31 Aug 2023 # How to find positions of spectra on a planet **(with IRTF_slit_position.pro)**

This document describes how to use a crude IDL routine that will (usually) accurately identify the latitude/longitude (etc.) for each pixel position along a spectral slit using SCAM (slit camera) images. Caveats include:

- only tested for Keck/NIRSPEC and IRTF/iSHELL
- assumes spectral slit is aligned either along the planetary meridian or at 90° from the meridian (i.e., N-S/vertically, or E-W/horizontally)

There are also a lot of touchy parameters to set, so it is unlikely to immediately work for a new dataset without some tuning. Therefore, to help with such tuning, and maybe with conversion to Python, we first describe the logic of the approach and follow that with step-by-step instructions for using these routines on data from the 2023 IRTF Jupiter campaign.

Brief summary of overall logical flow (detailed discussion below):

1. Prepare a planetary ephemeris file (e.g., obtained from JPL Horizons). This contains necessary detail ordered by UTC, such as sub-observer longitude, angular size, etc.
2. Match scam frames to spectra using UTC header information from each. For example, `IRTF_specmatch.pro` or equivalent.
3. Loop through scam frames, finding the best fit planetary limb ellipse for each. This is obviously the hard part and involves a few steps, such as:
    a. Clean up each frame (e.g., by subtracting a sky frame, hot pixel removal, …)
    b. Find the slit position (given in Keck header; or determine empirically for iSHELL – doesn't appear to shift position once known)
    c. (For Keck only: rotate the frame such that the central meridian is aligned vertically)
    d. Perform a "good" edge detection on the cleaned frame. This can be tricky, but generally involves zeroing out sky pixels and then using a function like `canny`.
    e. Create a "Jupiter mask" (=pixel region that is shaped according to angular size, polar tilt, and platescale) with the y position aligned with a visible pole of the planet
    f. Shift the mask in x to find the best x position (=greatest pixel count within mask)
4. Save this info in some convenient way (right now as image outputs and an IDL data structure).

(1) Planetary ephemeris file

We use JPL Horizons to specify planetary ephemerides. For a single instance, you can simply use the web interface to specify all the settings (e.g., https://ssd.jpl.nasa.gov/horizons/app.html#/). There are also API commands that you can use to help extract an ephemeris "automatically" (https://ssd-api.jpl.nasa.gov/doc/horizons_file.html). I use `ephgen.pro` to do this. For example, using only the default settings to generate a link might look like:

```
IDL> x = ephgen()
```

IDL> print, x

```
https://ssd.jpl.nasa.gov/api/horizons.api?format=text&COMMAND=%27599%27&OBJ_D
ATA=%27YES%27&MAKE_EPHEM=%27YES%27&EPHEM_TYPE=%27OBSERVER%27&CENTER=%27568%27
&START_TIME=%272022-11-05%2004:00%27&STOP_TIME=%272022-11-
05%2010:00%27&STEP_SIZE=%2710%20minutes%27&QUANTITIES=%271,2,8,13,14,15,17,20
,26%27CSV_FORMAT=%27no%27CAL_FORMAT=%27both
```

Then this ephemeris info would need to be saved in a file for ease of access later. Once you have generated the link you can just save that as a text file; alternatively, you could use something like `wget` to automate it, e.g.:

```
IDL> eph_out = wget([ephem_weblink], filename=[output_filename.txt])
```

would save the [ephem_weblink] to [output_filename.txt]. Then, for ease of use, this file should be parsed and saved as an IDL .sav file. I would use `IRTF_read_ephemeris_jup2023.pro` to do that. This routine can also interpolate to get approximate sub-minute variations. Note that this routine assumes you use very specific "quantities" in generating your ephemeris: 1,2,8,13,14,15,17,20,26! These are the tick boxes from the web app custom table settings and represent the required parameters.

(2) Matching scam and spectra frames

This step involves some method for assigning each spectral frame to a series of scam frames. I.e., "what scam images were taken during the integration of this spectrum?"

Eventually this step allows you to convert from scam slit parameters to spectral spatial parameters, as there is not a 1-to-1 correspondence (different platescale on detectors, different echelle order lengths, etc.) In the meantime, it also saves time, because you don't need to fit the limb to scam images that were taken while not obtaining science data.

I use `IRTF_specmatch.pro` to do this. You may need to modify a few variables, such as directories, but it should be relatively straightforward to use otherwise. It will create an IDL file (`specvars.sav`) in the spectral directory containing several variables – see "`readme`" for more detail.

(3) Fitting the planetary limb

The routine that does this is `IRTF_slit_pos.pro`. It includes many default variables that should work OK much of the time, but also the option to specify a wide range of parameter tweaks. As an example, it may be useful to look at `IRTF_slit_position_batch.pro` as well, as that demonstrates how you might call the slit mapping routine as well as what variables might commonly need to be modified.

Clean up each frame:

Whatever helps clean up the image will ultimately help with separating the planetary limb from the background as well. Here is an example of what `IRTF_slit_pos.pro` does.

Finding the slit position:

Then we need to find the position of the slit. I leave that as an exercise for the reader, but (a) it is given in the Keck header info, and (b) for IRTF it is straightforward to find by using a flat frame to sum in x (to find the y location) and then in y (to find the x locations). Note that the slit you see in Kyle frames is the full length, however, not the one specified based on the dekker (which you can find in the header)! This is important to keep in mind when we are converting between scam and spectral pixel positions later.

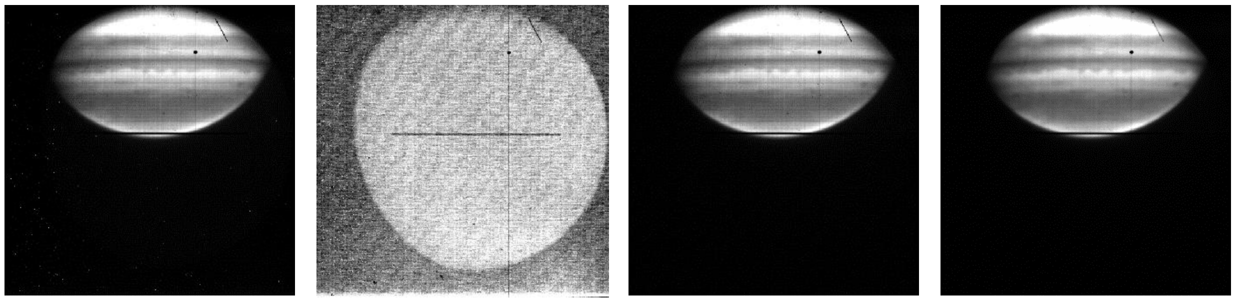
Figure 1. From left to right we have (a) the original scam image, (b) a nearby sky frame, (c) the sky subtracted image, and (d) the image with a light pass of acre.pro (hot pixel removal).

<u>Perform a "good" edge detection:</u>

Even with the quick image cleaning from above, there will likely be too much scatter to clearly delineate between "sky" and "planet" pixels in any easily-automated way. So let's clean even more! Presently, IRTF_specmatch.pro does this in a few steps (but they are very kludge-y and subject to change):

- Figure 2a: apply a crude hot pixel smoothing (e.g., threshold of 3σ and width of 12 pixels)
- Figure 2b: replace the pixel values along the slit with the mean image value (probably not needed in this example, but sometimes helps)
- Figure 2c: sum along rows and apply the same slit value correction if needed (to make sure the slit values are not "too low"; should then lead to a clear step from "sky" to "planet")
- Find "sky pixels" by (a) finding the mean and standard deviation of the sky background pixels from 2c, (b) and then using these to estimate what pixel values from the 2D image should be zeroed out (based on whether they are within nskySD σ of the mean)
- Figure 2d: at this point, we can perform an even better edge detection, so we use canny (this leaves behind only 0s and 1s)

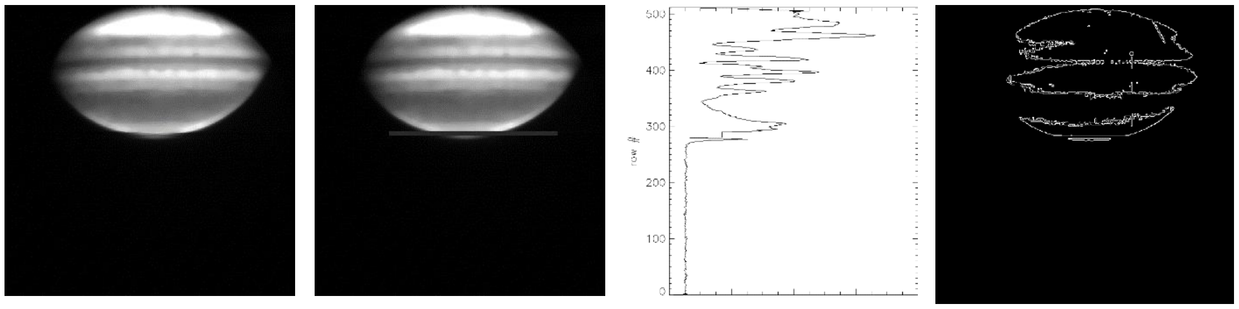

Figure 2. From left to right we have (a) a crude hot pixel smoothed version of the image in 1d, (b) the same with the slit pixels corrected, (c) the column-summed version of this, and (d) the result of using 2c to define and zero out sky pixels followed by applying the edge detection function, canny.

If the settings are just right (usually `nskySD` and `skyvalSD` may need to be altered), then we can overplot our new edge detected frame and slit position on top of our original image to confirm that everything looks good for fitting the limb. Figure 3a shows an example of what it should look like when things are going ok. Even though the red contours don't really match the planet, the ones near the pole are pretty good, even with the slit complicating matters, and those are the ones we use to fit our Jupiter ellipse.

The settings can be wrong in usually one of three ways: (1) there are red contours all over the sky (probably `skyvalSD` is too low), (2) the contours overhang the limb (probably `nskySD` is too low), or (3) the contours are either non-existent or well within the edge of the planet (probably `nskySD` is too high, `skyvalSD` may also need to be changed).

Then, based on whether there are more bright pixels in the top or bottom of the frame, we determine the slit as being in the south or north of the planet. This is important because this is the planetary limb we will try to fit with a Jupiter ellipse (that is drawn based on ephemeris and header info). We take the output from 2d and, going column-by-column, only enhance the first "edge" we see starting from the bottom. The bottommost of these white pixels (in Figure 3b) then marks the polar limb of a Jupiter ellipse, drawn here in cyan.

We next need to determine the best x pixel for the center of our Jupiter ellipse. We do this by simply shifting left and right in x to find when we have maximized the number of white pixels being within the ellipse.

At this point we know the best xy coordinates for centering our Jupiter ellipse, and we know the pixel coordinates of the spectral slit. All that is left to do is to figure out the planetary parameters for each pixel along the slit. To do this we turn to `deprob`, a routine developed by Mike Wong. Given our known positions and ephemeris, it will essentially draw a grid on the planet and we can then use this grid to extract things like latitude and longitude for each pixel position along the slit. E.g.,

```
IDL> ob_scam = deprob(xsq, ysq, ang_a, ang_b, lat_se, lon_se, npang=-90.)

IDL> lats_pc = ob_scam.lat_pc([xslitpixr[1:*]],[yslitpixr[1:*]])
```

would create a data structure, `ob_scam`, that contains things like planetocentric latitude as a function of pixel position. The second command then simply extracts that info based on known x and y slit pixel positions.

The end, success! (Other than a few important details that will be left for later, such as accounting for the dekker, and converting the positional info from scam images into positional info for spectra – though the latter is a basically just an interpolation.) The resulting fit for this example is given in Figure 4.

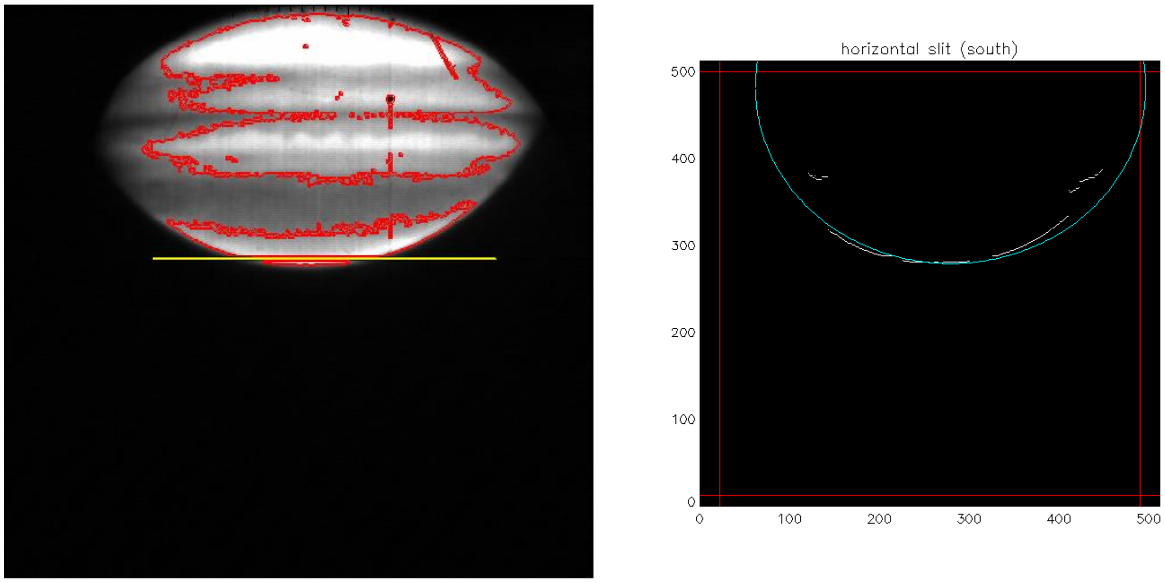

Figure 3. (left) original cleaned scam image overlaid with edge detection contours in red and the slit location in yellow. (right) The "southern edge only" edge detection frame in white with the Jupiter ellipse in cyan. Red lines mark regions of the detector that are not used in the fit.

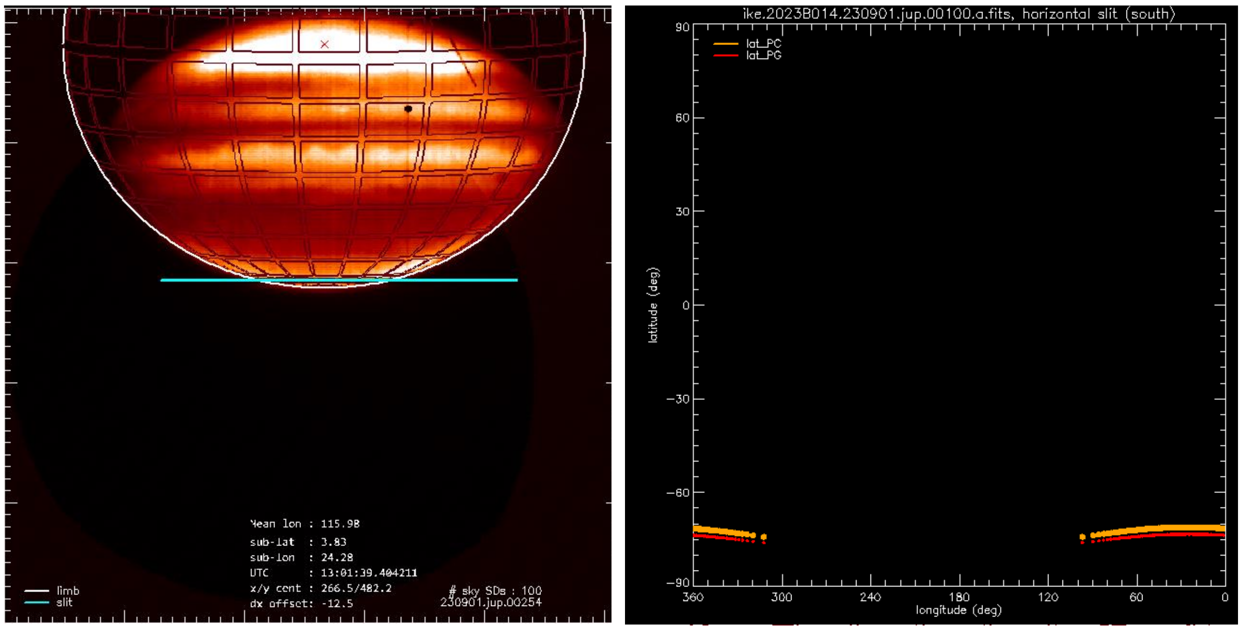
```
Mean lon : 115.98
sub-lat  : 3.83
sub-lon  : 24.28
UTC      : 13:01:39.404211
x/y cent : 266.5/482.2
dx offset: -12.5
```

Figure 4. The resulting fit, with (a) the Jupiter ellipse, grid, and slit location, and (b) corresponding planetocentric/planetographic latitudes and longitudes.

(4) Saving the outputs

`IRTF_slit_pos.pro` creates an IDL data structure output called `scamslit.sav` that is put into the corresponding scam directory. See `scamslit.readme` for details.

Additional Notes

This mapping routine is just a start. For instance, it only gives pixel positions along the center of the slit and doesn't currently account for slit width or seeing. More accurately, the slit would cover a range of latitudes and longitudes based on those parameters (e.g., see the Keck NIRSPEC data reduction guide, which does account for that). Combining positional info from spectrum to spectrum is therefore helped by mapping the slit position, but it is still probably non-trivial.

Here is a mostly complete list of all the routines described in this document, arranged in approximate order of how they would be used.

Subroutines are also identified, most of which will be standard IDL library or REDSPEC routines. We don't always identify those here (e.g., *readfits*).

Note that many of these routines also use the Coyote IDL library (and Image Magick). For more info, see http://www.idlcoyote.com/documents/programs.php#COYOTE_LIBRARY_DOWNLOAD, http://www.idlcoyote.com/graphics_tips/weboutput.php, and http://www.imagemagick.org/script/index.php.

Likely those requirements could be dropped if needed, but it might be tedious to correct each error that crops up to do so.

Program Name	(non-standard) Subroutines	Description
IRTF_specmatch	os_slash_fix	Matches scam images with spectra based on UTC
ephgen		Creates JPL Horizons web link for ephemeris
IRTF_read_ephemeris_jup2023	os_slash_fix, strfix	Reads ephemeris text file, interpolates, creates .sav.
IRTF_slit_pos	os_slash_fix	Finds the planetary coords for each slit pixel position
IRTF_slit_position_batch	os_slash_fix, strfix	Runs IRTF_slit_pos for each 2023 IRTF Jupiter date
acre		Automatic Cosmic Ray Extraction by Marc W. Buie
deprob		DEPRoject onto OBlate spheroid by Mike Wong

Many of these (basically all of mine) are in half-finished, poorly commented states, so this is just something to get things started, and hopefully code improvements and finalizations will follow soon$_{\text{TM}}$.