Working with data from the Solar Dynamics Observatory
11th January 2011, Version 1.2

Daniel Brown, Stephane Regnier, Mike Marsh, and Danielle Bewsher

University of Central Lancashire

Abstract

The aim of this document is to introduce the analysis of data from SDO using SSWIDL, specifically AIA and HMI observations. This primer will cover how to obtain data, how to read it in to SSWIDL, and some common analyses that the user may want to do.

This guide has grown out of material produced for an STFC Postgraduate Summer School that took place in September 2010.
1 Introduction

The Solar Dynamics Observatory (SDO) is a NASA mission that was launched in February 2010 to study the dynamics of the solar atmosphere and interior. It comprises three instruments, the Atmospheric Imaging Assembly (AIA), the Helioseismic and Magnetic Imager (HMI), and Extreme Ultraviolet Variability Experiment (EVE). AIA and HMI are both imaging instruments with both high spatial and temporal resolution in a range of spectral bands/data products, producing $\sim 1.5$ TB of data per day.

This large data set means that obtaining and analysing data becomes a tricky process. It will often be the case that all the data required for an analysis cannot be held in a computer's memory at once. Researchers using SDO data will have to be clever about how data acquisition and analysis is performed.

This guide aims to introduce researchers to obtaining SDO data in the first place, and some common things that might be useful in the course of a larger analysis. Tips about dealing with large data sets will be given as appropriate throughout the guide.

1.1 SSWIDL

SolarSoftWare IDL (SSWIDL) is a set of software libraries for IDL that has been developed by the solar community over the last few years. The aim is to provide tools to analyse data from solar physics missions and observatories, and many instrument teams use it to distribute software to read in, prepare, and analyse data from their instrument. The libraries can be obtained from http://www.lmsal.com/solarsoft/sswdoc/index_menu.html although as it is quite a large set of libraries, many institutes prefer to maintain an institute copy which is mounted on all relevant desktop PCs.

This guide assumes that the SDO branches of SSWIDL is available on the PC being used, and that the instruments have been added to the SSW instrument path with a line along the lines of

```
setenv SSW_INSTR "aia hmi"
```

or

```
setenv SSW_INSTR "aia hmi eis secchi trace"
```

depending on the instruments required, being in either the global .cshrc file or in a user's .cshrc file. In many institutes the system manager may have set this up already.

For more information about installing and setting up SSWIDL, please refer to the website above.

As well as assuming that SSWIDL is available, this guide also works on the basis that the reader has some familiarity with IDL.

2 Obtaining SDO/AIA and SDO/HMI data

2.1 Browsing for SDO data

Several sites offer visual browsing of the latest solar data, with options to review previous days in the archive. These sites are often useful for a quick look, but not so useful for obtaining data.

Some useful sites are:
• **The Sun Today** offered by Lockheed ([http://sdowww.lmsal.com/suntoday.html](http://sdowww.lmsal.com/suntoday.html)) shows up-to-date images of the Sun in all AIA wavelengths and provides information from an events catalog.

• **Helioviewer** ([http://www.helioviewer.org](http://www.helioviewer.org)) is a useful tool to visualise large amounts of data. This is an open source program. A database of the last 14 years of SOHO data as JPEG images are available using the online viewer or by installing the JHelioviewer on your computer (this requires the latest version of Java to be installed on your computer). This tool will be soon evolve to provide SDO/AIA images as JPEG 2000. There are options to produce movies, zooming and saving the results.

### 2.2 Data Archives

There are several ways to obtain observation from AIA and HMI. Most of these sources are based in the US, but European repositories are starting to appear.

Current data access points are as follows:

- From the HMI-AIA Joint Science Operations Center (JSOC - [http://jsoc.stanford.edu/](http://jsoc.stanford.edu/)) based at Stanford University;
- From the JSOC interface mirrored at the Royal Observatory of Belgium ([http://sdodata.oma.be/](http://sdodata.oma.be/)) working with the repository there;
- From the AIA cutout service at Lockheed ([http://lmsal.com/get_aia_data/](http://lmsal.com/get_aia_data/)) which allows smaller cutouts from the data to be extracted and downloaded;
- From the Virtual Solar Observatory (VSO - [http://sdac.virtualsolar.org/cgi/search](http://sdac.virtualsolar.org/cgi/search));
- In due course, from the University of Central Lancashire (to be implemented).

These are essentially web portals, where a web form is filled in with the required date/time and wavelength parameters, and a data request is made for subsequent download.

However, the VSO also has tools in the SolarSoftWare tree for obtaining data through SSWIDL.

### 2.3 Obtaining data via the VSO in SSW

SSW provides a set of commands for interacting with the Virtual Solar Observatory directly from within the SSWIDL environment. Two of these are of particular interest for downloading SDO data (and any other data in the VSO). These are:

- **vso_search** - which searches for data matching supplied time/date/instrument/wavelength/etc specifications;
- **vso_get** - attempts to download data corresponding to the meta-data from a previous VSO search.

#### 2.3.1 Searching for data - **vso_search**

A typical VSO search might be carried out as follows

```idl
IDL> md=vso_search(’2010/08/01 10:00:00’, ’2010/08/01 10:05:00’, instr=’aia’)```

3
which will search for all AIA data between 10:00 and 10:05 on 1st August 2010. This call will display search results similar to

Records Returned : JSOC : 187/187
Records Returned : JSOC : 0/0

which means that 187 observations were found in one catalog and none were found in a second catalog. In this case, the two catalogs were based at the JSOC, and refer to the AIA level 1 (4096 × 4096) catalog, and the AIA synoptic (1024 × 1024) catalog. Other catalogs may come on line in the future as other SDO data product become available, or if other providers feed in to the VSO.

The vso_search routine has downloaded meta-data associated with the found observations and stored them as a structure array in the variable md. The meta-data can be viewed with

IDL> help, md, /str

which displays the meta-data of the first record, something like the following:

** Structure VSORECORD, 13 tags, length=256, data length=252:
  TIME STRUCT -> VSOTIME Array[1]
  EXTENT STRUCT -> VSOEXTENT Array[1]
  WAVE STRUCT -> VSOWAVE Array[1]
  DETECTOR STRING ''
  INSTRUMENT STRING 'AIA'
  SOURCE STRING 'SDO'
  PROVIDER STRING 'JSOC'
  INFO STRING 'AIA level 1, 4096x4096'
  PHYSOBS STRING 'intensity'
  FILEID STRING 'aia_lev1:171:1059732035'
  SIZE FLOAT 66200.0
  URL STRING ''
  GETINFO STRING ''

As md is an array, meta-data for any specific observation can be viewed by specifying its number, i.e.,

IDL> help, md(10), /str

The meta-data structure also contains some sub-structures which can be viewed by

IDL> help, md.wave, /str

which displays something similar to

** Structure VSOWAVE, 4 tags, length=40, data length=40:
  MIN FLOAT 171.000
  MAX FLOAT 171.000
  TYPE STRING 'NARROW'
  UNIT STRING 'Angstrom'

So, it is clear that this particular observation is made using the 171 Å filter. Looking at the next observation

IDL> help, md(1).wave, /str
** Structure VSOWAVE, 4 tags, length=40, data length=40:
  MIN FLOAT 304.000
  MAX FLOAT 304.000
  TYPE STRING 'NARROW'
  UNIT STRING 'Angstrom'
it is clear that this observation is made using the 304 Å filter. The wavebands for the first few observations can be viewed with

```
IDL> print, md(0:14).wave.min
  171.000 304.000  94.0000  4500.00  171.000
  304.000 1600.00  94.0000  171.000  304.000
  1700.00  94.0000  171.000  304.000  1600.00
```
showing that observation in several different pass-bands have been picked up.

The VSO search can be refined further, for example, the search can be restricted to a fixed wavelength using the `wave` keyword in `vso_search`, i.e.,

```
IDL> md304 = vso_search('2010/08/01 10:00:00', '2010/08/01 10:05:00', $
              instr='aia', wave='304')
```
Records Returned : JSOC : 25/25
Records Returned : JSOC : 0/0

which finds 25 observations in the 304 Å pass-band in the desired time range.

The `sample` keyword attempts to find a subset of observations that are evenly spread throughout a time-range. For example

```
IDL> md304 = vso_search('2010/08/01 10:00:00', '2010/08/01 22:00:00', $
              instr='aia', wave='304', sample=600)
```
will try to find a subset of 304 Å observations in the 12 hour interval which are 600 seconds (10 minutes) apart.

Sometimes, the `wave` keyword is not enough to specify the required data product. For example, HMI data will all have the same wavelength range, but will offer different data products (e.g., magnetograms, dopplergrams, etc). In this situation the `physobs` keyword can be used. For example

```
IDL> mdhmi = vso_search('2010/08/01 10:00:00', '2010/08/01 10:05:00', $`
              instr='hmi', physobs='LOS_magnetic_field')
```

which will find the line-of-sight magnetograms in the specified time range. Other useful values of `physobs` include `physobs='intensity'` and `physobs='LOS_velocity'`.

There are a whole range of other keywords that can be used to refine a search. These have not been listed in detail here as they are not necessarily useful for SDO data at the moment. More details can be found with an `xdoc` search of the command.

**Exercise**

Try different VSO searches to identify different sets of data, e.g.,

- How many AIA observations are available in the first minute of the day on 15th August? How many different wavelengths are covered?
- Find 304 Å data on 12th August which are spaced 30 minutes apart;
- Find the corresponding magnetogram data for the 304 Å observations above;
- Find all of the AIA white light data (4500 Å) on 20th August. How many observation are there?
- Find HMI intensity data on the same day. How does the number of HMI observations compare to AIA?
2.3.2 Downloading data - vso_get

**Note:** Please read through this section carefully before downloading large amounts of data. The specific keywords you use will depend on your requirements (e.g., if you want to download compressed files, which directory they are put in, etc).

Once a search has identified suitable data, it needs to be downloaded to the local system. This is done using the `vso_get` command. In its simplest form, this can be done as follows:

```
IDL> md=vso_search('2010/08/01 10:00:00', '2010/08/01 10:00:30', $
        instr='aia')
```

```
Records Returned : JSOC : 10/10
Records Returned : JSOC : 0/0
```

```
IDL> status=vso_get(md)
```

This will download the requested files and place them in the current working directory (it may take a while depending on how many files have been requested).

It may more usefully be called with the `out_dir` keyword which specifies a target directory to copy the files into, e.g.,

```
IDL> status=vso_get(md, out_dir='/archive/sdo/aia/mydata/')
```

The variable `status` is a structure array that contains some download information for each file. e.g.,

```
IDL> help,status,/str
```

```
** Structure GETDATARECORD, 5 tags, length=72, data length=68:
  PROVIDER STRING 'JSOC'
  FILEID STRING 'aia_lev1:1600:1059732053'
  URL STRING 'http://vso.tuc.noao.edu/cgi-bin/drms_test/drms_export.cgi?series=aia_lev1;re'...
  INFO STRING ''
  SIZE FLOAT 0.00000
```

The fits files are now in the target directory ready for analysis.

The files that have been downloaded are uncompressed fits files and may have sizes around 65 MB. It is possible to download RICE compressed fits files instead with the keyword `/rice`, i.e.,

```
IDL> status=vso_get(md, out_dir='/archive/sdo/aia/mydata/', /rice)
```

These files are typically around 11.7 MB for AIA data and 15.4 MB for HMI (but may vary for different times and data products), and should improve download times, particularly where there is restricted bandwidth. SSW will automatically uncompress these files when they are read into SSWIDL.

Note 2, when downloading large amounts of data, give some thought to how much space the files will take up (will it fit on your local storage disk), how much bandwidth it will need to download (will it clog up everyone else’s internet traffic), how long it will take to download (will it still be downloading in a weeks time), and so on.

2.3.3 Downloading SDO data from UCLan, and other sites

By default, data is downloaded from a main archive in the US. However, other archives around the world are coming online, and it may be quicker and easier to download data from a site in (or near) your own country - for example, it may be quicker for users in the UK to try obtaining data from the University of Central Lancashire archive rather than the main US site.
The repository can be selected using the site keyword. For example, to download data from the UCLan repository, try

IDL> status=vso_get(md, out_dir='/archive/sdo/aia/mydata/', /rice, $ site='UCLan')

There are a variety of sites at the time of writing including

NSO National Solar Observatory, Tucson (US)
SAO (aka CFA) Smithsonian Astronomical Observatory, Harvard U. (US)
SDAC (aka GSFC) Solar Data Analysis Center, NASA/GSFC (US)
ORB (aka ROB) Observatoire Royal de Belgique (Belgium)
MPS Max Planck Institute for Solar System Research (Germany)
UCLan University of Central Lancashire (UK)
IAS Institut Aeronautique et Spatial (France)
KIS Kiepenheuer-Institut fur Sonnenphysik Germany)
NMSU New Mexico State University (US)

Note, not all sites will necessarily carry the full data set. For example, UCLan maintains an archive of AIA and HMI data, but only the most recent ~5 months. The routine \texttt{vso}\_search only queries the central database, not any of the other sites. This means that you may not know whether a repository has the data you want until you try downloading it with \texttt{vso}\_get. You should check the output from \texttt{vso}\_get to identify any missing files.

2.3.4 Checking which data you have downloaded

The filenames and locations on your system of downloaded files can be returned from \texttt{vso}\_get by using the filenames keyword, for example

IDL> status=vso_get(md, /rice, out_dir='/archive/sdo/', site='UCLan', $ filenames=files)

IDL> print, files
/archive/sdo/hmi.m_45s.2011.01.03_11_06_00_TAI.magnetogram.fits
/archive/sdo/hmi.m_45s.2011.01.03_11_05_15_TAI.magnetogram.fits
/archive/sdo/hmi.m_45s.2011.01.03_11_04_30_TAI.magnetogram.fits
...

This information can be used to identify any files that have not been downloaded and request them from another site, i.e.,

IDL> ii = where(strlen(files) eq 0, nii)
IDL> if (nii ne 0) then status=vso_get(md(ii), /rice, site='ORB', $ out_dir='/archive/sdo/', filenames=files2)

This could be repeated until all desired files had been obtained.

Note, it may be that in future a VSO routine to probe multiple sites will become available.

2.3.5 Further sorting of data

It is possible to apply further sorting to your search in SSWIDL before requesting files. For example

IDL> md=vso_search('2010/08/01 10:00:00', '2010/08/01 10:05:00', $ instr='aia')

Records Returned : JSOC : 187/187
would retrieve only the first five files found.

IDL> md=vso_search(’2010/08/01 10:00:00’, ’2010/08/01 10:05:00’, $ 
                   instr=’aia’)

Records Returned : JSOC : 187/187
Records Returned : JSOC : 0/0

IDL> status=vso_get(md([4,15,25]), out_dir=’/archive/sdo/aia/mydata/’)

would retrieve the 5th, 16th and 26th files in the list (remember, IDL starts counting at 0).

IDL> md=vso_search(’2010/08/01 10:00:00’, ’2010/08/01 10:05:00’, $ 
                   instr=’aia’)

Records Returned : JSOC : 187/187
Records Returned : JSOC : 0/0

IDL> ii = indgen(38)*5

IDL> status=vso_get(md(ii), out_dir=’/archive/sdo/aia/mydata/’)

would retrieve every 5th file found.

IDL> md=vso_search(’2010/08/01 10:00:00’, ’2010/08/01 10:05:00’, $ 
                   instr=’aia’)

Records Returned : JSOC : 187/187
Records Returned : JSOC : 0/0

IDL> ii = where(md.wave.min eq ’171’)

IDL> status=vso_get(md(ii), out_dir=’/archive/sdo/aia/mydata/’)

would identify all of the 171 Å data from the meta-data and retrieve only those files (though in practice, you 
might check to see if any 171 Å files had been found first).

Any reasonable IDL array indexing and searching should help refine the search further prior to downloading 
the data.

**Exercise**

Perform and refine a search on SDO data and download a couple of files to try out VSO downloads.

### 3 Reading SDO data into SSWIDL

#### 3.1 Using `read_sdo`

Now that some SDO data has been obtained, it needs to be read into SSWIDL. This can be done using the 
`read_sdo` command. For example

```idl```
IDL> read_sdo, ’aia_file.fits’, index, data
```
reads in the data from the supplied file `aia_file.fits`, and stores the fits header in the structure `index` 
and the SDO image in the array `data`. Querying the variables should return something like the following

```idl```
IDL> help,index,data
INDEX STRUCT = -> MS_020517472001 Array[1]
DATA LONG = Array[4096, 4096]
```

The size of the SDO image (4096 × 4096) can easily be seen.
Multiple files can be read in at once by supplying an array of file names to be read in instead of a single file name. The SSW `findfile` command can be used to search for files, e.g.,

```
IDL> afiles=findfile('aia*.fits')
IDL> read_sdo, afiles, aindex, adata
```

uses the `findfile` command to find all of the fits files in the current working directory which begin with aia and end with `.fits`, then the `read_sdo` command to read all of those files in. Querying the output variables should provide information similar to the following:

```
IDL> help, aindex, adata
AINDEX STRUCT = -> MS_020517472001 Array[10]
ADATA LONG = Array[4096, 4096, 10]
```

In this case, 10 files have been read in, and the size of the structure array, `aindex`, and the third dimension of `adata` reflect this.

The `findfile` function can be supplied with a directory path to look for files in different directories, e.g.,

```
IDL> afiles=findfile('/archive/sdo/aia/aia*.fits')
```

For the data provided on the memory sticks, the path could be something like

```
IDL> afiles=findfile('/media/USB/AIA_5min/aia*.fits')
IDL> read_sdo, afiles, aindex, adata
```

**Warning**: think carefully before reading in large numbers of files at once. If each file is \(\sim 65\) MB, then it doesn’t take too many files to fill up the available computer memory. For example, 15 images will require \(\sim 1\) GB of free memory. Exceeding available memory will cause the computer to start swapping leading to a painful decrease in speed.

You may (initially) only want to look at the fits headers of a file/files. This can be done by supplying the `/nodata` flag to the `read_sdo` command, e.g.,

```
IDL> read_sdo, afiles, aindex, adata, /nodata
```

```
IDL> help, aindex, adata
AINDEX STRUCT = -> MS_020517472001 Array[10]
ADATA UNDEFINED = <Undefined>
```

The `adata` variable is left completely empty allowing the fits headers to be queried (perhaps with further search refinements made) before reading in the data.

It is often the case that only a small part of an SDO image is of interest, and the `read_sdo` routine allows only a sub-field of the data to be read in to SSWIDL. This allows a region to be identified by other means (e.g., by examining only a single image in the sequence), and that selected portion of the image in a larger number of files to be read in at once.

To extract a sub-field, four additional inputs are required

```
read_sdo, afiles, aindex, adata, llx, lly, nx, ny
```

where \((llx, lly)\) are the pixel positions of the lower-left corner of the required cutout and \((nx, ny)\) is the size of the sub-field. So

```
IDL> read_sdo, afiles, aindex, adata, 3100, 2300, 800, 800
```

will cut out an \(800 \times 800\) pixel region from each image with a lower left hand corner located at pixel position \((3100, 2300)\) in the full image. Querying the two variables confirms this, i.e.,

```
IDL> help, aindex, adata
```
3.2 Reading in compressed data

RICE compressed fits files can also be read into SSWIDL in exactly the same way. When this is done, an uncompressed version of the fits file is created in the /tmp directory (a message to this effect is displayed). It is good practice to delete these files when you are done with them, i.e.,

```
rm /tmp/AIA*.fits /tmp/HMI*.fits
```

in the terminal window (not in SSWIDL), because when the /tmp directory is full, read_sdo will be unable to uncompress new files and will display error messages.

However, there are extra keywords that can be used with read_sdo to help manage this.

The command

```idl
IDL> read_sdo, afiles(0), index, data, /uncomp_delete
```

will uncompress a file into /tmp, read it in to SSWIDL, and then delete the uncompressed file from /tmp - and so clean up after itself. Note, there is also a /comp_delete flag that will keep the uncompressed files and delete the original compressed ones - be careful that you use the correct one.

Alternatively, it may be appropriate to keep uncompressed files (for quicker access at a future data), but store them in another directory. This can be done with

```idl
IDL> read_sdo, afiles, index, data, $
    parent_out='/archive/sdo/uncompressed_files/
```

which will uncompress the files and place them in the chosen directory (you may want to use the /comp_delete flag in this case so that you keep only the uncompressed files).

Finally, to only uncompress data, but not read it in to SSWIDL, the following command can be used

```idl
IDL> read_sdo, afiles, index, data, /only_uncompress, $
    parent_out='/archive/sdo/uncompressed_files/
```

This might be useful for uncompressing a large number of files in one go.

**Exercise**

Try reading some of the data that you have downloaded (or have been supplied with) into SSWIDL. Query the data with the help command.

3.3 Using aia_prep

The AIA data supplied by most repositories, and the VSO, is at level 1. There is also level 1.5 data which adds in some additional calibration such as translation, rotation and scaling for improved comparison between wavelengths, and fine tuning for better image statistics. Details of what is included in this calibration can be found at the AIA homepage (http://aia.lmsal.com/).

Level 1 AIA files can be brought up to level 1.5 by using the routine aia_prep. The simplest usage of this routine is as follows

```idl
IDL> files=findfile('/archive/sdo/aia/mydata/aia.lev1*.fits')
IDL> aia_prep, files, -1, index, data
```
The array files contains a list of AIA fits files. The flag -1 indicates that you wish to read in and convert all of files. The fits header and data are stored in index and data.

If you only want to read in a selection of the data in files, then -1 can be replaced with an array index. For example, suppose files contains 10 fits files, but you only wish to read in every other file, you could do something like

IDL> ii = indgen(5)*2
IDL> aia_prep, files, ii, index, data

More complex indices can be created as desired.

Level 1.5 fits files may also be created by aia_prep using the /do_write_fits flag, e.g.,

IDL> aia_prep, files, -1, /do_write_fits, outdir='/archive/sdo/aia/L1.5'

The use of the outdir keyword to specify a target directory, as at the time of writing the routine will reuse the same file name (and not modify it to reflect the fact the data is now at level 1.5). This data may not be as compressed as the level 1 data.

Note, at the time of writing aia_prep does not have keywords to delete uncompressed files from /tmp, or redirect uncompressed files to other directories.

4 Plotting SDO data

4.1 Basic plotting

Once data has been read in to SSWIDL, the first thing that many people want to do is look at the image. Two-dimensional image data (from SDO or many other sources) can easily be displayed in SSWIDL using the plot_image command. For example,

IDL> plot_image, data

This should create a graphics window with the SDO image displayed within. The axes have units of pixels.

This image may not come out too well (e.g., figure 1 top-left), perhaps it is too dim or bright. This is because the data has been scaled linearly between the lowest and highest value pixels. Alternative minimum and maximum values can be supplied to select a contrast range. This is done with the min and max keywords, e.g.,

IDL> plot_image, data, min=100, max=2500

This scales the image so that any pixel value below 100 is black and any above 2500 is white. Values in-between are linearly scaled to different shades of grey (e.g., figure 1 top-right). It may be useful to find the minimum and maximum pixel values of an image by using the minmax command:

IDL> print, minmax(data)
0   15800

Sometimes it may be appropriate to apply a nonlinear scaling. The easiest way to do this is to use the appropriate IDL function on the data. Some common functions are

IDL> plot_image, sqrt(data), min=sqrt(100), max=sqrt(2500)
IDL> plot_image, alog(data), min=alog(100), max=alog(2500)

which give a square root and logarithmic scaling accordingly (e.g., figure 1 bottom-left and right). Note that the function must also be applied to the min/max scalings as well. Be careful to avoid illegal scalings (e.g., the square root of a negative number or the log of zero) as this may produce unexpected results.
4.2 Plotting one image from a larger array

Often, multiple images may have been read in to a single array, e.g.,

```
IDL> help, adata
ADATA LONG = Array[4096, 4096, 10]
```

to plot one of these images, the array must be appropriately indexed, e.g.,

```
IDL> plot_image, adata(*,*,2), min=100, max=2500
```

plots the third index in the sequence (IDL counts from 0). The *s indicate that all columns and rows of the image will be plotted. Changing the 2 to a different number will plot a different image in the sequence.

**Exercise**
Try plotting some of the data that you have read in with different scalings.

4.3 Plotting part of an image

![Figure 2: AIA 193 Å data plotted showing (left) the full disk; (right) a cutout of an active region from the disk. The displayed cutout is shown as the white box in the left image.](image)

Plotting the complete image is fine, but often only a certain part of the image is interesting, and it would be useful to fill the graphics window with only that part of the image. This can be done by replacing the `*s in the above command to the desired cutout range, e.g.,

```idl
IDL> plot_image, data(1950:2549,1100:1699), min=100, max=2500
IDL> plot_image, adata(1950:2549,1100:1699,2), min=100, max=2500
```

This command plots the part of the image between pixel number 1950 and 2549 (inclusive) in the x-direction, and between 1100 and 1699 (inclusive) in the y-direction, as shown in figure 2. The numbers can be changed to represent the desired region within the SDO image.

**Exercise**

Try plotting different parts of images that you have previously read in.

4.4 Plotting images in colour

The standard IDL `loadct` command can be used to change the colour table used for the image. The default is greyscale, but can be changed by

```idl
IDL> loadct
0- B-W LINEAR 14- STEPS 28- Hardcandy
1- BLUE/WHITE 15- STERN SPECIAL 29- Nature
2- GRN-RED-BLU-WHT 16- Haze 30- Ocean
3- RED TEMPERATURE 17- Blue - Pastel - R 31- Peppermint
4- BLUE/GREEN/RED/YE 18- Pastels 32- Plasma
5- STD GAMMA-II 19- Hue Sat Lightness 33- Blue-Red
6- PRISM 20- Hue Sat Lightness 34- Rainbow
```
Figure 3: AIA 193 Å data plotted with (left) a greyscale colour table; (right) the proposed AIA 193 Å colour table.

However, this is restricted to the default IDL colour tables and may not include the desired scheme. SDO/AIA images have a set of proposed colour schemes for different spectral pass-bands. These can be accessed using the command `aia_lct`, e.g.,

```idl```
IDL> aia_lct, rr, gg, bb, wavelnth=193, /load
```
loads the proposed colour table for the 193 pass-band into IDL (it also stores the red, green, and blue component mixes in the arrays rr, gg, bb).

So a 193 image can be plotted by doing the following

```idl```
IDL> read_sdo, ‘aia_193_file.fits’, index, data
IDL> aia_lct, rr, gg, bb, wavelnth=193, /load
IDL> plot_image, sqrt(data), min=sqrt(100), max=sqrt(2500)
```

and typical results can be seen in figure 3.

**Exercise**
Try plotting images and sub-field images that you have previously tried using different colour tables. Do
different colour tables need different scalings?

4.5 Plotting HMI data

Data from SDO/HMI can be read in to SSWIDL and plotted in exactly the same way as described for the AIA
data above, i.e.,

IDL> read_sdo, 'hmi_file.fits', mindex, mdata

Figure 4: HMI data may be stored upside down in the fits file (left), this can easily be corrected for (right) using
the rotate command. Comparison with the corresponding AIA 193 Å observation in figure 3 indicates that
active regions in the rotated image are co-located with their coronal emission.

However, there are a couple of things to bear in mind when using HMI data. First, the data may be upside
down. This can be checked by examining the fits header

IDL> print, mindex.crota2
  180

This is the rotation angle of the image and if it is 180 then it is upside down. This can easily be corrected with

IDL> mdata = rotate(temporary(mdata), 2)
IDL> plot_image, mdata, min=-10000, max=10000

as shown in figure 4, though this must be done on an image by image basis for a sequence of images, e.g.,

IDL> nims=n_elements(mindex)
IDL> for i=0,nims-1 do mdata(*,*,i)=rotate(temporary(mdata(*,*,i)),2)

The second issue is that at the time of writing, read_sdo does not properly scale the image. If the following
message is printed

IDL> read_sdo, mfiles, mindex, mdata
-----------------------------------------------------------------------
| Not yet trained to apply this scaling: bsr,bsc      0.00000, 0.0100000 |
-----------------------------------------------------------------------
then the image has not been scaled properly. In the above case, it needs to be multiplied by $b_s c$ which is 0.01 to put it into physical units, i.e.,

```idl
IDL> plot_image, 0.01*mdata, min=-100, max=100
```

**Exercise**

Try reading in and plotting some HMI data, rotating as appropriate. Plot a corresponding AIA image in a different graphics window and see if the locations of the active regions agree.

## 5 Examining the fits headers

So far the fits headers have been loaded into SSWIDL and the `index.crota2` keyword has been mentioned. However, the information within the fits header can be far more useful than this. Contained within the fits header is a wealth of information about the corresponding observation such as the instrument, time taken, pass band, resolution, and so on.

### 5.1 Browsing the fits headers

To examine fits header information use the `help` command with the `/struct` keyword, i.e.,

```idl
IDL> help, index, /str
```

**Structure MS_020517472001, 176 tags, length=1560, data length=1500:

- **SIMPLE** INT 1
- **BITPIX** LONG 32
- **NAXIS** LONG 2
- **NAXIS1** LONG 4096
- **NAXIS2** LONG 4096
- **EXTEND** INT 1
- **DATE_OBS** STRING ’2010-06-13T23:59:30.63Z’

This lists all the available keywords, their data type, and their values. For example, `naxis` gives the number of dimensions of the observation, `naxis1` and `naxis2` give the array size of the stored data. Note, `/str` is short for `/structure`.

Individual elements of an array of fits headers can be accessed by specifying the desired number, e.g.,

```idl
IDL> help, index(4), /str
```

### 5.2 Useful AIA and HMI fits keywords

There are only a small number of standard fits keywords that all fits files must have, most of the keywords are chosen by the specific instrument team depending on the capabilities of the instrument itself (for example spectroscopic data may require different information than imager data), many instrument teams will produce a fits keywords specification document. However, there are some fairly common keywords that many instrument teams use.

There are too many AIA and HMI keywords to describe in full here, but some of the more useful ones are described in table 1. A complete list can be found on the JSOC website.
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>naxis1</td>
<td>long</td>
<td>pixel size of image in x-direction</td>
</tr>
<tr>
<td>naxis2</td>
<td>long</td>
<td>pixel size of image in y-direction</td>
</tr>
<tr>
<td>telescope</td>
<td>string</td>
<td>telescope/observatory name (e.g., SDO)</td>
</tr>
<tr>
<td>instrume</td>
<td>string</td>
<td>instrument name (e.g., AIA)</td>
</tr>
<tr>
<td>wavelnth</td>
<td>long</td>
<td>wave band of images (e.g., 193)</td>
</tr>
<tr>
<td>waveunit</td>
<td>string</td>
<td>units of wavelnth (e.g., angstrom)</td>
</tr>
<tr>
<td>date_obs</td>
<td>string</td>
<td>date and time of start of observation</td>
</tr>
<tr>
<td>t_obs</td>
<td>string</td>
<td>date and time of middle of observation</td>
</tr>
<tr>
<td>exptime</td>
<td>float</td>
<td>exposure time of observation - AIA only</td>
</tr>
<tr>
<td>cadence</td>
<td>long</td>
<td>cadence of image sequence - HMI only</td>
</tr>
<tr>
<td>crpix1</td>
<td>long</td>
<td>reference pixel in the x-direction (1 is centre of lefthand pixel)</td>
</tr>
<tr>
<td>crpix2</td>
<td>long</td>
<td>reference pixel in the y-direction (1 is centre of bottom pixel)</td>
</tr>
<tr>
<td>crval1</td>
<td>long</td>
<td>location of reference pixel in x-direction (e.g., 0 is Sun centre)</td>
</tr>
<tr>
<td>crval2</td>
<td>long</td>
<td>location of reference pixel in y-direction (e.g., 0 is Sun centre)</td>
</tr>
<tr>
<td>cdelt1</td>
<td>float</td>
<td>pixel size in x-direction</td>
</tr>
<tr>
<td>cdelt2</td>
<td>float</td>
<td>pixel size in y-direction</td>
</tr>
<tr>
<td>ctype1</td>
<td>string</td>
<td>coordinate system for crval1</td>
</tr>
<tr>
<td>ctype2</td>
<td>string</td>
<td>coordinate system for crval2</td>
</tr>
<tr>
<td>cunit1</td>
<td>string</td>
<td>units of crval1 and cdelt1</td>
</tr>
<tr>
<td>cunit2</td>
<td>string</td>
<td>units of crval2 and cdelt2</td>
</tr>
<tr>
<td>crota2</td>
<td>double</td>
<td>rotation angle about reference pixel</td>
</tr>
<tr>
<td>r_sun</td>
<td>float</td>
<td>radius of the Sun in image pixels</td>
</tr>
<tr>
<td>rsun_ref</td>
<td>double</td>
<td>radius of the Sun in metres - AIA only</td>
</tr>
<tr>
<td>rsun_obs</td>
<td>float</td>
<td>radius of the Sun in observing units (e.g., arcseconds)</td>
</tr>
<tr>
<td>content</td>
<td>string</td>
<td>data product (e.g., magnetogram) - HMI only</td>
</tr>
<tr>
<td>bunit</td>
<td>string</td>
<td>units of image data (e.g., Gauss) - HMI only</td>
</tr>
</tbody>
</table>

Table 1: Description of some useful fits keywords for SDO/AIA and SDO/HMI data.

The help command allows a quick look at all of the fits keywords, but it is also useful to be able to access individual keywords. This can be done as follows:

IDL> print, index.date_obs
2010-06-13T23:59:30.63Z
IDL> print, aindex(4).date_obs
2010-06-13T00:00:54.60Z
IDL> xcen = index.crpix1
IDL> ycen = index.crpix2
IDL> print, xcen, ycen
2047 2053
IDL> i193 = where(aindex.wavelnth eq 193)
IDL> print, i193

Exercise
Examine some of the fits headers in data that you have read into SSWIDL and pick out the values of useful keywords.

5.3 Using fits keywords to refine data searches

This last line of the previous case
IDL> i193 = where(aindex.wavelnth eq 193)

indicates how the fits headers can be used for a more refined search. Consider the following example

IDL> afiles = findfile('aia*.fits')
IDL> read_sdo, afiles, aindex, adata, /nodata
IDL> print, aindex.wavelnth

```
171 211 94 335 1600 193 304 131
171 211 94 335 1700 193 304 131
171 211 94 335 1600 193 304 131
171 211 94 335 1700 193 304 131
171 211 94 335 1600 193 304 131
171 211 94 335 1700 193 304 131
```

IDL> i304 = where(aindex.wavelnth eq 304)
IDL> afiles304 = afiles(i304)
IDL> read_sdo, afiles304, aindex304, adata304
IDL> aia_lct, rr, gg, bb, wavelnth=304, /load
IDL> plot_image, sqrt(adata304(*,*,0)), min=sqrt(10), max=sqrt(1000)

This identifies a set of AIA files and reads in the fits headers only. Printing the wave band information shows that the fits files contain images taken in a range of different pass bands. The `where` function can be used to find the indices of all of the 304 Å images which is used to extract the subset of the `afiles` array relating to 304 Å data. This subset is then read in and the first image plotted (in colour).

**Exercise**

Use the `where` function to perform more refined searches on your fits files.

### 5.4 Making plot axes more meaningful

![Figure 5: An AIA 304 Å image displayed using `plot_image` (left) without the use of any extra arguments, (right) using the fits header to convert the scale to arcseconds.](image)

The fits keywords can be used to convert the axes of the plot into more meaningful units, e.g.,
IDL> plot_image, sqrt(adata304(*,*,0)), min=sqrt(10), max=sqrt(1000), $ 
origin=-[(aindex(0).crpix1-1)*aindex(0).cdelt1, $ 
   (aindex(0).crpix2-1)*aindex(0).cdelt2], $ 
scale=[aindex(0).cdelt1,aindex(0).cdelt2], $ 
xtitle='X-position (arcseconds)', $ 
ytitle='Y-position (arcseconds)'

which produces the image in figure 5 right.

Extra arguments have been added to the plot_image command. The scale argument takes a two element array containing the x- and y-scales of the images, this is stored in the cdelt1 and cdelt2 keywords. The origin argument is also a two element array that defines the lower-lefthand corner of the axes. This can be calculated using the negative location of Sun centre given by -crpix1 and -crpix2 and multiplying by the pixel size to convert to arcseconds. Note, the aindex(0).crpix1-1 and aindex(0).crpix2-1 adjusts for the fact that crpix counts the first pixel as 1 where SSWIDL counts it as 0. A quick check of crval1,crval2,cunit1 and cunit2

IDL> print, aindex(0).crval1, aindex(0).crval2
  0    0
IDL> print, aindex(0).cunit1, ' ', aindex(0).cunit2
  arcsec arcsec

confirms that the reference pixel is indeed Sun centre and the units are arcseconds. The xtitle and ytitle argument allow labelling of the x- and y-axes.

Exercise

Try modifying the axes of plots of data that you have previously read in.

5.5 Working with date/time keywords

The data/time fits keywords (e.g., date_obs) are given as strings, e.g.,

IDL> print,index.date_obs
2010-08-09T19:49:00.34Z

This is fine for human inspection or labelling of plots, but sometimes it is useful to be able to perform time-based calculations, and a string is not so useful for doing this.

A date/time string can be converted into seconds (from a standard reference time) using the anytim2tai function, i.e.,

IDL> secs = anytim2tai(aindex304.date_obs)
IDL> print, secs
  1.6600746e+09  1.6600746e+09  1.6600747e+09  1.6600748e+09
  1.6600748e+09  1.6600749e+09  1.6600749e+09

Now these values may not initially seem useful as the reference time was sufficiently far back that the values have become quite large. However, this can be brought under control by choosing a more useful reference time and subtracting this from the above result. For example

IDL> tref='2010-08-09T19:00:00.00Z'
IDL> sref=anytim2tai(tref)
IDL> secs = anytim2tai(aindex304.date_obs) - sref
IDL> print, secs
  2948.1300  3008.1200  3068.1300  3128.1200
IDL> mins = secs/60.0
IDL> print,mins
49.135500 50.135333 51.135500 52.135333
53.135500 54.135333 55.135500

gives the observation times in seconds and minutes respectively from the new reference time.

This information might be useful for refining a search (e.g., identifying an image every 10 minutes), or for plotting quantities that vary with time.

**Exercise**

Manipulate the date/time information from fits headers that you have read in to SSWIDL.

### 5.6 Rotating HMI data

Section 4.5 explained how sometimes HMI data might be rotated by 180° and how to rotate the image to agree with the AIA observations, i.e.,

IDL> mdata = rotate(temporary(mdata), 2)
IDL> plot_image, mdata, min=-10000, max=10000

This rotation will most likely change the location of the reference pixel. If the fits headers are to be used for further analysis, then it may be sensible to update this information when the image is rotated, i.e.,

IDL> mindex.crpix1 = mindex.naxis1 - mindex.crpix1 + 1
IDL> mindex.crpix2 = mindex.naxis2 - mindex.crpix2 + 1
IDL> mindex.crota2 = mindex.crota2 - 180

Again, the +1 is due to the crpix keywords counting the centre of the first pixel as 1.

**Exercise**

Read in an HMI file, rotate the data and modify the keywords, then plot the image with the origin at Sun-centre and the axes in arcseconds.

### 6 Tracking a region across the disk

Section 3 showed how a sub-field of an SDO observation could be read into SSWIDL. Picking a fixed region for analysis over a few minutes is fine, but over longer periods of time the feature of interest will likely move out of the sub-field as the Sun rotates. In order to follow a feature, its position as it rotates across the solar disk must be calculated. This can be done using the rot_xy function. Given a position in arcseconds on the solar disk, a reference time, and a required time, this function will calculate the location of the reference position at the required time (which can be before the reference time as well as after). For example

IDL> xcen=-50.0
IDL> ycen=100.0
IDL> tref='2010-08-09T12:00:00.00Z'
IDL> treq='2010-08-10T12:00:00.00Z'
IDL> npos = rot_xy(xcen,ycen,tstart=tref,tend=treq)
IDL> print,npos
163.612
100.515
In this case the reference position is \((-50, 100)\) arcseconds from disk centre at a reference time of 12:00 on 9th August. The position of this region at 12:00 on 10th August is given by the result of \(\text{rot}_xy\), and is \((163.612, 100.515)\) arcseconds from disk centre.

### 6.1 Co-aligning a pair of observations

When a feature is being tracked in actual SDO observations, the information in the fits headers can be used to calculate the required clipping region for each file.

First a reference image and region must be selected, e.g.,

```
IDL> afiles=findfile('/archive/aia_set_*.fits')
IDL> help, afiles
IDL> read_sdo, afiles(8), rindex, rdata
IDL> plot_image, sqrt(rdata), min=sqrt(100), max=sqrt(2500)
IDL> plot_image, sqrt(rdata(2130:2429,2350:2649)), min=10, max=50
```

From plotting sub-fields, a cutout of 2130-2429 pixels in the x-direction, and 2350-2649 pixels in the y-direction has been selected, which is a \(300 \times 300\) pixel cutout. The centre of this cutout in the x- and y-directions is then given by

```
IDL> xcenp = (2130.0 + 2429.0)/2.0
IDL> ycenp = (2350.0 + 2649.0)/2.0
IDL> print, xcenp, ycenp
  2279.50  2499.50
```

and the size of the cutout is

```
IDL> xsize = 2429.0 - 2130.0 + 1.0
IDL> ysize = 2649.0 - 2350.0 + 1.0
IDL> print, xsize, ysize
  300.000  300.000
```

The position of the centre of the cutout can now be converted into arcseconds by

```
IDL> xcena = (xcenp - rindex.crpix1 + 1)*rindex.cdelt1
IDL> ycena = (ycenp - rindex.crpix2 + 1)*rindex.cdelt2
IDL> print, xcena, ycena
  140.11821  268.53490
```

which has the associated reference time given by

```
IDL> tref = rindex.date_obs
IDL> print, tref
2010-08-09T00:00:07.84Z
```

The corresponding sub-field for a different observation can be calculated as follows

```
IDL> read_sdo, afiles(10), cindex, cdata, /nodata
IDL> treq = cindex.date_obs
IDL> print, treq
2010-08-10T00:00:07.84Z
IDL> npos = rot_xy(xcena, ycena, tstart=tref, tend=treq)
IDL> print, npos
  330.983
```
Figure 6: An AIA 193 Å clip at (left) 00:00 on 9th August 2010, and (right) the tracked sub-field 1 day later.

273.516

which is the new position of the centre of the sub-field in arcseconds. This can be converted into pixel position by

$$\begin{align*}
\text{IDL> } & \text{xcenp2} = \text{npos}(0)/\text{cindex}.\text{cdelt1} + \text{cindex}.\text{crpix1} - 1 \\
\text{IDL> } & \text{ycenp2} = \text{npos}(1)/\text{cindex}.\text{cdelt2} + \text{cindex}.\text{crpix2} - 1 \\
\text{IDL> } & \text{print, xcenp2, ycenp2} \\
& \quad 2597.5673 \quad 2507.8009
\end{align*}$$

which with \text{xsize} and \text{ysize} can be used to calculate the location of the lower-lefthand corner as follows

$$\begin{align*}
\text{IDL> } & \text{xlo} = \text{round}(\text{xcenp2} - \text{xsize}/2.0) \\
\text{IDL> } & \text{ylo} = \text{round}(\text{ycenp2} - \text{ysize}/2.0) \\
\text{IDL> } & \text{print, xlo, ylo} \\
& \quad 2448 \quad 2358
\end{align*}$$

The required sub-field can now be read in using \text{read_sdo}

$$\begin{align*}
\text{IDL> } & \text{read_sdo, afiles(10), cindex, cclip, xlo, ylo, xsize, ysize} \\
\text{IDL> } & \text{plot_image, sqrt(cclip), min=sqrt(100), max=sqrt(2500)}
\end{align*}$$

assuming that the two observations are at the same resolution. Typical results can be seen in figure 6.

\textbf{Exercise}

Select a reference image, choose a region of interest, and track the region into a second image at a different time.

\textbf{6.2 \ Co-aligning a sequence of observations}

The above methodology is okay for co-aligning a pair of images, but repeating it for a sequence of observations is not necessarily the most efficient approach. Instead, sensible use of loops and arrays with a little programming may be better.

With the reference clip selected above, consider the following temporary program for tracking the region in all
of the observations in \texttt{afiles}.

\begin{verbatim}
IDL> .r
- nf = n_elements(afiles)
- ccube = fltarr(xsize, ysize, nf)
- for n=0, nf-1 do begin
  - read_sdo, afiles(n), cindex, cdata, /nodata
  - npos = rot_xy(xcena, ycena, tstart=tref, tend=cindex.date_obs)
  - xcenp2 = npos(0)/cindex.cdelt1 + cindex.crpix1 - 1
  - ycenp2 = npos(1)/cindex.cdelt2 + cindex.crpix2 - 1
  - xlo = round(xcenp2 - xsize/2.0)
  - ylo = round(ycenp2 - ysize/2.0)
  - read_sdo, afiles(n), cindex, cclip, xlo, ylo, xsize, ysize
  - ccube(*,*,n) = cclip
- endfor
- end
\end{verbatim}

This tracks and clips the region of interest from each of the files in turn, and stores the results in the three-dimensional array \texttt{ccube}.

If the fits headers are required, then these should also be collected within the loop, as keywords such as \texttt{crpix1}, \texttt{crpix2}, \texttt{naxis1}, and \texttt{naxis2} are modified by \texttt{read_sdo} to reflect the clip taken.

Note, while this tracking has been presented as a temporary program, for more general use it may be worth converting it into a user defined program that can be easily reused in different cases.

![Figure 7: An AIA 193 \AA{} clip tracked at 12 hour intervals starting at 00:00 on 5th August 2010.](image)

If there are not too many images, then they can all be viewed by something like

\begin{verbatim}
IDL> !p.multi=[0,6,3]
IDL> for n=0, nf-1 do plot_image, sqrt(ccube(*,*,n)), min=sqrt(100), max=sqrt(2500)
\end{verbatim}

where \texttt{!p.multi} specifies the number of plots to a page (start with the first location, with 6 columns and 3 rows, e.g., figure 7). Otherwise, some other form of presentation (such as a movie) is required.

Note, IDL will persist with the multiplot window - wrapping round to the first location once it reaches the end.
- until cancelled. This can be done with

IDL> !p.multi = 0

**Exercise**

Using the sequence of AIA 193 Å files on the memory stick, select a reference image and region of interest, and clip all of the other files to that region. Plot the region as you see fit.

Can you extract the same region from the corresponding HMI files on the memory stick.

Hint, as HMI and AIA are at different resolutions, the required HMI x- and y-sizes will need to be recalculated, this can be done with

- \[ xsize2 = xsize \times \text{rindex}.cdelt1 / \text{cindex}.cdelt1 \]
- \[ ysize2 = ysize \times \text{rindex}.cdelt2 / \text{cindex}.cdelt2 \]

Also, remember that HMI data may be upside down, so the appropriate HMI fits keywords in \text{cindex} will have to be modified before using them for calculation. Once the new centre pixel has been calculated \((xcenp2, ycenp2)\), this must be rotated once more for selecting the correct portion of the upside down data in the file, i.e.,

- \[ xcenp2rot = \text{cindex}.naxis1 - xcenp2 - 1.0 \]
- \[ ycenp2rot = \text{cindex}.naxis2 - ycenp2 - 1.0 \]

which can be used to calculate \(xlo\) and \(ylo\).

Finally, once the clip has been extracted from a file, it should be rotated before being stored in the \text{ccube} array.

### 7 Making SDO image output

Displaying SDO data within SSWIDL is fine for personal viewing of SDO data, but sooner or later a standalone image file will be required for logbooks, publications, talks, posters, and so on.

There are two basic options, a vector format such as postscript, or a bit-mapped format such as PNG or JPEG. In both cases the actual SDO data is bit-mapped, any text, plot axes, plot annotations will come out smoother in printed material in vector form, whereas correctly sized bit-mapped output tend to be better on computer screens (e.g., talks, web pages).

#### 7.1 Output as postscript

To write to a postscript file (e.g., figure 8), something like the following would be required:

IDL> set_plot, 'ps'
IDL> device, file='304output.ps', xsize=4, ysize=4, /inches, $ /color, bits=8
IDL> aia_lct, rr, gg, bb, wavelnth=304, /load
IDL> plot_image, sqrt(adata304(*,*,6)), min=sqrt(10), max=sqrt(1000), $ origin=[(aindex304(6).crpix1-1)\times aindex304(6).cdelt1, $ (aindex304(6).crpix2-1)\times aindex304(6).cdelt2], $ scale=[aindex304(6).cdelt1,aindex304(6).cdelt2], $ xtitle='X-position (arcseconds)', $ ytitle='Y-position (arcseconds)'
IDL> device, /close
IDL> set_plot, 'x'
Figure 8: SDO data can be easily output into postscript files suitable for inclusion in text documents.

This creates a file that, when printed, will produce a 4 × 4 inch colour plot of the file in question, with the x- and y-axes in arcseconds.

Any of the `plot_image` commands found in this primer can be substituted into this set of commands, additionally, `oplot` and other graphics commands may appear as well.

The postscript file generated above will be quite large (perhaps around 33 MB), so it is worth considering whether the image needs to be at full resolution. If a lower-resolution is acceptable, then the image can be resized with either of

```
IDL> lrdata1 = rebin(adata(*,*,6), 1024, 1024)
IDL> lrdata2 = congrid(adata(*,*,6), 1233, 1233)
```

The `rebin` command is when the desired resize is a simple fraction/multiple of the original size (e.g., $\frac{1}{8} \times$, $\frac{1}{4} \times$, $\frac{1}{2} \times$, $2 \times$, $3 \times$, $4 \times$, $8 \times$, etc), or a whole pixel multiple/fraction. The `congrid` function will resize to arbitrary sizes.

**Exercise**

Try writing out postscript images of different observations at different resolutions. These can be viewed under Linux with something like `gv`.

### 7.2 Output as PNG, JPEG, etc

The following is one method of writing to a JPEG file (e.g., figure 9)

```
IDL> window, 30, xsize=4096, ysize=4096, /pixmap
IDL> aia_lct, rr, gg, bb, wave=304, /load
IDL> tv, bytscl(sqrt(adata(*,*,6)), min=sqrt(10), max=sqrt(1000))
IDL> trimg = tvrd(/true)
IDL> wdel, 30
IDL> write_jpeg, 'test_im_304.jpg', trimg, true=1
```

The first line creates an imaginary graphics window in memory (reading/writing to an actual graphics window can sometimes be flawed), the third line writes the image to the imaginary window (without any axes or text),
Figure 9: SDO data can be easily output into JPEG and PNG files suitable for inclusion on web pages and in talks.

the fourth reads from the window into an array, the fifth deletes the imaginary window (it is a good idea to tidy up memory usage when it is no longer needed), and the sixth line writes to a JPEG file (note, some versions of IDL will write the file upside down, so the keyword /order must be included to correct this).

The quality of a JPEG can be controlled with the quality argument, e.g.,

```
IDL> write_jpeg, 'test_im_304.jpg', trimg, true=1, quality=85
```

The quality argument takes a value between 0 (poor) and 100 (very high), and defaults to 75 if not specified. The higher the quality, the larger the file size.

Writing a PNG has almost the same syntax, i.e.,

```
IDL> write_png, 'test_im_304.png', trimg
```

Writing to and from a graphics window (even an imaginary one) is a useful way to build up a plot, but can sometimes be slow. A quicker way to write just the image is

```
IDL> img = bytsc1(sqrt(adata(*,*)), min=sqrt(10), max=sqrt(1000))
IDL> trimg = bytarr(3,4096,4096)
IDL> trimg(0,*,*) = rr(img)
IDL> trimg(1,*,*) = gg(img)
IDL> trimg(2,*,*) = bb(img)
IDL> write_png, 'test_im_304.png', trimg
```

This method does not use any graphics windows (imaginary or otherwise) at all.

As with postscript files, it is worth considering the resolution of the image being written. Computer screens have resolutions considerably smaller than $4096 \times 4096$, and it can often be sensible to scale the image to the desired size at the outset. This can be done using `rebin` or `congrid` as shown above.

**Exercise**

Try writing out JPEG and PNG images of different observations at different resolutions.
8 Response function of the AIA telescopes

8.1 Wavelength response function

Figure 10: Wavelength response functions for six characteristic pass-bands, these are 94 Å, 131 Å, 171 Å, 193 Å, 211 Å, and 304 Å.

The AIA telescopes select the different wavelength bands using broad band filters. Each filter has a specific response function which is typically centred on the wavelength of the observed spectral line. The wavelength response functions are plotted in Figure 10 for six characteristic wavelengths.

To obtain the wavelength response functions try the following:

IDL> wave_resp = aia_get_response()
IDL> help, wave_resp, /str
IDL> help, wave_resp.a171, /str

which can be plotted with something like

IDL> plot, wave_resp.a171.wave, wave_resp.a171.ea
IDL> plot, wave_resp.a171.wave, wave_resp.a171.ea, xrange=[150,200]

It is noticeable that the filter is broader for larger wavelengths. It is clear that the 171 Å filter has a secondary peak at 175 Å. Currently there is no response functions for the visible spectral lines.
Figure 11: Temperature response functions for six characteristic pass-bands, these are 94 Å, 131 Å, 171 Å, 193 Å, 211 Å, and 304 Å.

8.2 Temperature Response function

A spectral line has a specific temperature of formation. Nevertheless, the broad band filters are responsible for a temperature response function resulting from the wavelength range observed. The different temperature response functions are summarised in Figure 11 for six characteristic wavelengths.

The temperature response functions can be obtained with

```IDL
IDL> temp_resp = aia_get_response(/temp)
IDL> help, temp_resp, /str
IDL> help, temp_resp.a171, /str
IDL> plot, temp_resp.a171.logte, temp_resp.a171.tresp
```

Most of the temperature profiles show that an AIA image does not correspond to a unique temperature, but mostly to two temperatures and a broad range of temperature. It is important for the interpretation of the AIA images to understand at which temperature the plasma is observed. For instance a plasma seen in the 94 Å filter has a temperature of 1 million degrees or close to 10 million degrees.

**Exercise**

Plot and examine the wavelength and temperature responses for other pass-bands. Can you modify the temperature plots so that the $\log_{10}(Te)$ axis becomes a plain temperature axis?