and MIRI), an image slicer reorganizes the incoming light to provide non-overlapping spectra of *slitlets* – small rectangular slices of the sky. These dispersed spectra are collected in a 2D detector and can then be re-assembled into a 3D data cube in the pipeline, after correcting for various distortions in the optical system. Generically, the data-analysis tools allow astronomers to view, select, and analyze data in this 3D space.

It is worth pointing out that this is not the only way of storing the 3D information. It is the most convenient way for a user to view the 3D data, but it involves resampling. Provided one has the tools to map from the original detector reference frame to the rectified 3D image frame, one can imagine developing tools that allow viewing and selection in a rectified 3D image, but do the extraction and analysis on the original pixels. Such tools are more challenging to develop, but are part of the trade space being considered.

In this section, for the sake of brevity, we concentrate mostly on tasks that are specific to the 3D nature of the data. Once one has selected regions to used for the source spectrum and the background, many of the tasks (e.g. fitting spatial or spectral models) are functionally equivalent to the 1D or 2D cases.

8.4.1 3D image arithmetic, filtering, thresholding, and masking

Basic tools for arithmetic operations on 3D data cubes will be provided. This includes simple operations like add/subtract/multiply/divide as well as applying more complex functions. All of this generic functionality already exists in numpy and scipy, so the development effort will be in developing interfaces that do sensible things with metadata, errors and masks.

The basic tasks for thresholding, masking, peak-finding, convolution, filtering, and general signal processing also mostly exist in the numpy/scipy infrastructure and will not require a huge effort to incorporate into 3D spectroscopy tools.

The standard mechanisms for slicing arrays, doing statitistics, and projecting/summing along dimensions will be available.

8.4.2 Align & combine datacubes

An alarmingly common operation will be to try to align and combine data cubes. This could be to produce spectra of the same sources over a wider spectral range (i.e. combining in the wavelength direction) or to produce co-added spectra of sources taken in separate (possibly dithered) exposures. The JWST pipeline will deal automatically with some of the latter, when the images fall in a pre-defined association. However, users will no doubt want to create their own associations.

If the image alignment is not already known, the tools described in the Imaging section above can be used to align extracted 2D projections of the 3D data cube. If this is shift, rotation and scaling, the alignment can be fairly simple. If it involves geometric distortion, it can be quite complex.

The image combination operation can be challenging. In the wavelength direction, the spectra may overlap and may have different wavelength scales. The user may want to weight the spectra via some algorithm and may want to apply masking. The user will generally want to produce an error array to accomany the combined data dube. These operations almost certainly involve interpolation, and therefore involve some science-specific decisions. The same considerations apply to the spatial dimension, which could in principle include image distortions. The challenge is to create a set of tools that include:

- a simple interface for quicklook or for cases where the exact details of the image combination are unimporant;
- more elaborate interfaces and documentation to allow users to tailor the image alignment and combination to their science goals.

8.4.3 Fit & remove continuum

Fitting and removing a continuum is a generic task that applies to 1D, 2D and 3D spectroscopy. In a 1D spectrum, the astronomer usually specifies regions to use for fitting the continuum – either interactively or via some algorithm that iteratively detects emission and absorption features and masks them out. The fitting algorithm could be parametric, or could involve interpolation. The choice depends on the data and the science application. Tools need to be flexible enough support many options.

The philosophy of region-selection selection in the wavelength dimension is the same for 2D or 3D data, but now the spectra have to be combined somehow either to provide a visual display of a 1D co-added spectrum of a given spatial region to allow the astronomer to select the regions to use (or mask out) in the fitting, or there needs to be a specification of how the iterative masking algorithm averages over the spatial pixels. Sometimes there will be spatial as well as spectral masks (e.g. to remove foreground sources).

In a 1D spectrum there is one continuum spectrum. In the 2D and 3D cases there are more choices to be made. In 2D spectral analysis, there could be a single 1D continuum spectrum that is modulated by the spatial profile of the source, or there could be separate continuum spectra for each pixel in the spatial dimension. The approach depends on the S/N and the science application. Similarly for a 3D data cube, the science and S/N may dictate whether one tries to derive and divide out a single continuum spectrum for all spatial pixels, a continuum spectrum that varies smoothly in some fashion along spatial coordinates (e.g. via fitting or interpolation) or a separate continuum spectrum for each spatial pixel.

8.4.4 Extract 1D spectra from regions

A common operation on 3D data cubes is to extract 1D spectra from specified regions. These regions could be selected:

- interactively on the 3D visualization tool
- via adaptive spatial binning such as Voronoi tesselation
- via thresholds in S/N or isophotes, applied to specific wavelength ranges
- via scripts
- via tables.

These regions can have a variety of shapes, which may or may not be connected. The tools should allow the selection of background regions as well.

The tools should support different ways of combining spectra (e.g. sums, means with and without weighting, or medians). The tools should return error arrays and bad-pixel masks when relevant.

8.4.5 Construct a 2D image from 3D cube

A typical operation on a 3D data cube will be to construct a 2D image of some specific region in wavelength (e.g. emission lines) This operation may include some interpolation over masked pixels in the wavelength dimension, so is not quite as simple as summing along one dimension of a 3D array.

One use of such a projected image is to drive automated algorithms for 1D spectral extraction from the data cube. For example: find peaks in the image, create extraction apertures around those peaks and extract the associated 1D spectra. The same tasks used for peak-finding and image segmentation described earlier in the Imaging section can be used for this.

8.4.6 Point-source extraction

The 1D extraction of the spectra of point sources is a special case.

Relatively Certain

The tools should support optimal extraction weighted by the PSF profile, or simple aperture extraction.

Under Consideration

More challenging, but possible, is to derive the 1D individual spectra of point-sources in very crowded fields where the PSF wings substantially overlap. This involves simultaneously fitting for the fluxes of the overlapping sources, and results in 1D spectra that have significant covariance with the spectra of their neighbors.

8.4.7 Line-profile fitting & correlation

3D spectroscopy is often used to analyze kinematics. At the root level, this involves extracting and analyzing 1D spectra of spatial regions within the image, but the desired data product returned to the user is generally not a big table of numbers, but rather some more informative visualization. Use cases include:

- Fit a Gaussian to and derive centroids and higher order moments of an emission line profile to derive gas kinematics of emission. The input is a 3-D datacube sub-region. The output is a set of 2-D images that describe the Gaussian fits, and velocity centroid, dispersion and higher order moments of the profile.
- Fit *two* Gaussian profiles to an emission line profile to derive gas kinematics of emission. The input is a 3-D datacube sub-region. The output is two sets of 2-D images that describe the velocity centroid, dispersion and higher order moments of the profile.
- Fit a user input 1-D spectrum (e.g., instrumental line profile model) to an emission line profile to derive gas kinematics of the emission. The input is a 3-D datacube sub-region and 1-D spectral template. The output is two sets of 2-D images that describe the velocity centroid, dispersion and higher order moments of the profile.
- Correlate the IFU datacube to a template spectrum to generate stellar kinematic diagnostics from absorption line spectra in a continuum source. Inputs are the 3-D datacube region plus a template 1-D spectrum, outputs are 2-D image maps of the velocity and dispersion.

It will clearly be beneficial to have user-friendly GUI-driven tools for this kind of work, layered on top of the simpler tools that do the spectral extraction and fitting.

8.4.8 Interactive 3D data-cube Vizualization

8.4.9 2D Plotting of 3D data

The 3D-spectroscopy toolbox needs to contain a variety of types of two-dimensional plots. There is a clear need for interactive plots to work with the data-analysis tools themselves. There is also a clear need for publication-quality graphics.

Use cases include:

- Create figures using two 2-D images -– e.g. using a continuum emission image, overplot contours from the other image. User inputs include display levels for image and contour levels and a color table. (This is already largely doable in the current version of DS9, although it could be made more convenient).
- Create a velocity channel map using a 3-D input cube
- Create a 3-D plot view with parameters from the 3D visualzation tool inputs
- Create a figure presenting multiple 2-D images (not linked in velocity) using input from the 3D visualization tool.
- Create a plot with multiple 1-D spectra from the 3D visualization tool.
- Create recipes or canned wrapper macros that permit creation of some of these figures with fewer clicks by the user, and no external saves.

Most of these plot types are easily supported in matplotlib (in both quicklook and publication-quality forms), so the software development can be concentrated primarily on providing the linkage to the 3D visualization tool, and on layers to make use of data models and metadata to put coordinate systems and physical units on the plots.

8.4.10 Derive phyiscal parameters from related spectra

Under Consideration

Deriving physical parameters from spectra was beyond the scope of most IRAF tasks, but is clearly of use to astronomers. The key question is whether tools can be made generic enough that they support a broad community of users. Specific use cases include:

- Provide the means to take a few inputs — such as emission line images or emission line datacubes -– to derive physical parameters (i.e., n_e from [Fe II] line ratio maps or A_v from H2 line maps).
- Have several canned relations available automatically (such as gas parameters for [Fe II] and H2 line regions, or common physical parameters derived from optical emission lines in red-shifted galaxies).

8.4.11 Velocity-field fitting

Under Consideration

A fairly common operation in galaxy research is to fit a velocity field with a model (e.g. tilted disks, with or without constraints on the one-dimensional rotation curve). Ideally, the fitting is done on the ful data cube, but the visualization is on lower-dimensional slices. Tools for such modeling exist, particularly in the radio community. Further investigation is prudent before developing a new tool.

8.4.12 Catalog overlap

Tools should be provided to ingest catalogs and use them to overlay on 3D data cubes or drive spectral extraction.

8.5 Source extraction, morphology and photometry

Various types of source extraction have already been mentioned above in the imaging and spectroscopy sections. The toolbox needs to include well-documented tools for the following:

- Aperture photometry
- PSF construction
- PSF-kernel construction for matching images with different resolution
- Multi-band crowded-field photometry with or without positional and flux priors (incorporating priors is an area of weakness of existing codes).
- Point-source detection in crowded field
- Faint-galaxy source detectection
- Faint-galaxy photometry
- Faint-galaxy morphology

This is an area where very capable tools already exist both inside and outside of IRAF. This includes:

- SExtractor, a stand-alone monolithic C program for faint-galaxy detection and photometry driven by a traditional parameter-file interface.
- GALFIT for fitting parametric models to galaxy images.
- GALAPAGOS A set of IDL scripts that tie together SExtractor and GALFIT.
- DAOPHOT for crowded-field stellar photometry.
- DOLPHOT for crowded-field stellar photometry.
- DOPHOT for crowded-field stellar photometry.
- MATPHOT for crowded-field stellar photometry.

The stellar-photometry codes all take somewhat different approaches to deriving and parametrizing the PSF and fitting the PSF to the crowded images. A version of DAOPHOT exists within IRAF.

Because these are very capable (and complex) codes, there is currently not much urgency in the HST+JWST community to create new codes. That said, many of these codes are maintained by a single person, with no guarantee they will be maintained indefinitely, and there are some limitations of current codes.

In summary, these capabilities are needed whether by ensuring maintenance and interoperability with a existing codes, or developing new tools.

Regardless of whether exiting monolithic packages for doing star and galaxy photometry are supported as part of the toolset for JWST observers, it would be very valuable to provide modular tools for astronomers to experiment with new algorithms. The tools for image filtering, segmentation, and morphology within scipy and scikit-image are already quite powerful, although it is unclear that they would be fast enough for a production environment.

8.6 Simulation

Simulations are absolutely essential for analyzing modern astronomical data. Simulations are the best way to estimate the the flux biases, uncertainties, selection functions, and completeness of samples of stars or galaxies measured in an image or a survey. Typically one would like to either construct purely simulated images with realistic simulations of the S/N, PSF, and instrumental charactiistics, or one would like to insert simulated objects into the real images. To prevent the uncertainties in the analysis from dominating the final uncertainties, one will typically insert many more sources into the images than true sources, but do this a few sources at a time to keep the crowding more or less the same.

In IRAF, simulations have been supported by the ARTDATA, which is flexible and quite fast. This package does not include tools for modeling model instrument characteristics.

A flexible simulation package should include the following:

- Source shapes (point sources, various galaxy models)
- A library of astronomical source spectra
- The ability to apply an instrumental response function to spectra (e.g. pysynphot)
- Distribution functions to describe populations of sources with varying fluxes or spectral properties (e.g. luminosity functions)
- Spatial distribution functions
- At a minimum, the ability to convolve with point-spread functions
- Poisson and Gaussian noise models

Certain aspects of this are already supported by pysynphot, which takes input spectra and instrument throughputs and calculates count rates.

A more sophisticated simulation suite would attempt to model the full effects of the environment, optics, and detector. The LSST Image Simulator, treats each incident photon individually, accounting for the effects of the atmosphere and optics, non-uniformities in the detectors, charge diffusion, etc. Various proprietary simulators exist for the JWST instruments, but there are currently no plans to support these for general observers. Nevertheless, it is important to have reasonably high-fidelity simulations that include the specific characteristics of the instruments in order to develop the JWST pipeline. It takes more effort to develop these for general use, but they would undoubtedly be useful to many observers.

It would also be extremely useful for the simulation suite to provide access various kinds of sky models:

- A Milky Way model for star counts
- A Milky Way model for extinction
- A diffuse sky-background model (including zodiacal light, galactic diffuse emission, and extragalactic background radiation)
- Hertsprung-Russell diagrams and a tool to build stellar populations
- A tool for building synthetic galaxy spectra for an assumed star-formation history and chemical evolution
- Cosmological galaxy-evolution simulations

The situation for these is very similar to that for photometry packages. There are many existing codes that work well and are widely used in the community. There is thus little pressure to create new codes and at least the initial focus should be providing access and interoperability.

8.7 Other tools

This section lists other astronomy-specific tools not mentioned above that will be commonly used. Many of these tools exist in some form in astropy already.

8.7.1 Utilities for reading, writing, and manipulating FITS files and headers

The standard for python has been pyfits for the past few years. This is now incorporated into astropy as astropy.io.fits. In addition to reading, writing, and creating files, it offers standard pythonic ways of accessing the metadata in the image headers.

The equivalent of the following IRAF tasks are needed:

- hselect - select files on disk via header keywords using various boolean operations
- hedit - batch editing of keywords
- eheader - an STSDAS task for editing keywords in emacs or vi

When fits IO is needed in C it will be provided by CFITSIO.

8.7.2 Utilities for reading, writing and manipulating tables

Astronomers deal with tabular data constantly. The storage formats includ ASCII files with various formats, FITS, VO Tables, and databases. For small tables, the routines for manipulating the table and doing operations (plotting statistics, model fitting, etc.) can be independent of the table format once the table is read in. For large data sets that do not fit in memory, this is not as straightfoward.

Various utilities exist within python and astropy for dealing with tabular data, all of which have slightly different purposes:

- astropy.table,
- pytables,
- pandas.

There is work to be done to improve the convience to astronomers for:

- reading and writting various formats
- dealing with missing data
- dealing with uncertainties
- interpolation (with a rich set of options)
- gridding irregularly sampled data

This is a place where well-designed tools with good documentation and tutorials can save the community a lot of time.

8.7.3 Utilities for dealing with astronomical times and positions

Astropy now includes tools for manipulating world coordinate systems in astropy.wcs. It includes basic functionality for manipulating times and dates in astropy.time. The open-source pyephem packages is well supported and offers utilities for time and position as well as for computing epheremerides of the solar-system bodies and satellites.

Under Consideration

Perry include something about SOFA?

If the license on the SOFA astronomy library can be made compatible with the astropy license, then it will be incorporated with a python wrapper.

8.7.4 Utilities for model fitting and optimization

This section needs work....

Model fitting is an essential part of the toolbox. It is required in many steps of instrument calibration, as well as in the analysis and interpretation of science data. It requires both full-featured GUI interfaces that allow data selection and interactive manipulation of fitting parameters, as well as fast and robust routines that can be used repeatedly in fitting large data sets.

The tool set should certainly include various kinds of standard functions such as Gaussians and polynomials. Within IRAF there are ~80 different tasks that involve fitting, from general curve and surface fitting like curfit, polyfit and imsurfit, to more specialized routines for fitting continuum in spectra, fitting sky background, fitting standard-star photometry, or fitting ellipses.

The scipy optimize package includes a variety of optimization options. Eric Tollerud, one of the founders of astropy, has provided a somewhat higher level framework for model fitting, including a customized GUI in PyModelFit.

There are a couple of python versions of Craig Markwardt's popular IDL fitting library.

CHAPTER 9

Graphics and Visualization

9.1 2D image display

Probably the most popular 2D image display tool in use by the US community is ds9, from SAO, which is a stand-alone application actively maintained at SAO. It has the ability to communicate with other tasks using XPA or SAMP. Other sophisticated image display utilities for astronomical images include:

- Aladin
- ximtool
- GAIA::SkyCat

The default will be to support image tools that communicate using SAMP. SAMP is a message-passing protocol, but doesn't specify the functions that should be supported by the tools on either end or how to invoke those functions. Ds9, for example, has 1150 different `xpaget` and `xpaset` commands, supporting 96 different areas of functionality. It would take a considerable effort to implement all of this functionality afresh in a new tool. Nevertheless there are some limitations of ds9 that make it worth considering. In particular, it is not particularly fast at loading images or zooming and panning, and is not designed to deal with very large images that do not fit entirely in memory. Both are in contrast to many tools available on the web that pan and zoom quite fast in extremely large images.

Because there are good image viewers available, building a new 2D image viewer is not currently a high priority for STScI. However, it would be very useful to maintain a wish-list of features to include in any such tool (in addition to supporting all of ds9's features).

9.2 3D image display

Observers using the MIRI or NIRSpec spectrographs on JWST will need a full-featured 3D image display tool. This must interact with analysis tools and 2D graphics tools and provide a variety of options for data selection and visualization. Under consideration are:

- ds9
- glue
- join forces with radio astronomers to provide a tool to support both communities.

9.3 Interactive 2D & 3D graphics

9.3.1 2D graphics

Within python, the standard is matplotlib, which has had heavy STScI involvement (and some funding) in its development. It supports multiple "backends" to build in device and GUI independence. Efforts are underway to make it possible to plot interactively in a web browser. Most users interact with matplotlib via pyplot, which provides a MATLAB-like plotting framework.

Relatively certain

For 2D graphics, the plan is to continue to improve matplotlib and use it as the basis for both interactive and publication-quality graphics.

Under consideration

Matplotlib's greatest weakness is probably speed, which makes it not particularly suitable to video frame-rate animations or real-time updates. While most astronomers don't need these capabilities for day-to-day data analysis, they are occasionally needed. For speed, it is useful to consider hardware-accelarated graphics, such as that provided by NodeBox for OpenGL. Another potential weakness of Matplotlib is that it is was not originally engineered to operate in a web browser. The Bokeh visualization library, under development at Continuum Analytics, aims to implement the Grammar of Graphics, which has become popular in statistical packages such as R.

There is also some interest in having the capability of making dynamic graphs, such as those provided by the d3 javascript library.

9.3.2 3D graphics

Relatively certain

Matplotlib provides some 3D capabilities within the mplot3d package.

Under consideration

There are other options for 3D visualization as well, including:

- Mayavi
- VPython (licensing issues?).
- PyQtgraph 2D and 3D visualization.
- Mayavi 3D visualization.
- VTK 3D computer graphics with python wrappers.
- Paraview An open-source python-scriptable 3D visualization application aimed at very large datasets.
- yt A volumetric data-analysis program for astrophysical simulations.

9.4 Publication-quality graphics

The plan is to continue to improve matplotlib and use it as the basis for publication-quality graphics.

9.5 Easy-to-construct widgets

9.6 Easy-to-construct web graphics

CHAPTER 10

Development Timeline

Placeholder

CHAPTER 11

Indices and tables

- genindex
- modindex
- search