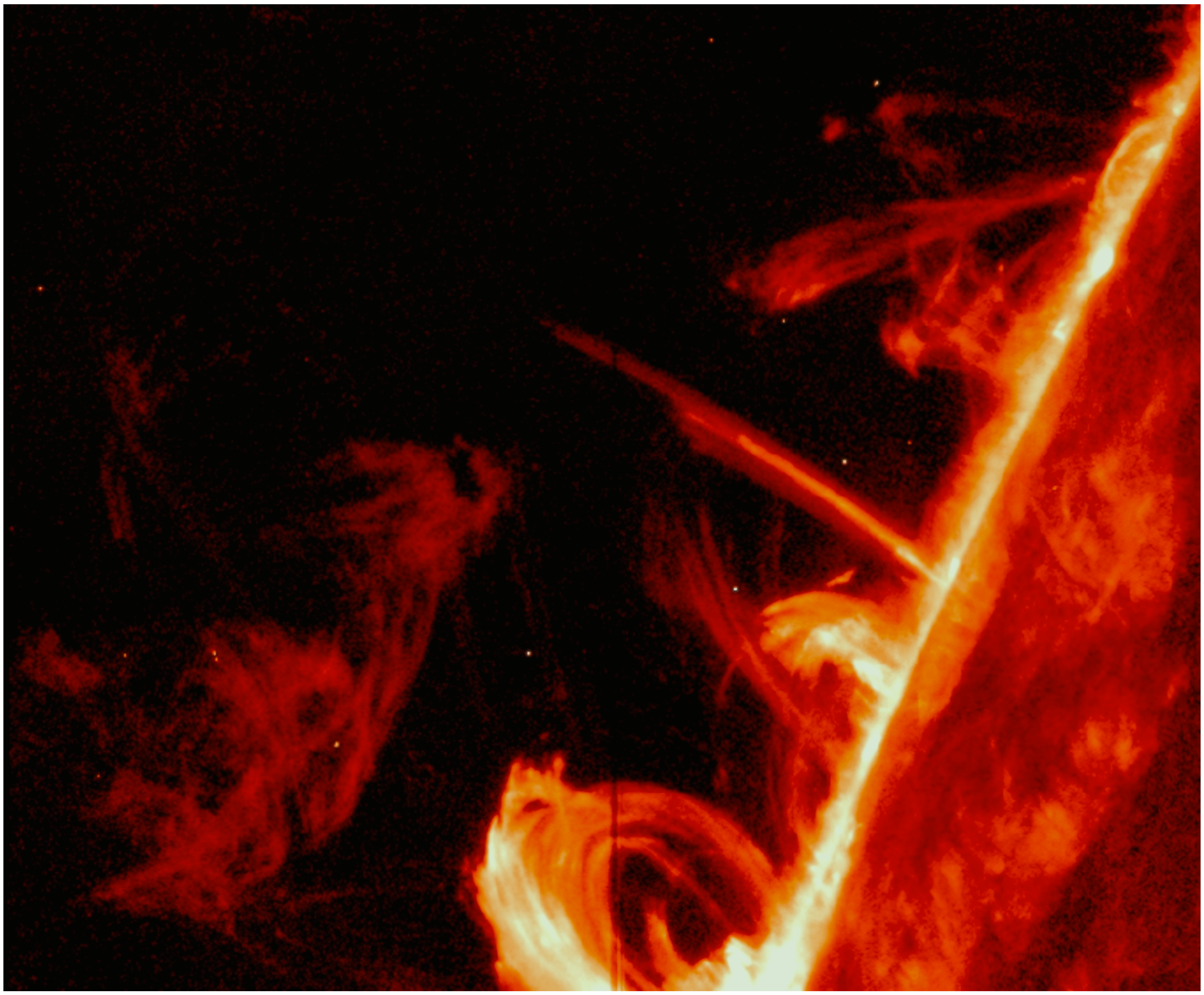
IRIS Darks Guide



2026

Table of Contents

Overview of IRIS Darks

The general goal of this pipeline and the person who uses it is to update the long-term dark model correction contained in *iris_dark_trend_fix.pro* file and send it to LMSAL to be used in the IRIS data processing pipeline. This file is used to provide the long-term dark correction to the overall IRIS dark model. The main IRIS dark model (*iris_make_dark.pro)* has been largely unchanged since launch and is not modified through this pipeline. The long-term dark model is updated several times a year to incorporate changes to the observed darks throughout the mission lifetime with a major update once a year.

Darks

Summary from this section is adapted from IRIS ITN14 and the IRIS Calibration Paper. Please read those original sources for a complete look at the more technical aspects. This is only meant to serve as a broad introduction to IRIS darks.

IRIS exposures with the shutter closed ("darks") still show a residual signal coming from two sources: an electronic pedestal introduced as a base level, and the dark current, which arises from the thermal energy within a CCD that is counted as signal independent of light actually falling on it. Regular (~monthly) observations of varying exposure times and binning schemes are taken by IRIS with the shutter closed to measure these contributions to the dark level.

A model for the shape and level of the dark was developed from these observations, which was created early in the mission and has not been changed since. This dark model is then used instead of actual darks, which reduces the need for extensive daily dark calibrations.

In addition to helping develop and refine the dark model, the dark observations provide a good dataset to observe the change in hot pixels. Hot pixels are individual pixels whose values are consistently above a defined threshold, and therefore falsely report an inflated value. The unbinned dark observations can be used to easily set a threshold above a uniform background. We can use this information to determine which pixels are consistently hot and track the change in the number of hot pixels over time.

Model

Initial calibration of the dark frame levels was discussed in De Pontieu et al. (2014), where the model for the total dark level, D, in read port j is given as:

$$D_j = P_j\left[T_{\mathrm{CEB}\,j}(t - \delta t_j)\right] + \mathrm{e}^{(a_j + b_j T_{\mathrm{CCD}\,j})} n_x n_y t_{\mathrm{int}} + \Delta D_j(x, n_x, n_y, t_{\mathrm{int}}).$$

Here, P_j is the pedestal level in read port j, which is a function of the camera electronics box (CEB) temperature, T_{CEBj}, time lagged by δ_{tj}. The second term gives the average dark current rate, which is the product of an exponential dependence on the CCD temperature, T_{CCDj}, the amount of on-chip summing, $n_x n_y$, and the time between CCD reads (i.e. t_{int}). The final term, ΔD, models the change in the shape of the dark in the wavelength (i.e. x) direction as t_{int} and summing are increased, from flat for tint ≈ 0 s and 1×1 summing, to roughly bilinear in x, rising first quickly and then more gradually away from the read-out point. The full amplitude of the pattern is $\Delta D \approx 10$ data numbers (DN) for the FUV ports with $n_x n_y = 32$.

In practice, D_j is computed for each port and added to the appropriate port of a basal dark. This basal dark was constructed by averaging 30 tint ≈ 0 s images, after the pedestal P_j, particle hits, and hot pixels were removed. It has not been updated since mission start.

After more on-orbit data had been accumulated, it became clear that the model above was incomplete. There were small differences in the predicted and observed dark levels for certain summing schemes. It was found to be due to an additional noise source dependent on n_x alone. There were also small differences, differing from port to port and varying quasi-periodically in time, when the average model dark level was compared with actual darks. A long-term trend function, L_j, was developed to attempt to model and predict these long-timescale variations.

The revised dark model, D_j, can thus be written as:

$$D'_j = D_j + c_j(n_x) + L_j(t),$$

The exact nature and cause of the long-term trend is uncertain, and thus the exact form of the trend has evolved over time, as more data become available to characterize it. It is currently modeled by the sum of two sinusoids with periods p_{Lj} and p_{Lj} /2, plus a weakly quadratic background. The variations on a timescale of p_{Lj} are typically ≤ ±1 DN (except for FUV port 3, where it is ±6 DN), on a background that has risen ≈ 10 DN (FUV) or 3 DN (NUV). Hints about the origin of the trend may be found in the facts that it is independent of summing and t_{int}, and that all the $p_{Lj} \approx 1$ year; hence some yearly and biennial orbital variations (possibly due to seasonal temperature changes) may be playing a role.

The full form of the long-term trend model Lj is currently:

$$L_j = d_j \sin\bigl(2\pi(t/p_{Lj} + \phi_{1j})\bigr) + e_j \sin\bigl(2\pi(2t/p_{Lj} + \phi_{2j})\bigr) + f_j + g_j t + h_j t^2,$$

Where d_j, e_j , f_j, g_j , h_j , p_{Lj} , ϕ_{1j} , and ϕ_{2j} are constants, fit for each port using the offsets, over time, between the predicted and the actual average (cleaned) dark levels.

In the IRIS Prep Pipeline

IRIS data is prepped via the file *iris_prep.pro.* This script includes a call to the script *iris_dark_prep.pro,* which gets the calculated dark (including long-term offsets) for a given observation, subtracts it from the provided data, and returns the dark subtracted data.

The dark model values are generated by the script *iris_make_dark.pro*, which will first read in the average base dark using *iris_prep_read_dark.pro*. This script pulls the average base dark calculated in 2013 (*dark_avg_fuv_20130925.dat* and *dark_avg_nuv_20130925.dat*) from the IRIS SSWDB at /iris/data. The script *iris_make_dark.pro* will then perform the calculations to add to the base dark to create a full dark for the given observing conditions.

Part of these calculations include calling *iris_ dark_trend_fix.pro* to generate the long-term model offsets. Using both the base model dark calculations and the additional long-term offset values, it will return the final full dark array to subtract from the prepped input file.

Again, this pipeline procedure documented here is only responsible for updating the offsets due to the long-term portion of the model from *iris_ dark_trend_fix.pro*. Modifications to the base dark contained in *iris_make_dark.pro* have not been made since 2015 and the average base dark files have not been updated since 2013.

There was some discussion in 2023 to create updated average base darks to add to the one called by *iris_prep_read_dark.pro*, but that was not moved forward with.

Longterm Model

This pipeline is only responsible for generating the long-term offset parameters produced by *iris_ dark_trend_fix.pro*. When it was noticed that there was a time dependent variation in the average dark value, this script was added to take into account this longer time period variation over the course of the mission.

The long-term model was found to be only time and port dependent, with no dependence on summing modes. (*Note: Analysis was started in 2025 to confirm that summing or binning did not have any impact on the long-term trend but was halted due to budgetary reasons. Initial analysis seemed to confirm this independence, but sufficient investigation was unable to be completed.)*

For the first part of the IRIS mission, the long-term model was a single continuous model with parameters that covered the entire mission length. In 2022, this was changed. Rather than having a single set of parameters to cover the entire mission, separate parameters are used depending on the time period. This has the advantage of better model fitting and removes modifications to the model in the later mission impacting data from early on in the mission.

The long-term model is broken into multiple segments, referred to by letter. Each segment covers a specific time frame of the mission and contains unique parameters for the model during that time. Once a model segment is completed, it is ‘locked in’ and those parameters for that segment should not be changed in the future. This improved model performance and overall data consistency.

Longterm Model Parameters

The long-term model originated with 7 parameters to define the offset for a given port at a given time. Before the model separation, additional parameter terms were added on a time dependent basis. To simplify things, this practice was restricted to model A (which covers the first half of the mission) and moving forward models contained just the original terms. This means that model A is more complicated and covers a longer time period than the other models. The model segments after A all take the same form and cover between 8-18 months’ worth of data.

The values for these parameters are recorded directly in the *iris_dark_trend_fix.pro* file. They are monitored, updated, and calculated from the *fit_ports_gui.py* file and stored and read into the GUI via in the *Model_X_Parameters.txt* files.

Parameters for Model A:

amp1: The amplitude of the approximately 1 year sine function.

amp2: The amplitude of the approximately ½-year sine function.

phi1: The phase of the approximately 1-year sine function in radians.

phi2: The phase of the approximately ½-year sine function in radians.

trend: The linear coefficient explaining the increase in the pedestal level.

quad: The quadratic coefficient explaining the increase in the pedestal level.

off: The intercept for the quadratic and linear function

qscale: The flattening of the linear and quadratic term after August 2017

bo_drop: The fractional drop in the offset (intercept term) due to the bake out on June 13-15, 2018

sc_amp: The amplification fraction in the in the sine function amplitudes due to the bake out on June 13-15, 2018.

ns_incr: The fractional increase in the offset (intercept term) due to non-standard IRIS operations from October 27th to December 15th, 2018.

Parameters for Model B, C, D, E, F, etc…

amp1: The amplitude of the approximately 1-year sine function.

amp2: The amplitude of the approximately ½-year sine function.

phi1: The phase of the approximately 1-year sine function in radians.

phi2: The phase of the approximately ½-year sine function in radians.

trend: The linear coefficient explaining the increase in the pedestal level.

quad: The quadratic coefficient explaining the increase in the pedestal level.

off: The intercept for the quadratic and linear function

Pipeline Overview

The processor runs this code, looks at it with the GUI, makes the correct adjustments to our model, then incorporates those adjustments into the iris_dark_trend.pro file so that it can be used in the data pipeline.

The main directory contains a c-shell script (*run_dark_checks.csh*), which runs a series of codes.

First, it finds the most recent run of dark scripts by querying the IRIS timeline page with the script *find_dark_runs_no_google.py*. Then it grabs the text of the timeline file for that day and searches for the simpleb and complexa OBSIDs.

Using the last set of observed dark times, the dark files are download from JSOC using the drms module (*get_dark_files.py*)

Currently at SAO, the archive for IRIS darks is stored on the following server and drive:

newtokyo2:/vol/vol2/alisdair

Which can be mounted to machines at a set location. For the examples in this guide and the latest version of the pipeline, this was mounted to a location on a machine called:

/Volumes/IRIS_Darks/

The code initially places the level1 dark files in the directory listed on the second line of the parameter file. This can be wherever the archive is located or mounted to. Currently that is listed at:

/Volumes/IRIS_Darks/IRIS_LEVEL1_DARKS/ YYYY/MM/

This is the standard location at SAO and is dictated by the parameter file. If this needs to be changed, it should be done in the parameter file. It will rename the files to adhere to previous standards.

Next, it converts the level1 files to level0 darks (*do_lev1to0_darks.pro*) for a given month and moves them to the level0 directory, located as the 3rd line in the parameter file. Currently for SAO that is listed at:

/Volumes/IRIS_Darks/opabina/level0/

Then the script checks for darks significantly affected by SAAs or transient particle hits using *find_contaminated_darks.pro*. It looks for too many 5 sigma hot pixels for a Gaussian distribution.

Next, it will download the temperature files for the day darks are observed plus +/- 1 day and format the output temperature file for IDL.

Once all the files are in place, the core of the pipeline will be running dark_trend.pro. This script will use the contam txt files to read in every good level0 simpleB dark from the mission so far and compare the calculated dark model for that observation WITHOUT the long-term fix to the observed dark file. The last step in the dark pedestal pipeline creates plots to compare the observed values to the modeled dark pedestal trend.

When the model does not align well with the taken darks, you may need to refit the model with new parameters. This is done once or twice a year typically. The GUI is used to find improved parameters and the new parameters are added in the correct locations.

Additionally, at the end of each year, the model parameters for the previous year are 'locked in' and a new segment of the model is created. See the Python Plotting and Model Refitting section for more info.

The hot_pixel_plot_wrapper is included at the end of the script. After the pedestal analysis finishes, the code runs the hot pixel analysis. The hot pixel analysis counts the number of Hot (5sgima) pixels in the level 1 IRIS observations.

The program works automatically because the plots output to the 'Z' window in IDL. Therefore, the job maybe cronned. Though it can be easier to troubleshoot if run in sections once a month instead.

There are some minor changes between the *iris_dark_trend_fix.pro* file used locally and the one used at LMSAL. Copy the latest *iris_dark_trend_fix_VXX.pro* file and update for the file to be sent to LMSAL. Update the iris_dark_trend_fix.pro for local use. Both need to be updated after each model change or refit.

Step by Step

All of the following steps are performed when calling the main file of *run_dark_checks.csh:*

1. Set up the user environment with a startup file to define SSWIDL environment and python environment (if needed). Ensure that all important files are included in your Python and IDL paths.
2. Read the parameter file to get the locations of the archive for IRIS darks (this file is also read by several other scripts directly)
3. Find the latest dark run with *find_dark_runs_no_google.py*. Looks back 15 days to find the matching OBSIDs on the IRIS timeline webpage. When it finds a match, it calls *get_dark_files.py* to utilize the python module drms (associated with Sunpy) to download the data to the location defined in the parameter file. Finally, updates the downloaded data into *processed_dark_months.txt* to avoid repeated downloads if things are rerun.
4. Converts the downloaded Level1 data into Level0 using the IDL script *do_lev1to0_darks*. First writes the fits files to a temporary *dummydir* location and then moves them to the location defined in the parameter file. Performed for both SimpleB and ComplexA files.
5. Find darks contaminated by particle hits, SAA, or other outliers with the IDL script *find_con_darks_no_thread.pro*. Generates a list of each file's pass or not and places a text file summary of these results at */contam_txt_files/*. Does this for both NUV and FUV. These files serve as the list of what dark data will be utilized in the model.
6. Downloads the temperature files for surrounding days of the darks using the python script *get_list_of_days.py*. The four days of temp files (in .txt and .fmt formats) are placed in */temps/*.
7. The main portion of the pipeline is calculating the dark model trend values using the script *dark_trend.pro*. This script reads all of the files from */contam_txt_files/* and creates a full array of each single dark file to be used.
8. Each file is processed with *check_ave_pixel_sub.pro*. This script will read in the date of the observation and the required temperature data files. It will then create a modeled dark without any long-term modification applied using a special version of *iris_make_dark.pro* called *iris_make_dark_no_longterm.pro*. It will subtract the modeled dark from the real dark file and return the average difference and sigma values for each file. Basically how much each image differs from the base model dark when we don't apply the long-term portion.
    a. If the keyword writefits is on, it will save a final fully modeled dark (including the long-term parameters) as a fits file at a set location hard coded in the script. These files aren't currently used for anything and remaking every

single one each time the pipeline runs was taking up a lot of time and space unnecessarily. If you ever need all of these files for whatever reason, just turn the keyword back on and it will start saving all of these files (for the full mission) each time.

9. Once all of the files are read and calculated (full mission, takes several hours at this point) the data is saved to a file called *alldark_ave_sig.sav*. There is also a more readable text file of these values saved to *current_pixel_averages.txt*.
10. Two separate versions of basically the same script are called. The script *plot_dark_trend.pro* is called directly from *dark_trend.pro*. The script *iris_make_dark_trend_plots.pro* is called from the main .csh file. Both these scripts take the data and generate .png plots along with .dat files. The *plot_dark_trend.pro* script generates the files *offset30nj.dat* and *offset30fh.dat* and the *_test.png* files. The *iris_make_dark_trend_plots.pro* script generates the files *offset30n.dat* and *offset30f.dat* along with the *_trend_plots.png* files.
    a. These used to be named differently and differed more but are basically repeats of each other at this point and can probably be combined. Only the files from *iris_make_dark_trend_plots.pro* are currently useful and the images from taking a screenshot of the GUI are typically much better than the plots generated from these scripts.
11. The hot pixel portion of the pipeline is called via *hot_pixel_plot_wrapper.pro*. This will look for the running .sav files located in */Hot_pixel_sav_files/5sigma_cutoff/*. There will be two for each port, both FUV and NUV. It will read and update each of these files with *iris_heat_map2.pro*, which goes through all of the dark calibration images of 0s and 30s exposure times, analyzes them for dark pixel count rates. It will then create plots of these saved as .png files with *hot_pixel_plot_wrapper.pro*.
    a. *These files are stored here for SAO which are then grabbed by LMSAL to be hosted here.*
12. The model fit GUI can be launched by running the latest version of *fit_ports_gui.py*. This will read all of the observed minus base dark values saved in *offset30n.dat* and *offset30f.dat* and plot them along with the model trend line. The model parameters are pulled for the *Model_Parameters.txt* files for each model segment. You can use the GUI to refit the model and print out new parameters to update in the *Model_Parameters.txt* to see in the GUI and eventually add to the new version of *iris_ dark_trend_fix.pro*.
13. Report and send out any new files to the IRIS_Calib mailing list. Typically a high-level summary is provided along with the current fits and a PNG of the most recent model fit from the GUI.
    a. Attach an updated *iris_ dark_trend_fix_VXX.pro* if the parameters changed

File Structure

The IRIS Dark Pipeline code is contained in a directory called */IRIS_Darks/*. It contains the following sub-directories.

- */calc_trend_darks/*
- */contam_txt_files/*
- */dummydir/*
- */IDL_Code/*
- */IRIS_dark_and_hot_pixel/*
- */OUTDATED/*
- */temps/*

/calc_trend_darks/ is where the processing code lives. It is also where the GUI directory is located, */python_fit_ports/*.

/contam_txt_files/ houses the txt files listing which fits files are good to use.

/dummydir/ is used to temporarily store level0 files before sending them to the archive

The majority of IDL code used in the pipeline is located in the */IDL_Code/* directory. However, most of the higher-level scripts instead live in their respective directories.

/IRIS_dark_and_hot_pixel/ has all of the files and scripts related to the Hot Pixel process

/temps/ stores the temperature files and the code used to download those files

Pipeline Requirements

- **Pipeline Files**: All of the files needed for running the pipeline. A version was created in early 2026 which can be found here. A zip file of this version also exists.
- **Python**: An installation of Python3. Will need the ability to download and install packages through something like pip. Required python packages to install include scipy, urllib, numpy, requests, drms, datetime, ssl, matplotlib, and tkinter.
- **IDL:** A working installation and license for IDL to run the .pro files.
- **SSWIDL Distribution:** An installation of solarsoft to launch sswidl
- **SSW and SSWDB:** A ssw installation including ssw/iris/ and sswdb/iris.
- **Archive Mounted:** A mounted version of the archive of IRIS dark files. This is updated each time with the new files and you will need all of them going back to 2014 to properly run the pipeline. At SAO that drive is mounted from:
  newtokyo2:/vol/vol2/alisdair

Setting Up the Environment

For the pipeline to run, you will need to make sure that the appropriate paths are setup in both the Python and IDL environments. This will only need to be setup once and then the pipeline should run smoothly in all future runs once you get it right.

This is currently done by setting up the required environments in the *.cshrc.user* file, which is sourced at the beginning of *run_dark_checks.csh*.

Python

For Python the following lines are added to make sure the installed Python is called:

```
alias python /usr/local/bin/python3
alias pip3 /usr/local/bin/pip3.14
```

All required packages can be installed via pip. At SAO, they should default to installing to your /home/ directory and should be already in your Pythonpath.

You can check what is included in your Pythonpath with:

```
import sys
print(sys.path)
```

If there are still issues with your Pythonpath finding the correct modules, you can add the following line to the *run_dark_checks.csh* file:

```
setenv PYTHONPATH ${PYTHONPATH}:/path/to/your/modules
```

Note: You can look at each Python script and see what is being imported and then confirm that you have each package installed. Or you can just run each script until it fails and then use pip to install whichever package it says you need until it runs. There aren't many required packages (~10) so this isn't that painful a way to do it.

IDL

For IDL, the paths to be included should be defined in your *.idl_startup* file. This file is defined in the *.cshrc.user* file with a line like:

```
setenv IDL_STARTUP $HOME /.idl_startup
```

The startup file should have the following lines to include the directories in your path:

```
!Path = expand_path('/PATH_to_IRIS_Darks/IRIS_Darks/IDL_Code') + ':'+!PATH
!Path = expand_path(' PATH_to_IRIS_Darks /IRIS_Darks/calc_trend_darks') + ':'+!PATH
!Path = expand_path('/ PATH_to_IRIS_Darks /IRIS_Darks/IRIS_dark_and_hot_pixel') + ':'+!PATH
```

Note: By placing the directories at the start of the !PATH, it will default to calling any functions or procedures from those locations before ones later in the path, such as the /ssw/ distribution. This is helpful to ensure the correct version of everything is being called but be aware of what order you are putting thing in for your path and if there are any duplicate routines that may cause an issue if you call one version over the other. This shouldn't be an issue with these files, but I default to having the dark code first just to be sure.

IRIS Dark Pipeline

Below is a summary of each step of the pipeline process and the main scripts used for each.

Run Dark Checks

The c-shell file is a wrapper combining the IDL and python portions of the program. This wrapper is all you need to run for a simple run of IRIS dark calibrations.

It can be cronned, run manually, or run in segments by copy and pasting the lines into terminal. The latter is obviously the least efficient but helps with troubleshooting if there are issues or you are just starting out running things.

In order to run the script from your machine you will need to do a few things:

- Make this script executable by typing chmod a+x run_dark_checks.csh.
- Update the HOME variable at the top of the directory to be your HOME directory.

It will get the locations and email address from the parameter file, look to find new darks to download, and if there are new darks found run the rest of the pipeline.

You will need to update the parameter file to have the correct info for:

1. Email address used to download JSOC files
2. Location of the level1 dark archive to originally download files
3. Location of the level0 dark archive to copy files to once that conversion is done

Downloading Dark Files

To find files to download the script *find_dark_runs_no_google.py* is used. This version of find_dark_runs uses the iris timeline files on the IRIS website to search for the correct OBSIDs. The program searches archived timeline files and sends the day to *get_dark_files.py* and the year,month to c-shell script which the c-shell script uses to pass to IDL functions.

NOTE: *This used to be done with a script find_dark_runs.py which found the observed darks by querying the google calibration-as-run calendar for IRIS dark runs in the last 25 days. It would then grab the text of the timeline file for that day and search for the simpleb and complexa OBSIDs (find_dark_runs.py, N.B. add a proxy server to your environment in find_dark_runs.py if your institution does not allow access to the Google Calendar API). The No Google version became the default to avoid needing any updates or changes to the API. There is additional info in the README file about this process and those version are stored in the /OUTDATED/ directory.*

The actual downloading of the files is done by *get_dark_files.py*, which is called when new darks are found to be downloaded. This program is the work horse for obtaining the darks from JSOC. It takes the time and whether you are seeking complexA or simpleB darks (find_dark_runs asks for both).

The program then gets the timeline text from Lockheed, which it parses to find start and stop times for OBSIDs corresponding to the selected dark. Next, it uses the time frame found in the timeline to query JSOC iris level1 using the drms module in Python.

Once the JSOC query finishes, the program downloads the files to the first location in the parameter file and renames them according to a previous file naming convention for convince.

Level1 to Level0 Conversion

The script will the call *do_level1to0_darks.pro,* which utilizes *iris_lev120_darks.pro* which in turn calls *iris_leve120.pro* from ssw distribution. This is a legacy program, which uses sswidl libraries.

You may call it by the following commands in IDL:

```
do_lev1to0_darks,MM,YYYY,/simpleB,'0','dummydir/'
do_lev1to0_darks,MM,YYYY,/complexA,'0','dummydir/'
```

The program assumes the darks are located in line 2 of parameter file in addition to the YYYY/MM subdirectory passed to the program.

The level 1 to level 0 conversion is small and mostly rotates the image using the sswidl function iris_lev120_darks.

Currently, the program is set to output to a dummy directory because saving to a network directory from IDL can cause hangs. Therefore, it creates the files locally and then immediately move them in the script to the output directory specified in line 3 of the parameter file.

Contaminated Files

This work is performed by the script *find_con_darks_no_thread.pro*, which is an IDL program which finds IRIS darks contaminated by SAA or CMEs. (*It used to have a more complicated multi-threading structure that caused issues and was more complicated than it needed to be, so a non-threading version was used moving forward.)*

The main processing and analysis of each file is done by *check_sig_level.pro*.

The program runs by taking an array of dates, the dark file types, and channel (NUV or FUV). You can specify the day in either 1 or 2 digits and the year in 2 or 4 digits. If only one year is specified then all months in a month array are assumed for that year.

The program finds contaminated darks by breaking each dark image into its four ports. Then it finds the number of pixels more than 5 sigma away from the mean.

It then sums the total number of pixels 5 sigma away from the mean and normalizes that number by the integration time (if the integration time is greater than 1). Then it uses the pixel fraction greater than 5 sigma to find images affected by SAA.

If you assume a Gaussian distribution, we expect that fraction to be 6.E-5, so we assume the 5 sigma Gaussian fraction to reject images with fraction higher than the Gaussian value.

Finally, the program writes the file name, start time of integration, whether it passed (1 is passed 0 is failed), total pixels above the 5-sigma level normalized by exposure time, and the integration time to a file in */contam_txt_files/*. The output file is formatted *NUV(or FUV)_YYYY_MM.txt*.

Temperature Files

The script *get_list_of_day.py* is in the */temps/* directory. It is a simple python script, which grabs the temperature information from Lockheed.

It uses the files in */contam_txt_files/* to find days darks were observed.

Then the program checks to make the temperature information does not exist locally, and if it does not exist it downloads it.

Finally, it formats the file to just the important temperatures so IDL calls it easily.

Dark Trend

The program *dark_trend.pro* uses the SAA free darks found previously to look in the level0 directory for darks.

Again, the program again assumes the level0 data is located on the given network drive, but that can be changed by setting the sdir keyword.

After grouping all dark observations, it loops over all darks computing the average and sigma values over the whole chip in the program *check_ave_pixel_sub,* which is the workhorse of the program. This program's primary function is subtracting the current dark model from the set of darks passed to the program.

The syntax for *check_ave_pixel_sub* is as follows:

>check_ave_pixel_sub,file,endfile,timfile,avepix,sigpix,temps,levels,writefile=writefile

All parameters are set by default in dark_trend.pro, but for clarities sake they will be described here.

- **file** is the full path to a given dark observation (string),
- **endfile** is a formatted file name which we will pass to plot_dark_trend (string),
- **timfile** is the file formatted for reading the Lockheed IRIS temperature data and is computed in the program (string),
- **avepix** is the average model subtracted dark pixel value returned by the program (4D vectory),
- **sigpix** is the 1 sigma variation in the average value (4D vector),
- **temps** is an array of temperatures used to derive the dark model (3x6 array),
- **levels** is an array of dark pedestal values computed from the long term trend (4D vector),
- **writefile** is keyword which writes out the full dark-model dark-long term trend dark to a file. (Default is set to OFF, since these files aren't super useful currently.)

Upon completion of finding the averages the program calls *plot_dark_trend* with the after-pixel values and the observation times.

Finally, the program saves the information to .sav and .txt files (alldark_ave_sig.sav and *current_pixel_averages.txt*). This is done by both *plot_dark_trend.pro* (called from *dark_trend.pro)* and *iris_make_dark_trend_plots.pro,* which is called from the .csh file.

NOTE: You will need to update check_ave_pixel_sub.pro with the correct output location if you want to write the fits files out. This is currently set to a location at SAO and the write function is not the default. But if you want to program to output the fully modeled dark fits files (including the long-term fix), you will need to modify the code to have both the keyword on and verify your set output location in check_ave_pixel_sub.pro.

Python Plotting and Model Refitting

After the code is completed, the user should look at the current agreement of the model to the observed darks. If large disagreement is seen, the model will need to be updated. In this case, large disagreement is defined as more than 2 sigmas off the overall model. However, the true definition of when to refit or not is really up to the pipeline manager.

The python GUI script is used to visualize the model agreement and update the parameters to improve the fit. These updated parameters can be printed out by the GUI and added to the *iris_dark_trend_fix.pro* script to be integrated into our pipeline and the *iris_dark_trend_fix_VXX.pro* script to be sent to the IRIS pipeline.

There are two types of model refits: Basic and Segment Separation. Both are covered below. A basic refit will be performed is poor alignment is seen for a few months in a row. The GUI will be used to improve the current parameters, files will be updated, and a new version of iris_dark_fix.pro will be sent out to be used moving forward. The addition of a new model segment is more complicated. This is done once a year to lock in the parameters covering the last years' worth of data.

Python GUI

A python GUI for fitting the long-term pedestal trend of the IRIS CCDs was created in the file *fit_ports_gui.py.* It is updated each time a model segment is created, so there will also be a version number associated with it. *Note: versions before V7 were written to run with Python2, which is long since depreciated. Use version 7+ moving forward with Python3*.

The program uses the sav files (*offset30n.dat* and *offset30f.dat*) created by *iris_make_dark_trend_plots.pro.*

The python GUI imports the following modules.

- *matplotlib*
- *numpy*
- *sys*
- *datetime*
- *tkinter*
- *scipy*
- *fancy_plot (Module written as part of the pipeline and included as fancy_plot.py)*

To run the program type the following command in a terminal window:

> *python fit_ports_gui.py* (or correct version number)

After typing the command you will be greeted with a GUI containing two plots.

The left and right plots contain the FUV and NUV, respectively, difference between the model pedestal and the measured dark pedestal as a function of time. Both the FUV and NUV CCDs contain four ports for rapid read out of the CCD:

port 1 = red circle, port 2 = blue square, port 3 = teal diamond, and port 4 = black triangle

The plot also contains a model for the pedestal's evolution with the color corresponding to the port number. The model pedestal parameters for CCD type and port are below their respective plots in the med row. The model is set to plot a cycle forward as well.

Above and below the med row for each parameter is the maximum and minimum range to search for new parameters. The parameter range may be set automatically by using the % Range text box in the bottom right of the gui.

Fortunately, deviations from the trend do not happen to all ports at the same time. Therefore, you are sometimes only refitting a few ports, which is why the GUI allow you to select the ports you want to refit.

Furthermore, the parameters not all parameters need refit every recalibration, which is why the GUI allows you to dynamic freeze some parameters. In the example below I only wanted to refit FUV port 3 for the Amplitude of the sin function.

Selecting port and freezing parameters example below:

http://www.youtube.com/watch?v=v3VH7uBjTJw

In the above example using an infinite range worked well. Frequently, using an unrestricted range causes the program to find nonoptimal minimums. Therefore, I included a range box in the lower right. The range box sets the minimum and maximum allowed value for all thawed parameters. Of course this example did not benefit from a restricted range, but it is an outlier not the norm.

Setting parameter range example below:

http://www.youtube.com/watch?v=1Nu14eoA0ww

Finally, you will want to efficiently save new parameters. The GUI has the print button for that.

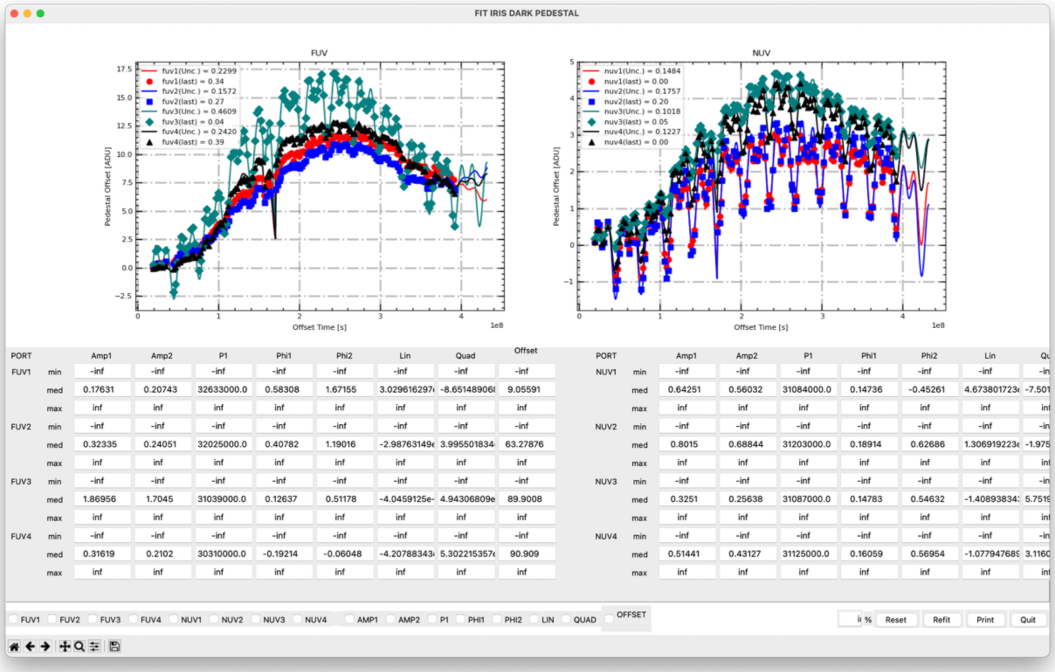
The print button prints the new parameter values in a format for the *iris_trend_fix* program, as well as the initial_parameters.txt file.

Printing new parameters example below:

http://www.youtube.com/watch?v=jC0AbvZRth8

Model parameters as read in by the GUI are stored in the files *initial_parameters.txt*. There is a version of these parameters for each model segment. To update these files, just copy the output of the GUI print function. The next time the GUI is launched it will read from the new file. The names of these files are defined in the GUI code.

Important Note: The code will only optimize parameters for the latest segment of the model. The parameters without a letter prefix will be optimized back to the point of last separation. You can 'trick' the model into using more data to create a model by temporarily moving the last separation date backwards.



PORT		Amp1	Amp2	P1	Phi1	Phi2	Lin	Quad	Offset
FUV1	min	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
	med	0.17631	0.20743	32633000.0	0.58308	1.67155	3.029616297	-8.65148906	9.05591
	max	inf	inf	inf	inf	inf	inf	inf	inf
FUV2	min	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
	med	0.32335	0.24051	32025000.0	0.40782	1.19016	-2.98763149	3.995501834	63.27876
	max	inf	inf	inf	inf	inf	inf	inf	inf
FUV3	min	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
	med	1.86956	1.7045	31039000.0	0.12637	0.51178	-4.0459125e-	4.94306809e	89.9008
	max	inf	inf	inf	inf	inf	inf	inf	inf
FUV4	min	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
	med	0.31619	0.2102	30310000.0	-0.19214	-0.06048	-4.20788343	5.302215357	90.909
	max	inf	inf	inf	inf	inf	inf	inf	inf

PORT		Amp1	Amp2	P1	Phi1	Phi2	Lin	Qu
NUV1	min	-inf	-inf	-inf	-inf	-inf	-inf	-in
	med	0.64251	0.56032	31084000.0	0.14736	-0.45261	4.673801723	-7.501
	max	inf	inf	inf	inf	inf	inf	inf
NUV2	min	-inf	-inf	-inf	-inf	-inf	-inf	-in
	med	0.8015	0.68844	31203000.0	0.18914	0.62686	1.306919223	-1.975
	max	inf	inf	inf	inf	inf	inf	inf
NUV3	min	-inf	-inf	-inf	-inf	-inf	-inf	-in
	med	0.3251	0.25638	31087000.0	0.14783	0.54632	-1.40893834	5.751
	max	inf	inf	inf	inf	inf	inf	inf
NUV4	min	-inf	-inf	-inf	-inf	-inf	-inf	-in
	med	0.51441	0.43127	31125000.0	0.16059	0.56954	-1.07794768	3.116
	max	inf	inf	inf	inf	inf	inf	inf

Model Refitting (Basic)

Performing a basic model refit (without creating a new model segment) is pretty straightforward. You will use the GUI to refit the parameters, print them out, and then update them in the both the *iris_dark_trend_fix.pro* file (for being used in our pipeline) and *iris_dark_trend_fix_VXX.pro* (format used in the IRIS pipeline to be sent out)

The process for using the GUI is explained above.

Be aware of what time period you are refitting over. While there is no exact formula for how far to go back, you want to avoid overfitting to small datasets when possible. So try to make sure that the period being refit covers at least two cycles of IRIS darks. You can edit the time of the latest entry in the Parameters Read section of the code to start as far back as you would like to perform the refit over.

So if it is December 2025, model F is being used, and I see a large disagreement that I want to refit the model over. I would:

a. Set the self.seperation_F variable to go to back to the earliest data I want to build my new model from.
    i. It will default to the start of Model F on 11/01/2024, but if I want to build a model from data covering further back than that, I can temporarily change that time to be earlier to refit with that data as well.
    ii. So if I change this go back to 01/01/2024, I will be building a model with all of the data from 2024 and 2025.
b. Launch the GUI and look at the fit for the model over the time period I am concerned about.
c. Use the GUI to improve the parameter fits in each port until I am satisfied with it.
d. Print out the new parameters.
e. Update the new parameters in the appropriate *initial_parameters.txt file* and in the new version of the *iris_dark_trend_fix.pro* files.
f. Return the separation time (self.seperation_F) to the correct time if changed.
g. Send the *iris_dark_trend_fix_VXX.pro* file to the calibration team for ingestion into the pipeline.

Model Refitting (Model Separation)

Starting in 2022, the model moved from a continues model to one with distinct segments. This was done for simplicity and to ensure proper data processing. The model is now refit every so often (annually) into distinct segments, referred to as Model (Letter), such as Model A and Model B.

The idea is that for a set period of time, a distinct set of parameters will be used. The actual separating process is usually performed towards the start of a new year, often in March or April to get some data after the end of the newly separated model segment. For example, Model F (covering 2024/01/01 through 2025) would be locked in around March 2026 and would be calculated using data going at least 1 cycle back (so to late 2023) and a few months forward (so using the first few sets from 2026). This is done to avoid overweighting any point from a single year and improve the model fit.

Below are the steps needed when a new model segment is to be created, where we are looking to lock in the parameters for Model F (covering 2024/11/01 through 2025) and add a new model segment for Model G covering 2026 moving forward:

1. Determine the time where the model will be separated and the new model will begin. When will we switch from Model F to Model G?

    i. This needs to be converted into the python format used by the GUI code. Record this time but don't modify any code yet.
    ii. IDL (the format the data in the .sav file is stored in) utilizes anytim with a Jan 1st 1979 start time
    iii. Data times in the python GUI are generated with a Jan 1st 1979 start in mind
    iv. Python utilizes a Jan 1st 1970 start time, so a conversion factor is needed between the two
    v. Conversion factor in the code: *self.convert = 284014800.0*
    vi. To convert manually in Python, use: datetime.fromtimestamp(284014800+X).strftime("%m/%d/%Y %I:%M") to find the correct start time (you can use previous entries to test)
    vii. Start with the previous model separation time as X and iterate until correct time is found.
    viii. In this example, Model G would want to start right at 2026/01/01, which corresponds to 1.4832288e+09.
2. In the GUI code, set the separation time to go back to the first set of data you want to use to optimize your parameters.
    i. Don't delete the actual separation time, we are only doing this temporarily to 'trick' the GUI into refitting with a bigger dataset. You can just comment out the real time and put in the temporary one.
    ii. For model F, temporarily setting the self.seperation_F value to something like 1.42e+09 will means it looks back to data from the end of 2023 through now.
3. Run a normal model refit to optimize the fit FOR THE TIME PERIOD COVERED BY THE NEW MODEL!
    i. At this step, you are only looking to create a good fit for your latest segment but will be using data from outside the range actually covered by the model
    ii. If we perform a refit in the GUI, the model on the plot will be updated from the end of 2023 all the way to the current time. But we only care about the fit of the segment over the data actually in Model F (so 2024/11/01 through the end of 2025)
    iii. You can go back and change the start time to get better results if needed to include more or less data.
4. Print the results and update parameters in the parameter file for the current model segment. Update these parameters in the appropriate place in the *iris_dark_trend_fix.pro* file.
    i. Congrats! Model F is now locked in with optimized parameters. Never change them again.

5. Once the current model is locked in with optimized parameters, now we want to add a new model segment (Model G) to our code. This will start with the same optimized parameters you just created and can be updated throughout the year as we get more 2026 that Model G will be covering.
6. Copy the *fit_ports_gui_VX.py* file and create *fit_ports_gui_V(X+1).py.* It can technically be done in the same version, but I have been creating a new version each time the model is separated.
7. Copy the current model parameter file and rename for the new model segment (Copy Model_Parameters_F.txt and then rename new copy Model_Parameters_G.txt').
8. Update the new version of the GUI code:
   i. This work is pretty straight forward. Basically all you need to do is copy the previous format to shift the model segment forward. I have noted each section where changes are needed
      i. In the **'Parameter Read In Section'**, you need to create a new section to read in parameters and update the previous one. Copy the previous block and paste a new one below. Change references to the previous letter to the new one including the new parameters.txt file. In the previous one, create a new dictionary with the correct letter and change the current_dict variables in the section to the LETTER_dict. See previous versions and match. The code expects the latest model to be using current_dict for the parameters.
      ii. In the '**Timing Section**', create a new self.seperation_X variable for the start of the new model. (From step 2) Follow previous format
      iii. In the '**Model Parameters Section'**, you will need to create a new set of parameters to be read in. Following the previous format, create a new set of parameters for the PREVIOUS model version following the correct letter format. (So in this example copy the parameters E section and make a parameters F section) The script is reading in the CURRENT parameters (with no letter) without needing this step, but you need to hard code in the parameters for the previous version. Just copy the previous and change the parameter names accordingly.
      iv. In the '**Model Trend Section'**, you will need to both create a new model and update the previous one to use the renamed parameters. First, copy the previous model and create a new entry. In the new entry, change all letter variables to the next one forward. (NOTE: cport variable is not part of the letter scheme and stays the same. c is also a constant, so be sure you are only updating the parameters). In the

previous model, change parameter names to include letter prefix (amp1 --> F_amp1)

9. The GUI will now have locked in parameters for Model F (covering 2024/11/01 through 2025) and will be plotting Model G with those same parameter values for now. It will be able to perform refits from the start of Model G (2026/01/01) to the current point without impacting model F at all.
    i. If model G starts to get misaligned in say June and you go to refit, you will get a bad result as it will only be using data back to the separation point. To get a better refit, add more data to the refit by temporary moving the point of seperation_G back to incorporate more data.
10. Update the *iris_dark_trend_fix_VXX.pro* file for the new version. This is the file that will actually be used in processing the data.
    i. You should have already added the optimized parameters for the previous model (Model F here). Doing that first makes copying easier.
    ii. Copy block of text that described the parameters for the previous model. Paste below in the same format. Update variables with new letter name. IT should start by using the same parameter values as the previous model. Copy from parameter text file. Update start time for this new segment to match model start time.
    iii. Create a new model entry for the new model by copying and pasting the previous model. Update the parameters to use the new model (change the letters) and verify your model is of the same form (should be unless there were major changes in one of the refits)
    iv. Create a new array to determine the time that the new model will use. This works by creating arrays with 0s for times outside the model segment and 1 for times within the model segment. Then by multiplying the calculated model offsets by these arrays, we are only getting data for the correct model for the correct time. Create an array called Model_X_time with the correct letter. Modify the previous model time to end at the start of the new model. The new model time should start from that time. Follow previous formats. Add a new + (Model_X_offsets * Model_X_time) to the final offsets equation. Again, follow previous format.
    v. Double check that all parameter names and variables have been updated correctly.
    vi. Run test to ensure update was properly done. To do this, use the 'iris_dark_trend_fix.pro' files. You can get the offset values for a given time with each version of the file. Confirm that you are getting the expected

results. Try running in each version right before and right after the model separation to confirm the difference.

11. You will also need to create a new version of the the *iris_dark_trend_fix.pro* used in our code. Just copy and paste the new and modified segments and save.
12. Send out new model with instructions for reprocessing.

Note: If better alignment is found a few months off of the end of the year in either direction, that is ok. You want to try to minimize discontinuities, so sometimes moving forward or back a month produces a better result. That is why Model F starts in November rather than at the start of the new year. It produced a better alignment that way.

Hot Pixels

The IRIS CCD has many pixels that are noticeably 'hot', or consistently brighter than the surrounding pixels. Looking at the data, it seems that there are an increasing number of bright pixels. We set out to determine how many pixels are affected, how serious the damage of those pixels is, how the number of affected pixels changes with time, and finally whether the pixels can be corrected. Several codes have been developed for this end. The following programs are utilized, which are either in the hot pixel directory or the IDL code directory, both of which must be in your IDL path.

- *do_lev1to0_darks.pro*
- *iris_lev120_darks.pro*
- *hot_pixel_plot_wrapper.pro*
- *iris_heat_map2.pro*
- *get_hot_pix2.pro*
- *median_data_by_month.pro*
- *median_index_by_month.pro*
- *despike_data.pro*
- *get_uniform_struct.pro*
- *hot_pixel_trend_plots.pro*

The data used for all calculations comes from the simpleB dark calibrations performed by IRIS on a ~monthly basis. Only the 0s and 30s exposure times are considered here. To stay consistent with previous dark analysis, we make use of only the level0 data, which is back calculated from the level1 data.

The program *hot_pixel_plot_wrapper.pro* will automatically call all of the other programs you need. If you are running it by default, you don't need to call any keywords. It is setup to run correctly from *run_dark_checks.csh*.

IRIS_heat_map2.pro updates the existing save files that have information about the number of hot pixels for each exposure time, both as a raw number (hot_pix_by_month_30s) and a percent (norm_hot_pix_by_month_30s). It also contains an array with the median data values by month for each exposure (median_data_by_month_30s).

This will call *get_hot_pix2.pro*, which is the work horse that actually calculates the number of times each pixel is hot. It returns an array where each pixel gives a total count for times in the given calibration sequence the pixel at that location was considered 'hot'. If the value at that pixel is 0, that pixel's value was always within the sigma provided. If the pixel value is 1 or 2, it was probably due to a cosmic ray hit. If the value is >2, we define it as a hot pixel. Note that the highest value can change by month. For the dark calibration sequence, there are usually ~11 images of each exposure time, but time constraints can vary that number, so it may be better to think of the value as a percent. i.e. if the value is 3 and there were 11 total images, that pixel was hot 27% of the time.

The program *get_uniform_struct.pro* is important. The old level1 files and the new ones have different keywords that don't play well together. I made a new structure using only keywords I use that avoids the problem. If you decide to change things in a way that needs different keywords, you will have to update this file so those keywords are saved.

The script *hot_pixel_trend_plots.pro*, restores the files with the naming convention *NEW_port1_FUV_hot_pixel_counts.sav'* (generated with IRIS_heat_map2.pro) and then plots them for each percent cutoff, meaning the percentage of time a pixel must be flagged in a given month to be called hot. The decided upon cutoffs we are tracking are 10%, 50%, and 90%.

Once the plots are made, they need to be moved to a folder reachable to the outside world. This is because Lockheed will take the plots and put them on the iris webpage on the 16th of every month.

The plots must be put into the directory (at SAO)

kurasuta-0-0:/var/www/projects/IRIS/public_html/IRIS_plots/

To be hosted at:

http://helio.cfa.harvard.edu/IRIS/IRIS_plots/

If at a different institution or this changes at SAO, you will need to determine where to host these files for IRIS to grab.

Once Lockheed has put them on the IRIS health and safety page, you can view them here: http://iris.lmsal.com/health-safety/longtermtrending/in_other.html

Important Note: The code for *hot_pixel_plot_wrapper.pro* should correctly handle creating a year list to send to *iris_heat_map2.pro*. However, there have been some issues with this not working correctly before. If you experience issues with the plots not updating correctly, you may want to manually add the newest year to the year list at the top of *iris_heat_map2.pro* to avoid any potential issues.

Reporting

After each reprocessing, a report is sent to the IRIS_Calib list. The report is a simple high-level overview of the results from the month, when a model refit may need to be performed, a summary of current fit values, and plots of the most recent data showing alignment. Below is a sample report:

==
I have processed the October 2025 IRIS darks. Alignment looks pretty good, especially for heading into eclipse season where we typically see larger than normal deviations. There is no need to perform a model refit for at least another month or two as we look to things to stabilize during eclipse season.
I have attached a plot showing the model fit. Below are the current fit results:
FUV1: 0.05 with a 0.2300 sigma
FUV2: 0.33 with a 0.1514 sigma
FUV3: 0.46 with a 0.4655 sigma
FUV4: 0.30 with a 0.2417 sigma
NUV1: 0.16 with a 0.1498 sigma
NUV2: 0.12 with a 0.1761 sigma
NUV3: 0.07 with a 0.1024 sigma
NUV4: 0.14 with a 0.1240 sigma
==
Whenever the model is refit and a new version of *iris_dark_trend_fix* is created, it should be sent along with the report to the IRIS_Calib list. They will acknowledge the new file and integrate the new version with the improved parameters into the main IRIS pipeline.

You should also inform the team how far back data should be reprocessed with this data. This will depend on how poorly aligned the model was before it was refit. Current standard practice is to not let the model diverge far enough before performing a refit that the data becomes truly bad. That way the new model can be brought online without the immediate need to reprocess the latest data.

Current practice is to only perform data reprocessing once a new version of the model is created covering a new period of locked in data. That way it can process all of the data from the year with the best model parameters that will not be changed in the future. This minimizes the need for reprocessing due to darks. When a new version of *iris_dark_trend_fix_VXX.pro* is created between model segments, it is ingested into the pipeline and used moving forward for a better refit, but no reprocessing is performed.

Notes

- In general, locations should only be hard coded as the default option if a different location is not provided for a script. Since in practice the pipeline is setup and not modified frequently, it is possible some locations are hard coded without noticing it. If you run into issues running the pipeline, verifying correct locations are being passed to the scripts and updating the default locations is a good place to start.
- Work was put into cleaning up the IRIS Dark pipeline after multiple years of being passed from person to person. The current state of the pipeline removed duplicate routines and unnecessary files and includes important changes added to the pipeline. This version was preserved in early 2026, along with a zip of a version from mid-2024. The 2026 version contains important changes and should be the default. Other versions are only preserved for recording keeping purposes and will not function properly in their current state.
- The version that was taken over in late 2019 (excluding some modifications made during the time at LMSAL in-between) can be found in the original form at: https://github.com/jprchlik/find_contaminated_darks
- There was discussion around 2023 to perform analysis on the base level dark created from *iris_make_dark.pro*. This base model has not been modified since near the start of the mission, only the long-term parameters have. The discussion involved creating a new version of the base dark files *dark_avg_fuv_20130925.dat* and *dark_avg_nuv_20130925.dat* and changing the code to use the one closest in observing time to provide a better base to generate the model dark on top of. All of the dark data exists if more versions of an average dark are desired, but that work was never moved forward with due to lack of interest.
- The ComplexA darks aren't actually used in any of the processing. They are a good dataset to have for reference throughout the mission in case we want to examine the model performance across binning modes, but this pipeline does not utilize the binned files at all. I am not aware of any part of the full IRIS processing pipeline that utilizes these files either. This can be mentioned to the IRIS team, but most likely it is best to continue to take these to have them in case any future issues arise or future analysis is desired on these files.