UCF Physics: AST 4762: Astronomical Data Analysis

Fall 2019 Final Project

Exoplanetary Transit

Measure the radius of an exoplanet. You will use data from Spitzer Space Telescope program 30825, a transit of planet HD 189733b at 8 µm observed by Harvard University Prof. David Charbonneau and his (then) graduate student Heather Knutson (now a professor at Caltech). You will search for bad pixels and perform photometry on the pre-processed data frames provided by the Spitzer Science Center. These frames are already flat-fielded, debiased, darkened, etc. You will measure the transit depth from the resulting light curve by finding the ratio of in- and out-of-eclipse data. Finally, you will calculate a radius for the planet by scaling the stellar radius.

See the file `projdesc.pdf` for general instructions.

1. Learn about the planet. Get and read the discovery paper and at least one other paper on it (the paper written for this dataset is a good one to choose). It will likely help to read one or more of Charbonneau et al. (2005) or Harrington et al. (2007), including the latter's supplementary information. These discuss InfraRed Array Camera (IRAC) systematics, but not this dataset.

2. Familiarize yourself with the data in `WebCourses/Files/data/transit`. Locate the *IRAC Data Handbook* online and skim it. Identify the science frames, the uncertainty frames, the raw frames, etc. Note that we will only be using the science frames, the Basic Calibrated Data (BCD, `_bcd.fits`). Figure out the naming scheme for files. This dataset was taken in IRAC subarray mode, which takes 64 quick exposures, stacks them in a 3D data cube, writes a FITS file, and repeats. The directory does not contain all the files in the dataset, but only the BCD files from the time near transit. It does include a directory called `fullset`, containing one full set of files, so you can see what professional astronomers get when they download data. One can also download the raw data, of course. The BCD frames resulted from processing the raw frames through Spitzer's calibration software.

3. (10 points) Start your Python script with a section of variable definitions. Put whatever constants you will need here as you encounter them below. Define variables you will use to index the frame parameters array, so that you never index that array with a constant. Functions should be defined in one or more separate files, as appropriate, and need to be documented per the standard doc header. Print the directory name where you have the data saved.

4. (20 points, including specific points below) Write a routine that reads the data and extracts header information at the same time. It will need to do a few things beyond the following items:

   (a) (3 points) Read the first data block and get any information you need from it, such as image cube size. Print the number of frames in a subarray set.

(b) (2 points) Pre-allocate a 3D data cube and a 2D frame parameter array. The latter has one row per image. Each row contains information about one frame, such as the photometry results. One piece of information you will need from the header is the exact time each image was taken. You may need others. As you work and find them, come back to the code in this step and add columns in the frame parameter array. Fill in the columns as you read data, below. Print the shape of each array.

(c) Loop over the images, starting at a given subarray set number and continuing for a given number of sets (print these), and:

   i. (1 point) Read the data and header.
   ii. (2 points) Put the science data into the data cube. Remember that you are inserting them in blocks of 64 frames.
   iii. (4 points) Find the observation time in the header and put it in the frame parameters array along with the frame number. Use the frame interval to calculate the timing of the subsequent frames in the set. Also populate the frame parameters array with any other header values you may need (you may discover these as you go).
   In the header, FRAMTIME is the interval between images in one set. All of the time keywords (DATE_OBS, MJD_OBS, UTCS_OBS, SCLK_OBS) refer to the start of the first frame in a set. Be careful about units and precision in picking one. EXPTIME is the duration of one exposure, with the difference between that and FRAMTIME being the time to transmit the data from the readout electronics into memory and reset the array for the next integration. So, take half of EXPTIME and add it to a time keyword (be careful of units) to get the mid-time of the first frame. The subsequent images in the set are separated by FRAMTIME. There is a longer gap between sets, to move the image set from the camera to the spacecraft memory.
   Every 10th file, print the filename and its DATE_OBS.

5. (10 points) Run the routine, starting at subarray set 0187 and ending at set 0686 (inclusive). Look at the results. Look at a few frames in detail in DS9. Skim the rest of the images by sending some to DS9 in rapid sequence. Check that you do not have blank frames at the end of your data array, etc. You may "protect" this check inside a conditional so that it only runs when you set a variable to `True` by hand.

6. (5 points) Make a Boolean mask array that is the same shape as the data array. A value of `True` indicates a good pixel and a value of `False` indicates a bad pixel. Initialize it to `True`. From this point forward, whenever you modify the mask array **or** data array, set all flagged pixels to 0 in the data array.

7. (5 points) Find the median value in each frame. Record it in the frame parameters array as the median background and subtract it from each frame. Plot the median background *vs.* frame number.

8. (20 points) Write a sigma-rejection routine. You may use code you wrote for an early homework assignment for this. However, you may not use code from the homework solution sets. For a given, arbitrary data set, find the median and standard deviation *from the median*.

Flag as bad anything more than, say, 5σ from the median. Then, iterate this again, using only your unflagged ("good") numbers to calculate the median and standard deviation in the second iteration. Return a Boolean array that is True for good pixels and False for bad pixels.

9. (5 points) Find bad pixels. This can be done with three nested loops: an outer loop that steps by jumps of 64 frames over the dataset and two inner loops that go over y and x. Extract each set of 64 pixels at a given y, x in each subarray set, call your sigma-rejection routine on each set, and insert the resulting 64-item Boolean array into your mask array. Plot the number of bad pixels *vs.* frame number. There is a way to do this step with one loop using the sigma-rejection routine described above. EXTRA CREDIT (10 points): Do it with no loops. The sigma-rejection routine will differ slightly from that described above.

10. (10 points) Find the approximate location of the center of the star in the first image. Fit a 2D Gaussian to a small region around that point and record the center position in the frame parameters array. Loop over all the images, using the fitted position of the previous frame as the guess position for the current frame. Plot each of the x and y position data sets *vs.* frame number on a single plot with two stacked panels, sharing the x axis.

11. (10 points) Mask the fainter companion star in each image. It will be at a constant vector offset from the program star. Calculate its offset from your program star using a mean or median image from the whole data set (to get a really good estimate), apply this offset to each of the centers you just calculated, and mask an appropriate `disk` for each frame. Print the y and x offsets between the two stars.

12. (20 points, see below) Write an aperture photometry routine that accepts data, a mask, radii, and a center for photometry; applies the mask to the sky before computing the sky level; does the photometry; and reports the results and number of bad pixels in the aperture.

    You may use your routine from the homework. You do not have to use a sub-image, since the Spitzer images are only 32×32. If you do, be careful that the sub-images are the same size and that, even if the annulus extends off the edge of the array, the sub-image just goes to the edge. There will be outlier frames when the spacecraft's reaction wheels adjust their spin rates, in which the star momentarily moves and recovers. The annulus may go off the frame then, and you don't want your routine to crash in that case by calculating an image slice that is off the edges of the array.

    The routine should:

    (a) (3 points) Use `disk` twice to make a mask for the sky annulus.

    (b) (2 points) Apply the bad-pixel mask for the sub-image to the sky annulus so that only the good pixels that are in the sky annulus are `True`.

    (c) (4 points) Calculate the average sky pixel in the annulus. Do not use any masked pixels in your sky estimate.

    (d) (1 point) Subtract this value from each pixel in the sub-image.

    (e) (2 points) Use `disk` to make a mask for the photometry aperture. Do not apply the bad-pixel mask!

   (f) (3 points) Calculate the total flux in the aperture.

   (g) (3 points) Calculate the total bad pixels in the aperture.

   (h) (2 points) Return a 1D array containing the stellar flux, the average sky, and the number of bad pixels in the aperture.

13. (5 points) Loop over the images and do aperture photometry. Store the aperture counts, average sky level, and number of bad pixels in the aperture in the frame parameters array.

14. (20 points) Check the photometric outputs by plotting just the photometry for the good frames (frames with no bad pixels in the aperture). Do you see the transit? Are there many outliers? If so, consider ignoring them. Or, you may flag them as bad by putting a -1 in the frame parameters entry for number of bad pixels in the aperture. Justify each elimination with a sentence fragment in your code, such as "very high background," "large star motion," etc. You would want to do these as classes of eliminations, of course, not frame-by-frame. You may not need to eliminate (m)any outliers, however (why?). Also, look at the sky, x and y centers, and number of bad pixels. Discuss in your paper what you see (if anything), and what you did about it. Do you have to do much, or are these frames already eliminated for some reason? Finally, inspect the two or three frames with the largest numbers of bad pixels. Describe what appears in each in the paper, and what you did about it.

15. (10 points) Bin the data into a few hundred bins and plot the bin averages. Adjust the number of bins until you like the plot. The transit is small, a few percent of the star's brightness, so you will have to zoom in. You should see a slight overall ramp in the data. Plot the data.

16. (4 points) Manually identify four frames in the unbinned data where the following happens:

    (a) first contact (start of transit),

    (b) second contact (planet fully on stellar disk),

    (c) third contact (start of egress), and

    (d) fourth contact (end of transit).

    Print these numbers.

17. (16 points) Fit a parabola to the good flux *vs.* time data outside of transit. Evaluate it at each frame time (all frames, not just good frames). Divide aperture photometry by these values for each frame to normalize it to this ramp model. It won't be perfect but it should improve it some. Plot the corrected data.

18. (10 points) Find a flat run of 100 or more normalized frames, perhaps a set just after the transit (avoid long-period ripples). Calculate the standard deviation of the photometry in this set of frames. What is your S/N per frame?

19. (10 points) Using your uncertainties, and taking averages of the good out-of-transit and in-transit data to find the best estimate of the normalized flux for each, calculate the ratio of planet cross-sectional area to stellar cross-sectional area (and, of course, its uncertainty). Ignore bad frames and frames from when the star's limb cuts through the planet. Print all of these values, including the averages.

20. (10 points) Look up the radius of the star (including uncertainty) in a refereed article and calculate the radius of the planet. Print it.

21. (10 points) Include a copy of your class log file in your handin. Print the Git log for your main project file into a text file named `project-`<username>`-prob21-gitlog.txt`. Include the PDF file for the paper in your handin.