SunPy - Python for Solar Physics

The SunPy Community,

http://sunpy.org

E-mail: sunpy@googlegroups.com

Stuart J Mumford[1], Steven Christe[2], David Pérez-Suárez[3], Jack Ireland[2,4], Albert Y Shih[2], Andrew R Inglis[2,5], Simon Liedtke[6], Russell J Hewett[7], Florian Mayer[8], Keith Hughitt[9], Nabil Freij[1], Tomas Meszaros[10], Samuel M Bennett[1], Michael Malocha[11], John Evans[12],Ankit Agrawal[13], Andrew J Leonard[14], Thomas P Robitaille[15], Benjamin Mampaey[16], Jose Iván Campos-Rozo[17], and Michael S Kirk[2]

[1]Solar Physics & Space Plasma Research Centre (SP^{2}RC), School of Mathematics and Statistics, The University of Sheffield, Hicks Building, Hounsfield Road, Sheffield, S3 7RH, UK
[2]NASA Goddard Space Flight Center, Greenbelt, MD, USA
[3]South African National Space Agency - Space Science, Hospital Street, 7200 Hermanus, Western Cape, South Africa
[4]ADNET Systems Inc., Mail Code 671.1, NASA Goddard Space Flight Center, Greenbelt, MD, USA
[5]The Catholic University of America, Washington, DC, USA
[6]University of Bremen, Bibliothekstraße 1, 28359 Bremen, Germany
[7]Department of Mathematics, Massachusetts Institute of Technology, 77 Massachusetts Ave, E17-317, Cambridge, MA, USA
[8]Vienna University of Technology, Karlsplatz 13, 1040 Vienna, Austria
[9]Department of Cell Biology and Molecular Genetics, University of Maryland, College Park, MD, USA
[10]Masaryk University, Faculty of Informatics, Botanicka 68a Brno, Czech Republic
[11]Humboldt State University, 1 Harpst St, Arcata, CA, USA
[12]Boston Python User Group, Boston, MA, USA
[13]Indian Institute of Technology, Bombay, India
[14]Department of Mathematics and Physics, Aberystwyth University, Physical Sciences Building, Aberystwyth, SY23 3BZ, UK
[15]Max-Planck-Institut für Astronomie, Königstuhl 17, Heidelberg 69117, Germany
[16]Royal Observatory of Belgium, Brussels, Belgium
[17]Observatorio Astronómico Nacional, Universidad Nacional de Colombia, Bogotá, D.C., Colombia

Abstract. This paper presents version 0.4 of SunPy, a community-developed Python package for solar physics. Python, a free, cross-platform, general-purpose, high-level programming language, has seen widespread adoption among the scientific community, resulting in the availability of a large number of software packages,

from numerical computation (`NumPy`, `SciPy`) and machine learning (`scikit-learn`) to visualisation and plotting (`matplotlib`). SunPy is a data-analysis environment specialising in providing the software necessary to analyse solar and heliospheric datasets in Python. SunPy is open-source software (BSD licence) and has an open and transparent development workflow that anyone can contribute to. SunPy provides access to solar data through integration with the Virtual Solar Observatory (VSO), the Heliophysics Event Knowledgebase (HEK), and the HELiophysics Integrated Observatory (HELIO) webservices. It currently supports image data from major solar missions (e.g., *SDO*, *SOHO*, *STEREO*, and *IRIS*), time-series data from missions such as *GOES*, *SDO*/EVE, and *PROBA2*/LYRA, and radio spectra from e-Callisto and *STEREO*/SWAVES. We describe SunPy's functionality, provide examples of solar data analysis in SunPy, and show how Python-based solar data-analysis can leverage the many existing tools already available in Python. We discuss the future goals of the project and encourage interested users to become involved in the planning and development of SunPy.

1. Introduction

Science is driven by the analysis of data of an ever-growing variety and complexity. Modern advances in sensor technology, combined with the availability of inexpensive storage, has led to rapid increases in the amount of data available to scientists in almost every discipline. Solar physics is no exception to this trend. For example, NASA's *Solar Dynamics Observatory* (*SDO*) spacecraft, launched in February 2010, produces over 1 TB of data per day (Lemen et al., 2012). Managing and analysing this mountain of data requires increasingly sophisticated software tools. These tools should be robust, easy to use and modify, have a transparent development history, and conform to modern software-engineering standards. Software with these qualities provide a strong foundation that can support the needs of the community as data volumes grow and science questions evolve.

The SunPy project aims to provide a software package with these qualities for the analysis and visualisation of solar data. SunPy makes use of Python and scientific Python packages. Python is a free, general-purpose, powerful, and easy-to-learn high-level programming language. Additionally, Python is widely used outside of scientific fields in areas like 'big data' analytics, web development, and educational environments. For example, `pandas` was originally developed for quantitative analysis of financial data and has since grown into a generalised time-series data-analysis package. Python continues to see increased use in the astronomy community (Greenfield, 2011), which has similar goals and requirements as the solar physics community. Finally, Python integrates well with many technologies such as web servers (Dolgert et al., 2008) and databases.

The development of a package such as SunPy is made possible by the rich ecosystem of scientific packages available in Python. Core packages such as `NumPy`, `SciPy`, and `matplotlib` provide the basic functionality expected of a scientific programming language, such as array manipulation, core numerical algorithms, and visualisation, respectively. Building upon these foundations, packages such as `astropy` (astronomy), `pandas` (time-series), and `scikit-image` (image processing) provide more domain-specific functionality.

SunPy is designed to be a clean, simple-to-use, and well-structured open-source package that provides the *core* tools for solar data analysis, motivated by the need for a free and modern alternative to the existing SolarSoft (SSW) library. While SSW is open source and freely available, it relies on IDL (Interactive Data Language), a proprietary data-analysis environment.

The purpose of this paper is to provide an overview of SunPy's current capabilities, an overview of the project's development model, community aspects of the project, and future plans. The latest release of SunPy, version 0.4, can be downloaded from `http://sunpy.org` or can be installed using the Python package index (`http://pypi.python.org/pypi`).

2. Core Data Types

The core of SunPy is a set of data structures that are specifically designed for the three primary varieties of solar physics data: images, time series, and spectra. These core data types are supported by the SunPy classes: `Map` (2D spatial data), `LightCurve` (1D temporal series), and `Spectrum` and `Spectrogram` (1D and 2D spectra).

These classes allow access to the data and associated metadata and provide appropriate convenience functions to enable analysis and visualisation. For each of these classes, the data is stored in the `data` attribute, while the metadata should be stored in the `meta` attribute (currently, only `Map` has this feature fully implemented). It is possible to instantiate the data types from various different sources: e.g., files, URLs, and arrays. In order to provide instrument-specific specialisation, the core SunPy classes make use of subclassing; e.g., `Map` has an `AIAMap` sub-type for data from the *SDO*/AIA instrument.

All of the core SunPy data types include visualisation methods that are tailored to each data type. These visualisation methods all currently utilise the `matplotlib` package and are designed in such a way that they integrate well with the `pyplot` functional interface of `matplotlib`.

This design philosophy makes the behaviour of SunPy's visualisation routines intuitive to those who already understand the `matplotlib` interface, as well as allowing the use of the standard `matplotlib` commands to manipulate the plot parameters (e.g., title, axes). Data visualisation is provided by two functions: `peek()`, for quick plotting, and `plot()`, for plotting with more fine-grained control.

This section will give a brief overview of the *current* functionality of each of the core SunPy data types.

2.1. Map

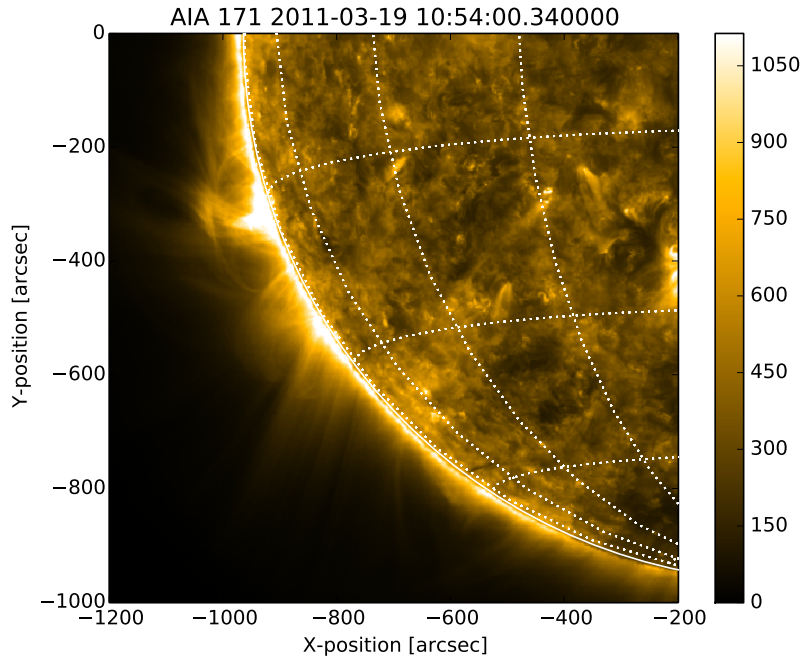
The map data type stores 2D spatial data, such as images of the Sun and inner heliosphere. It provides: a wrapper around a `numpy` data array, the images associated spatial coordinates, and other metadata. The `Map` class provides methods for typical operations on 2D data, such as rotation and re-sampling, as well as visualisation functionality. The `Map` class also provides a convenient interface for loading data from a variety of sources, including a FITS file as shown in Listing 1.

The architecture of the map subpackage consists of a template map called `GenericMap`, which is a subclass of `astropy.nddata.NDData`. `NDData` is a generic wrapper around a `numpy.ndarray` with a `meta` attribute to store metadata. As `NDData` is currently still in development, `GenericMap` does not yet make full use of its capabilities, but this inheritance structure provides for future integration with `astropy`. In order to provide instrument- or detector-specific integration, `GenericMap` is designed to be subclassed. Each subclass of `GenericMap` can register with the `Map` creation factory, which will then automatically return an instance of the specific `GenericMap` subclass dependent upon the data provided. SunPy v0.4 has `GenericMap`

specialisations for the following instruments: *Yohkoh*/SXT, *SOHO*/EIT and LASCO, *RHESSI*, *STEREO*/EUVI and COR, *Hinode*/XRT, *PROBA2*/SWAP, *SDO*/AIA and HMI, and *IRIS* SJI frames.

The `Map` class stores all of the metadata retrieved from the header of the image file in the `meta` attribute and provides convenience properties for commonly accessed metadata: e.g., `instrument`, `wavelength` or `coordinate_system`. Listing 1 demonstrates the quick-look functionality of `Map`.

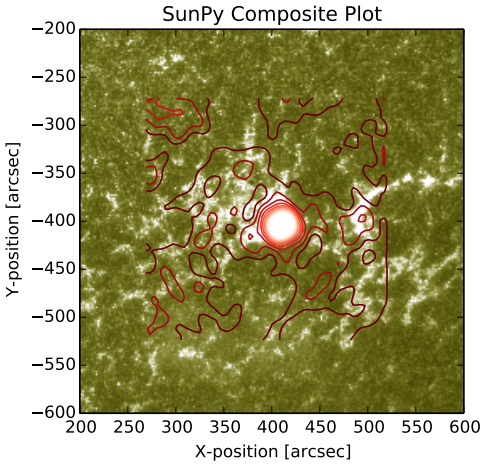
```
>>> import sunpy.map
>>> aiamap = sunpy.map.Map('aia_file.fits')
>>> smap = aiamap.submap([-1200, -200], [-1000, -0])
>>> smap.peek(draw_grid=True)
```



Listing 1: Example of the `AIAMap` specialisation of `GenericMap`. The map is created from an *SDO*/AIA FITS file, a cutout of the map is created, and then a quick-view plot is created with lines of heliographic longitude and latitude over-plotted.

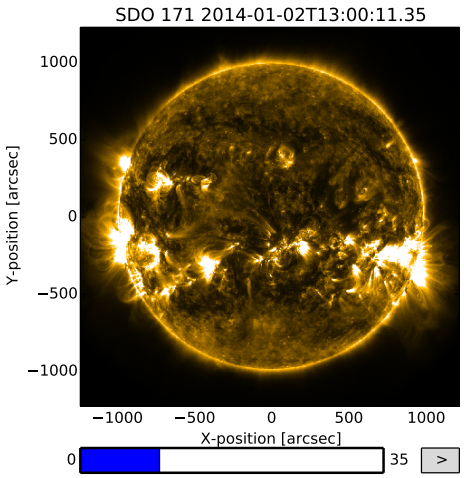
In addition to the data-type classes, the `map` subpackage provides two collection classes, `CompositeMap` and `MapCube`, for temporally and spatially aligned data respectively. `CompositeMap` provides methods for overlaying spatially aligned data, with support for visualisation of images and contour lines overlaid upon each other. `MapCube` provides methods for animation of its series of `Map` objects. Listings 2 and 3 show how to interact with these classes.

```
>>> import sunpy.map
>>> import matplotlib.pyplot as plt
>>> compmap = sunpy.map.Map('aia_1600_image.fits', 'RHESSI_image.fits',
...                         composite=True)
>>> compmap.set_levels(1, range(0, 50, 5), percent=True)
>>> compmap.set_colors(1, 'Reds_r')
#Plot the result and crop
>>> ax = plt.subplot()
>>> compmap.plot()
>>> ax.axis([200, 600, -600, -200])
>>> plt.show()
```



Listing 2: Example showing the functionality of `CompositeMap`, with RHESSI data composited on top of *SDO*/AIA data. The `CompositeMap` is plotted using the integration with the `matplotlib.pyplot` interface.

```
>>> import sunpy.map
>>> cubemap = sunpy.map.Map('aia_lev1_171a_2014_01*fits', cube=True)
>>> cubemap.peek()
```



Listing 3: Example showing the creation of a `MapCube` from a list of files. The resultant plot makes use of `matplotlib`'s interactive widgets to allow scrolling through the `MapCube`.

2.2. Lightcurve

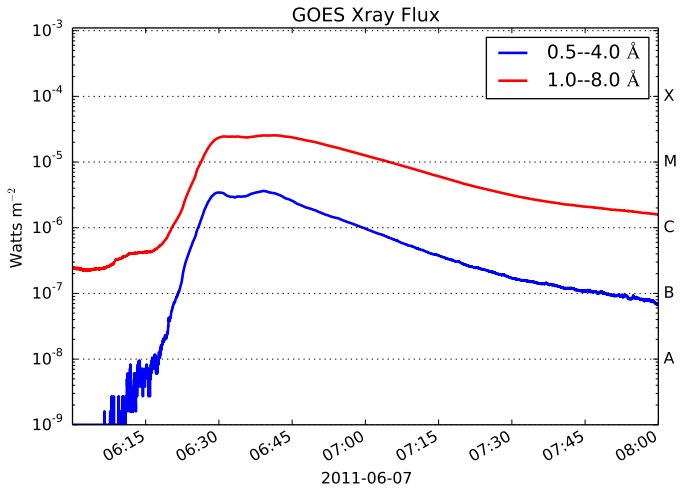
Time series data and their analyses are a fundamental part of solar physics for which many data sources are available. SunPy provides a `LightCurve` class with a convenient and consistent interface for handling solar time-series data. The main engine behind the `LightCurve` class is the `pandas` data-analysis library. `LightCurve`'s `data` attribute is a `pandas.DataFrame` object. The `pandas` library contains a large amount of functionality for manipulating and analysing time-series data, making it an ideal basis for `LightCurve` (McKinney, 2012). `LightCurve` assumes that the input data are time-ordered list(s) of numbers, and each list becomes a column in the `pandas` DataFrame object.

Currently, the `LightCurve` class is compatible with the following data sources: the *GOES* X-ray Sensor (XRS), *PROBA2*/LYRA, and *SDO*/EVE†. For each of these instruments, a subclass of the `LightCurve` object is initialised (e.g., `GOESLightCurve`) which inherits from `LightCurve`, but allows instrument-specific functionality to be included. Future developments will introduce support for additional instruments and data products, as well as implementing a factory interface similar to that of `Map`. Since there is no established standard as to how time-series data should be stored and distributed, each SunPy `LightCurve` object subclass provides the ability to download its corresponding specific data format in its constructor and parse that file type.

† Note that only the level "OCS" and average CSV files is currently implemented – see `http://lasp.colorado.edu/home/eve/data/`

A `LightCurve` object may be created using a number of different methods. For example, a `LightCurve` may be created for a specific instrument given an input time range. In Listing 4, the `LightCurve` constructor searches a remote source for the GOES X-ray data specified by the time interval, downloads the required files, and subsequently creates and plots the object. Alternatively, if the data file already exists on the local system, the `LightCurve` object may be initialised using that file as input.

```
>>> from sunpy import lightcurve
>>> from sunpy.time import TimeRange
>>> goes = lightcurve.GOESLightCurve.create('2011-06-07 06:00',
...                                         '2011-06-07 08:00')
>>> goes.peek()
>>> print goes.data['B_FLUX'].max()
2.55542e-05
>>> print goes.data['B_FLUX'].idxmax()
2011-06-07 06:41:26
```



Listing 4: Example retrieval of a GOES lightcurve using a time range and the output of the `peek()` method. The maximum flux value in the *GOES* 1.0–8.0Å channel is then retrieved along with the location in time of the maximum.

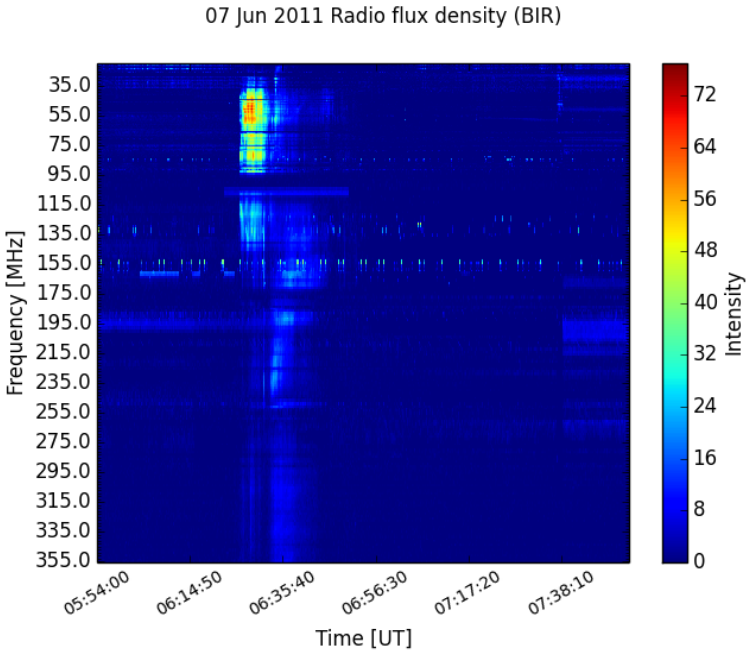
2.3. Spectra

SunPy aims to provide broad support for solar spectroscopy instruments. The variety and complexity of these instruments and their resultant datasets makes this a challenging goal. The `spectra` module implements a `Spectrum` class for 1D data (intensity as a function of frequency) and a `Spectrogram` class for 2D data (intensity as a function of time and frequency). Each of these classes uses a `numpy.ndarray` object as its `data` attribute.

As with other SunPy data types, the `Spectrogram` class has been built so that each instrument initialises using a subclass containing the instrument-specific functionalities. The common functionality provided by the base `Spectrogram` class includes joining different time ranges and frequencies, performing frequency-dependent background subtraction, and convenient visualization and sampling of the data. Currently, the `Spectrogram` class supports radio spectrograms from the e-Callisto solar radio spectrometer network and *STEREO*/SWAVES spectrograms.

Listing 5 shows how the `CallistoSpectrogram` object retrieves spectrogram data in the time range specified taken at the observatory of interest. When the data is requested using the `from_range()` function, the object merges all the downloaded files into a single spectrogram, across time and frequency. In the example shown, data is provided in two frequency ranges: 20–90 MHz and 55–355 MHz. Since the data are not evenly spaced in the frequency range, the `Spectrogram` object linearises the frequency axis to assist analysis. The example also demonstrates the implemented background subtraction method, which calculates a constant background over time for each frequency channel.

```
>>> from sunpy.spectra.sources.callisto import CallistoSpectrogram
>>> tstart, tend = "2011-06-07T06:00:00", "2011-06-07T07:45:00"
>>> callisto = CallistoSpectrogram.from_range("BIR", tstart, tend)
>>> callisto_nobg = callisto.subtract_bg()
>>> callisto_nobg.peek(vmin=0)
```



Listing 5: Example of how `CallistoSpectrogram` retrieves the data for the requested time range and observatory, merges it, and removes the background signal. The data requested – 'BIR' – is the code name of the Rosse Observatory at Birr Castle in Ireland.

3. Solar Data Search and Retrieval

Several well-developed resources currently exist which provide remote access to and data retrieval form a large number of solar and heliospheric data sources and event databases. SunPy provides support for these resources via the `net` subpackage. In the following subsections, we describe each of these resources and how to use them.

3.1. VSO

The Virtual Solar Observatory (VSO) provides a single, standard query interface to solar data from different archives around the world (Hill et al., 2009). Data products can be requested for specific instruments or missions and can also be requested based on physical parameters of the data product such as the wavelength range. In addition to the VSO's primary web-based interface, a SOAP (Simple Object Access Protocol) service is also available. SunPy's `vso` module provides access to the VSO via this SOAP service using the `suds` package.

Listing 6 shows an example of how to query the VSO using the `vso` module. Queries are constructed using one or more attribute objects. Each attribute object is a constraint on a parameter of the data set, such as the time, instrument, or wavelength. Listing 6 also shows how to download the data using the constructed query. One can use the metadata associated with the data sets to specify how the files will be named locally, such as creating subdirectories based on the instrument names.

Listing 7 shows an example of how to make an advanced query by combining attribute objects. Two attribute objects can be combined with a logical `or` operation using the `|` (pipe) operator. All attribute objects provided to the query as arguments are combined with a logical `and` operation.

```
>>> from sunpy.net import vso
>>> client = vso.VSOClient()
>>> tstart, tend = '2011/6/7 05:30', '2011/6/7 10:30'
>>> lasco_query = client.query(vso.attrs.Time(tstart, tend),
...                            vso.attrs.Instrument('lasco'))
>>> lasco_query.num_records()
40
>>> lasco_query.show()
Start time           End time             Source  Instrument  Type
----------           --------             ------  ----------  ----
2011-06-07 05:35:23  2011-06-07 05:35:48  SOHO    LASCO       CORONA
2011-06-07 05:43:09  2011-06-07 05:43:29  SOHO    LASCO       CORONA
...
>>> pathformat = '/data/{instrument}/{detector}/{file}.fits'
>>> results = client.get(coronagraphs, path = pathformat)
```

Listing 6: Example of querying a single instrument over a time range and downloading the data

```
>>> condition = (vso.attrs.Detector('cor1') |
...              vso.attrs.Wave(125, 135) |
...              vso.attrs.Wave(165, 175) ) # in angstroms
>>> advanced = client.query(vso.attrs.Time(tstart, tend), condition)
>>> advanced.num_records()
4434
>>> advanced.show()
Start time           End time             Source    Instrument  Type
----------           --------             ------    ----------  ----
2011-06-07 00:00:00  2011-06-08 00:00:00  SDO       EVE         FULLDISK
...
2011-06-07 05:31:09  2011-06-07 05:31:19  PROBA2    SWAP        FULLSUN
...
2011-06-07 10:25:43  2011-06-07 10:25:45  STEREO_B  SECCHI      CORONA
2011-06-07 10:30:00  2011-06-07 10:30:01  STEREO_A  SECCHI      CORONA
...
2011-06-07 10:30:00  2011-06-07 10:30:01  SDO       AIA         FULLDISK
```

Listing 7: Example of an advanced VSO query using attribute objects, combining both data from a detector and any data that falls within two wavelength ranges, continuing from Listing 6.

3.2. HEK

The Sun is an active star and exhibits a wide range of transient phenomena (e.g., flares, radio bursts) on many different time-scales, length-scales, and wavelengths. Observations and metadata concerning these phenomena are collected in the Heliophysics Event Knowledgebase (HEK) (Hurlburt et al., 2012). Entries are generated both by automated algorithms and human observers. Some of the information in the HEK reproduces feature and event data from elsewhere (for example, the *GOES* flare catalogue), and some is generated by the Solar Dynamics Observatory Feature Finding Team (Martens et al., 2012). The advantage of the HEK is it provides an homogeneous and well-described interface to a large amount of feature and event information of interest to the solar physics community. SunPy accesses this information through the `hek` module. The `hek` module makes use of the HEK API†.

Simple HEK queries consist of start time, an end time, and an event type (Listing 8). Event types are specified as upper case, two letter strings, and these strings are identical to the two letter abbreviations found at the HEK website (`http://www.lmsal.com/hek/VOEvent_Spec.html`).

```
>>> from sunpy.net import hek
>>> client = hek.HEKClient()
>>> tstart, tend = '2011/08/09 00:00:00', '2011/08/10 00:00:00'
>>> result = client.query(hek.attrs.Time(tstart, tend),
...                       hek.attrs.EventType('FL')) # flares
>>> len(result)
52
```

Listing 8: Example usage of the `hek` module showing a simple HEK search for solar flares on 2011 August 9.

The module `hek.attrs` contains attributes of the HEK that can be used to construct HEK queries. For example, a flare is an attribute of the HEK, and so `hek.attrs.EventType('FL')` in Listing 8 can be alternatively replaced with `hek.attrs.FL`.

HEK attributes differ from VSO attributes (Section 3.1) in that many of them are wrappers that conveniently expose comparisons by overloading Python operators. This allows filtering of the HEK entries by the properties of the event. As was mentioned above, the HEK stores feature/event metadata obtained in different ways, known generally as feature recognition methods (FRMs). Example in Listing 9 filters the results of the previous result to return only those events that have the FRM 'SSW Latest Events'. Multiple comparisons can be made by including more comma-separated conditions on the attributes in the call to the HEK query method.

† For more information see `http://vso.stanford.edu/hekwiki/ApplicationProgrammingInterface`

```
>>> result = client.query(hek.attrs.Time(tstart, tend),
...                       hek.attrs.EventType('FL'),
...                       hek.attrs.FRM.Name=='SSW Latest Events')
>>> len(result)
9
```

Listing 9: An HEK query that returns only those flares that were detected by the 'SSW Latest Events' feature recognition method.

HEK comparisons can be combined using Python's logical operators (e.g., `and` and `or`). This makes complex queries easy to create: Listing 10 returns flares west of 50 arcseconds or those that have a peak flux above 1000.0 (the units of the flux are FRM-dependent and are described at the HEK website).

```
>>> result = client.query(hek.attrs.Time(tstart, tend),
...                       hek.attrs.EventType('FL'),
...                       (hek.attrs.Event.Coord1>50)
...                       or (hek.attrs.FL.PeakFlux>1000.0))
```

Listing 10: HEK query using the logical `or` operator.

All FRMs report the required feature attributes, but the optional attributes are FRM dependent. If a FRM does not have one of the optional attributes, `None` is returned by the `hek` module.

The ability to use comparison and logical operators on HEK attributes allows the construction of queries of almost arbitrary complexity. The results of a HEK query can be used to download the corresponding data from the VSO using SunPy's `H2VClient`, as demonstrated in Listing 11.

```
>>> from sunpy.net import hek2vso
>>> h2v = hek2vso.H2VClient()
>>> vso_results = h2v.translate_and_query(result[0])
>>> h2v.vso_client.get(vso_results[0]).wait()
```

Listing 11: Code snippet continuing from Listing 10 showing the query and download of data from the first HEK result from the VSO.

3.3. HELIO

The HELiophysics Integrated Observatory (HELIO)† has compiled a list of web services which allows scientists to query and discover data throughout the heliosphere, from solar and magnetospheric data to planetary and inter-planetary data (Pérez-Suárez et al., 2012). HELIO is built with a Service-Oriented Architecture, i.e., its capabilities

† For more information see `http://helio-vo.eu`

are divided into a number of tasks that are implemented as separate services. HELIO is made up of nine different public services, which allows scientists to search different catalogues of registered events, solar features, data from instruments in the heliosphere, and other information such as planetary or spacecraft position in time. Additionally, HELIO provides a service that uses a propagation model to link the data in different points of the solar system by its original nature (e.g., Earth auroras are a signature of magnetic field disturbances produced a few days before on the Sun). In addition to the primary, web-based interface to HELIO, its services are available via an API.

SunPy's `hec` module provides an interface to the HELIO Event Catalogue (HEC) service. This module was developed as part of a Google Summer of Code (GSOC) project in 2013. The HEC service currently provides access to 84 catalogues from different sources. As with all of the HELIO services, the HEC service provides results in VOTable data format (defined by IVOA Ochsenbein et al. (2011)). The `hec` module parses this output using the `astropy.io.votable` package. This format has the advantage of containing metadata with information like data provenance and the performed query.

For example, Listing 12 shows how to obtain information from different catalogues of coronal mass ejections (CMEs).

```
>>> from sunpy.net.helio import hec
>>> hc = hec.Client()
>>> tstart, tend = '2011-06-07T06:00:00', '2011-06-07T12:00:00'
>>> event_type = 'cme'

# From all the catalogues these which name contain our event of interest
>>> catalogues = hc.get_table_names()
>>> catalogues_event = [l[0] for l in catalogues
...                     if event_type in l[0] and 'list' not in l[0]]

# Query all the catalogues that comes from cactus
>>> results = [hc.time_query(tstart, tend, event)
...            for event in catalogues_event if 'cactus' in event]
>>> for cat in results:
...     print "{cat} has {nres} results".format(cat = cat.ID, \
...                                             nres = len(cat.array))
__helio_hec-cactus_stereoa_cme has 4 results
__helio_hec-cactus_stereob_cme has 3 results
__helio_hec-cactus_soho_cme has 7 results
```

Listing 12: Example of querying the HEC service to multiple CME catalogues, in this case the ones detected automatically by CACTus (Robbrecht et al., 2009).

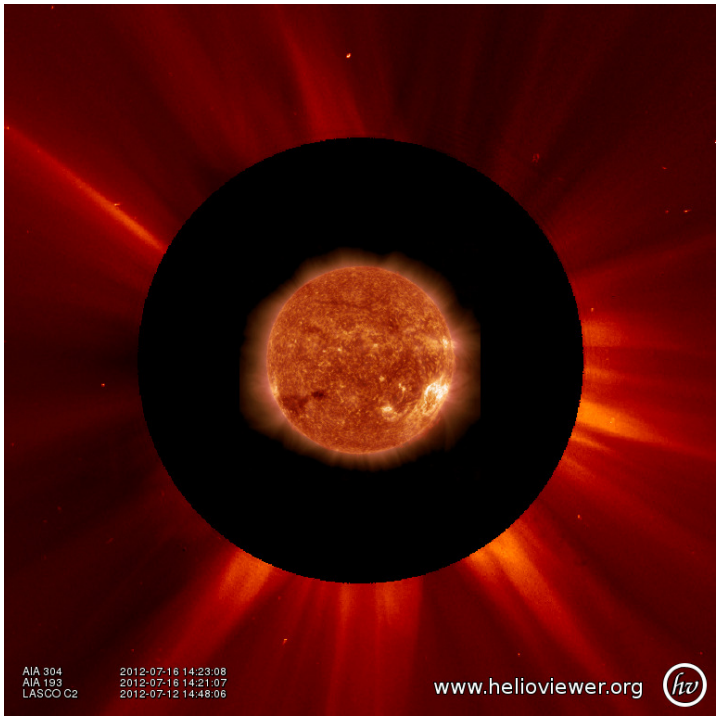
3.4. Helioviewer

SunPy provides the ability to download images hosted by the Helioviewer Project (http://helioviewer.org). The aim of the Helioviewer Project is to enable the exploration of solar and heliospheric data from multiple data sources (such as instrumentation and feature/event catalogues) via easy-to-use visual interfaces. The Helioviewer Project have developed two client applications that allow users to browse images and create movies of the Sun taken by a variety of instruments: Helioviewer.org, a Google Maps-like web application, and JHelioviewer, a movie streaming desktop application. In order to manage all of this data, the Helioviewer project maintains archives of all of its data in JPEG2000 format (for further details on the Helioviewer Project see Muller et al. 2009). The JPEG2000 files are typically highly compressed compared to the source FITS files from which they are generated, but are still high-fidelity, and thus can be used to quickly visualise large amounts of data from multiple sources. SunPy is also used in Helioviewer production servers to manage the download and ingestion of JPEG2000 files from remote servers.

The Helioviewer Project categorises image data based on the physical construction of the source instrument, using a simple hierarchy: observatory → instrument → detector → measurement, where '→' means 'provides (possibly multiple)'. Each Helioviewer Project JPEG2000 file contains metadata which are based on the original FITS header information, and carry sufficient information to permit overlay with other Helioviewer JPEG2000 files. Images can be accessed either as PNGs (Section 3.4.1) or as JPEG2000 files (Section 3.4.2).

3.4.1. Download a PNG file The Helioviewer API allows composition and overlay of images from multiple sources, based on the positioning metadata in the source FITS file. SunPy accesses this overlay/composition capability through the `download_png()` method of the Helioviewer client. Listing 13 gives an example of the composition of three separate image layers into a single image.

```
>>> from sunpy.net.helioviewer import HelioviewerClient
>>> hv = HelioviewerClient()
>>> hv.download_png('2099/01/01', 6,
...                 "[SDO,AIA,AIA,304,1,100],[SDO,AIA,AIA,193,1,50],"+
...                 "[SOHO,LASCO,C2,white-light,1,100]",
...                 x0=0, y0=0, width=768, height=768)
```



Listing 13: Acquisition of a PNG image composed from data from three separate sources.

The first argument is the requested time of the image, and Helioviewer selects images closest to the requested time. In this case, the requested time is in the future and so Helioviewer will find the most recent available images from each source. The second argument refers to the image resolution in arcseconds per pixel (larger values mean lower resolution). The third argument is a comma-delimited string of the three requested image layers, the details of which are enclosed in parentheses. The image layers are described using the observatory $\rightarrow$ instrument $\rightarrow$ detector $\rightarrow$ measurement combination described above, along with two following numbers that denote the visibility and the opacity of the image layer, respectively (1/0 is visible/invisible, and opacity is in the range $0 \rightarrow 100$, with 100 meaning fully opaque). The quantities `x0` and `y0` are the x and y centre points about which to centre the image (measured in helio-projective cartesian coordinates), and the `width` and `height` are the pixel values for the image dimensions.

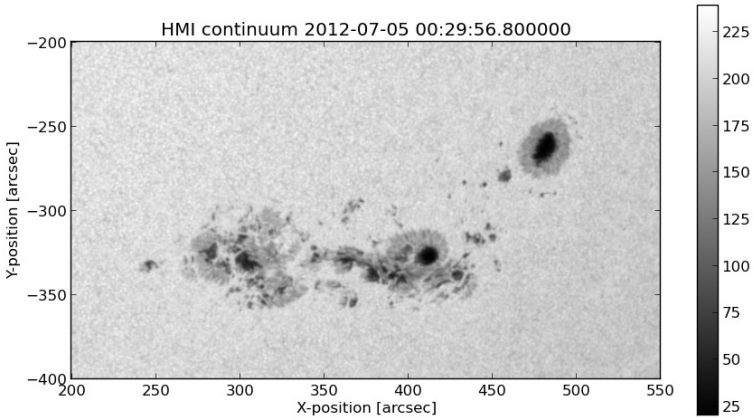
This functionality makes it simple for SunPy users to generate complex images from multiple, correctly overlaid, image data sources.

3.4.2. Download a JPEG2000 file As noted above, Helioviewer JPEG2000 files contain metadata that allow positioning of the image data. There is sufficient metadata in each file to permit the creation of a SunPy `Map` object (see Section 2.1) from a Helioviewer JPEG2000 file. This allows image data to be manipulated in the same way as any other map object.

Reading JPEG2000 file into a SunPy session requires installing two other pieces of software. The first, `OpenJPEG`, is an open-source library for reading and writing JPEG2000 files. The other package required is `glymur`, an interface between Python and the OpenJPEG libraries (note that these packages are *not* required to use the functionality described in Section 3.4.1).

Listing 14 demonstrates the querying, downloading, reading and conversion of a Helioviewer JPEG2000 file into a SunPy map object. This functionality allows users to visualise and manipulate Helioviewer-supplied image data in an identical fashion to a SunPy `Map` object generated from FITS data (see Section 2.1).

```
>>> import sunpy.map
>>> filepath = hv.download_jp2('2012/07/05 00:30:00',
...                            observatory='SDO',
...                            instrument='HMI', detector='HMI',
...                            measurement='continuum')
>>> sunpy.map.Map(filepath).submap([200, 550], [-400, -200]).peek()
```



Listing 14: Acquisition and display of a Helioviewer JPEG2000 file as a SunPy `Map` object. Images values are byte-scaled in the range 0–255.

3.5. The File Database

Easy access to large quantities of solar data frequently leads to data files accumulating in local storage such as laptops and desktop computers. Keeping data organised and available is typically a cumbersome task for the average user. The file database is a

subpackage of SunPy that addresses this problem by providing a unified database to store and manage information about local data files.

The `database` subpackage makes use of a database supported by `SQLAlchemy` (http://www.sqlalchemy.org). This library was chosen since it supports many SQL dialects. If SQLite is selected, the database is stored as a single file, which is created automatically. A server-based database, on the other hand, could be used by collaborators who work together on the same data from different computers: a central database server stores all data and the clients connect to it to read or write data.

The database can store and manage all data that can be read via SunPy's `io` subpackage, and direct integration with the `vso` module is supported. It is also possible to manually add file or directory entries. The package also provides a unified data search via the `fetch()` method, which includes both local files and files on the VSO. This reduces the likelihood of downloading the same file multiple times. When a file is added to the database, the file is scanned for metadata, and a file hash is produced. The current date is associated with the entry along with metadata summaries such as instrument, date of observation, field of view, etc. The database also provides the ability to associate custom metadata to each database entry such as keywords, comments, and favourite tags, as well as querying the full metadata (e.g., FITS header) of each entry.

The `Database` class connects to a database and allows the user to perform operations on it. Listing 15 shows how to connect to an in-memory database and download data from the VSO. These entries are automatically added to the database. The function `len()` is used to get the number of records. The function `display_entries()` displays an iterable of database entries in a formatted ASCII table. The headlines correspond to the attributes of the respective database entries.

A useful feature of the database package is the support of `undo` and `redo` operations. This is particularly convenient in interactive sessions to easily revert accidental operations. This feature will also be desirable for a planned GUI frontend for this package.

```
>>> from sunpy.net import vso
>>> from sunpy.database import Database
>>> database = Database('sqlite:///')
>>> database.download(
...     vso.attrs.Time('2012-08-05', '2012-08-05 00:00:05'),
...     vso.attrs.Instrument('AIA'))
>>> len(database)
2
>>> from sunpy.database.tables import display_entries
>>> print display_entries(
...     database,
...     ['id', 'observation_time_start', 'wavemin', 'wavemax'])
id observation_time_start wavemin wavemax
-- --------------------- ------- -------
1  2012-08-05 00:00:01    9.4     9.4
2  2012-08-05 00:00:02    33.5    33.5
```

Listing 15: Example usage of the `database` subpackage.

4. Additional functionality

SunPy is meant to provide a consistent environment for solar data analysis. In order to achieve this goal SunPy provides a number of additional functions and packages which are used by the other SunPy modules and are made available to the user. This section briefly describes some of these functions.

4.1. World Coordinate System (WCS) Coordinates

Coordinate transformations are frequently a necessary task within the solar data analysis workflow. An often used transformation is from observer coordinates (e.g., sky coordinates) to a coordinate system that is mapped onto the solar surface (e.g., latitude and longitude). This transformation is necessary to compare the true physical distance between different solar features. This type of transformation is not unique to solar observations, but is not often considered by astronomical packages such as the Astropy `coordinates` package. The `wcs` package in SunPy implements the World Coordinate System (WCS) for solar coordinates as described by Thompson (2006). The transformations currently implemented are some of the most commonly used in solar data analysis, namely converting from Helioprojective-Cartesian (HPC) to Heliographic (HG) coordinates. HPC describes the positions on the Sun as angles measured from the center of the solar disk (usually in arcseconds) using Cartesian coordinates (X, Y). This is the coordinate system most often defined in solar imaging data (see for example, images from *SDO*/AIA, *SOHO*/EIT, and *TRACE*). HG coordinates express positions on the Sun using longitude and latitude on the solar sphere. There are two standards for this

coordinate system: Stonyhurst-Heliographic, where the origin is at the intersection of the solar equator and the central meridian as seen from Earth, and Carrington-Heliographic, which is fixed to the Sun and does not depend on Earth. The implementation of these transformations pass through a common coordinate system called Heliocentric-Cartesian (HCC), where positions are expressed in true (de-projected) physical distances instead of angles on the celestial sphere. These transformations require some knowledge of the location of the observer, which is usually provided by the image header. In the cases where it is not provided, the observer is assumed to be at Earth. Listing 16 shows some examples of coordinate transforms carried out in SunPy using the `wcs` utilities.

```
>>> from sunpy import wcs
>>> wcs.convert_hg_hpc(10, 53)
(100.49244115330731, 767.97438321917502)
# Convert that position back to heliographic coordinates
>>> wcs.convert_hpc_hg(100.49, 767.97)
(9.9996521808465175, 52.999563684874893)
# Try to convert a position which is not on the Sun to HG
>>> wcs.convert_hpc_hg(-1500, 0)
sunpy/wcs/wcs.py:180: RuntimeWarning: invalid value encountered in sqrt
  distance = q - np.sqrt(distance)
(nan, nan)
# Convert sky coordinate to a position in HCC
>>> wcs.convert_hpc_hcc(-300, 400, z=True)
(-216716967.63331246, 288956420.9477042, 594364636.2208252)
```

Listing 16: Using the `wcs` subpackage.

4.2. Sun

The purpose of the `sun` subpackage is to provide solar-specific data such as ephemerides and solar constants. The main namespace contains a number of functions that provide solar ephemerides such as the Sun-to-Earth distance, solar-cycle number, the mean anomaly, etc. All of these functions take a time as their input, which can be provided in a format compatible with `sunpy.time.parse_time()`.

The `sun.constants` module provides a number of solar-related constants in order to provide consistency in the calculations of derived solar values within the SunPy code base, but also to the user. Every solar constant is provided as a `Constant` object as defined by Astropy. Each `Constant` object defines a `Quantity`, a number associated with a unit, along with the constant's provenance (i.e., reference) and its uncertainty. Using Astropy's `Quantity` objects, any solar constant can easily be converted between different units, including between the SI or cgs unit systems, as can be seen in Listing 17. As these objects inherit from NumPy's `ndarray`, they work well with standard representations of numbers. For convenience, a number of shortcuts to frequently used constants are

provided directly when importing the module. A larger list of constants can be accessed through an interface modelled on that provided by the SciPy constants package and is available as a dictionary called `physical_constants`. To view them all quickly, a `print_all()` function is available.

```
>>> from sunpy.sun import constants
>>> print(constants.mass)
  Name   = Solar mass
  Value  = 1.9891e+30
  Error  = 5e+25
  Units  = kg
  Reference = Allen's Astrophysical Quantities 4th Ed.
# Verify the average density of the Sun and convert to cgs
>>> (constants.mass/constants.volume).cgs
<Quantity 1.40851154227 g / (cm3)>
# Search for the age of the Sun
>>> constants.find('age')
['age', 'average angular size', 'average density', 'average intensity']
>>> constants.value('age'), constants.unit('age')
(4600000000.0, Unit("yr"))
```

Listing 17: Using the `sun.constants` module.

4.3. Instruments

In addition to providing support for instrument-specific solar data via the main data classes `Map`, `LightCurve`, and `Spectrum`, some instrument-specific functions may be found within the `instr` subpackage. These functions are generally those that are unique to one particular solar instrument, rather than of general use, such as a function to construct a *GOES* flare event list or a function to query the *LYRA* timeline annotation file. Currently, some support is included for the *GOES*, *LYRA*, *RHESSI* and *IRIS* instruments, while future developments will include support for additional missions. Ultimately, it is anticipated that solar missions requiring a large suite of software tools will each be supported via a separately maintained package that is affiliated with SunPy.

5. Development and Community

SunPy is a community-developed library, designed and developed for and by the solar physics community. Not only is all the source code publicly available online under the permissive 2-clause BSD licence, the whole development process is also online and open for anyone to contribute to. SunPy's development makes use of the online service GitHub (`http://github.com`) and Git† as its distributed version control software.

† For more information see `http://git-scm.com/`

The continued success of an open-source project depends on many factors; three of the most important are (1) utility and quality of the code, (2) documentation, and (3) an active community (Bangerth and Heister, 2013). Several tools, some specific to Python, are used by SunPy to make achieving these goals more accessible. To maintain high-quality code, a transparent and collaborative development workflow made possible by GitHub is used. The following conditions typically must be met before code is accepted.

(i) The code must follow the PEP 8 Python style guidelines to maintain consistency in the SunPy code.

(ii) All new features require documentation in the form of doc strings as well as user guides.

(iii) The code must contain unit tests to verify that the code is behaving as expected.

(iv) Community consensus is reached that the new code is valuable and appropriately implemented.

This kind of development model is widely used within the scientific Python community as well as by a wide variety of other projects, both open and closed source.

Additionally, SunPy makes use of 'continuous integration' provided by Travis CI (`http://travis-ci.org`), a process by which the addition of any new code automatically triggers a comprehensive review of the code functionality which are maintained as unit tests. If any single test fails, the community is alerted before the code is accepted. The unit-test coverage is monitored by a service called Coveralls (`http://coveralls.io`).

High-quality documentation is one of the most important factors determining the success of any software project. Powerful tools already exist in Python to support documentation, thanks to native Python's focus on its own documentation. SunPy makes use of the Sphinx (`http://sphinx-doc.org`) documentation generator. Sphinx uses reStructuredText as its markup language, which is an easy-to-read, what-you-see-is-what-you-get plaintext markup syntax. It supports many output formats most notably HTML, as well as PDF and ePub, and provides a rich, hierarchically structured view of in-code documentation strings. The SunPy documentation is built automatically and is hosted by Read-the-Docs (`http://readthedocs.org`) at `http://docs.sunpy.org`.

Communication is the key to maintaining an active community, and the SunPy community uses a number of different tools to facilitate communication. For immediate communications, an active IRC chat room (#SunPy) is hosted on freenode.net. For more involved or less immediate needs, such as developer comments or discussions, an open mailing list is hosted by Google Groups. Bug tracking, code reviews, and feature-request discussions take place directly on GitHub. The SunPy community also reaches out to the wider solar physics community through presentations, functionality demonstrations, and informal meetups at scientific meetings.

6. Future of SunPy

Over the three years of SunPy's development, the code base has grown to over 17,000 lines. SunPy is already a useful package for the analysis of calibrated solar data, and it continues to gain significant new capabilities with each successive release. The primary focus of the SunPy library is the analysis and visualisation of 'high-level' solar data. This means data that has been put through instrument processing and calibration routines, and contains full (WCS) coordinate information. The plan for SunPy is to continue development within this scope. The primary components of this plan are to provide a set of data types that are interchangeable with one another: e.g., if you slice a `MapCube` along one spatial location, a `LightCurve` of intensity along the time range of the `MapCube` should be returned. To achieve this goal, all the data types need to share a unified coordinate system architecture so that each data type is aware of what the physical type of its data is and how operations on that data should be performed. This will enable useful operations such as the coordinate and solar-rotation-aware overplotting of HELIO (Section 3.3) and HEK results (Section 3.2) onto maps (Section 2.1). Finally, support for new data providers and services will be integrated into SunPy. For example, new HELIO services will be supported by SunPy, aiming for seamless interaction between the other services and tools available (e.g., `hek`, `map`).

In concert with the work on the data types, further integration with the `astropy` package will enable SunPy to incorporate many new features with little effort. Collaboration and joint development with the Astropy project (The Astropy Collaboration et al., 2013) is ongoing.

7. Summary

We have presented the release of SunPy (v0.4), a Python package for solar physics. In this paper we have described the main functionality which includes the SunPy data types, `Map` (see Section 2.1), `Lightcurve` (see Section 2.2), and `Spectrogram` (see Section 2.3). We have described the data and event catalogue retrieval capabilities of SunPy for the Virtual Solar Observatory (see Section 3.1), the Heliophysics Event Knowledgebase (see Section 3.2), as well as the Heliophysics Integrated Observatory (see Section 3.3). We described a new organization tool for data files integrated into SunPy (see Section 3.5) and we discussed the community aspects, development model (see Section 5), and future plans (see Section 6) for the project. We invite members of the community to contribute to the effort by using SunPy for their research, reporting bugs, and sharing new functionality with the project.

8. Acknowledgements

Many of the larger features in SunPy have been developed with the generous support of external organizations. Initial development of SunPy's VSO and HEK implementations

were funded by ESA's Summer of Code In Space (SOCIS 2011, 2012, 2013) program, as well as a prototype GUI and an N-dimensional data-type implementation. In 2013, with support from Google's Summer Of Code (GSOC) program, through the Python Software Foundation, the `helio`, `hek2vso`, and `database` subpackages were developed. The Spectra and Spectrogram classes were implemented with support from the Astrophysics Research Group at Trinity College Dublin, Ireland, in 2012.

References

Wolfgang Bangerth and Timo Heister. What makes computational open source software libraries successful? *Computational Science & Discovery*, 6(1):015010, 2013. URL `http://stacks.iop.org/1749-4699/6/i=1/a=015010`.

Barry Boehm, Bradford Clark, Ellis Horowitz, Chris Westland, Ray Madachy, and Richard Selby. Cost models for future software life cycle processes: Cocomo 2.0. *Annals of Software Engineering*, 1(1):57–94, 1995. URL `http://dx.doi.org/10.1007/BF02249046`.

A Dolgert, L Gibbons, and V Kuznetsov. Rapid web development using AJAX and python. *Journal of Physics: Conference Series*, 119(4):042011, July 2008. URL `http://stacks.iop.org/1742-6596/119/i=4/a=042011?key=crossref.eda0671577dafc3c78be7e071da5a2fe`.

V. Domingo, B. Fleck, and A. I. Poland. The SOHO Mission: an Overview. Sol. Phys., 162:1–37, December 1995.

S. Freeland and R. Bentley. *SolarSoft*. November 2000. doi: 10.1888/0333750888/3390.

P Greenfield. What python can do for astronomy. In *Astronomical Data Analysis Software and Systems XX*, volume 442, page 425, 2011.

Frank Hill, et al. The virtual solar Observatory—A resource for international heliophysics research. *Earth, Moon, and Planets*, 104(1-4):315–330, April 2009. URL `http://solar.physics.montana.edu/martens/papers/Hill-VSO-Oct07.pdf`.

N. Hurlburt, et al. Heliophysics Event Knowledgebase for the Solar Dynamics Observatory (SDO) and Beyond. Sol. Phys., 275:67–78, January 2012.

J. R. Lemen, et al. The Atmospheric Imaging Assembly (AIA) on the Solar Dynamics Observatory (SDO). Sol. Phys., 275:17–40, January 2012.

P. C. H. Martens, et al. Computer Vision for the Solar Dynamics Observatory (SDO). Sol. Phys., 275:79–113, January 2012.

Wes McKinney. *Python for Data Analysis*. O'Reilly Media, Sebastopol, CA, 2012. ISBN 9781449323622 1449323626 9781449323615 1449323618 1449319793 9781449319793. URL `http://proquest.safaribooksonline.com/?fpi=9781449323592`.

Daniel Muller, et al. JHelioviewer: visualizing large sets of solar images using JPEG 2000. *Computing in Science & Engineering*, 11(5):38–47, September 2009. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5228714`.

F. Ochsenbein, et al. IVOA Recommendation: VOTable Format Definition Version 1.2. *ArXiv e-prints*, October 2011.

Fernando Perez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science & Engineering*, 9(3):21–29, 2007. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4160251`.

D. Pérez-Suárez, et al. Studying Sun-Planet Connections Using the Heliophysics Integrated Observatory (HELIO). Sol. Phys., 280:603–621, October 2012.

W. D. Pesnell, B. J. Thompson, and P. C. Chamberlin. The Solar Dynamics Observatory (SDO). Sol. Phys., 275:3–15, January 2012.

E. Robbrecht, D. Berghmans, and R. A. M. Van der Linden. Automated LASCO CME Catalog for Solar Cycle 23: Are CMEs Scale Invariant? ApJ, 691:1222–1234, February 2009.

The Astropy Collaboration, et al. Astropy: A community python package for astronomy. *Astronomy & Astrophysics*, 558:A33, September 2013. URL `http://www.aanda.org/10.1051/0004-6361/201322068`.

W. T. Thompson. Coordinate systems for solar image data. *Astronomy and Astrophysics*, 449(2):791–803, April 2006. URL `http://www.aanda.org/index.php?option=com_article&access=doi&doi=10.1051/0004-6361:20054262&Itemid=129`.