Death to IRAF

Vlad-Haralambie Ispas
Jake Noel-Storr
A world about it...

Image Reduction and Analysis Facility, or on short **IRAF**, is a collection of software developed by NOAO geared towards the reduction of astronomical images in pixel array form.
- It is **unintuitive** to work with.

- Despite its **complexity**, not much is left for us, the students, to grasp from it and make it **our own**.
Why bother?
Installations during this review
an end-of-support state, an

On and Analysis Fac
Because we want to make this code work for any given number of filters, we need to make the program itself be able to create lists (or later on titles) where to store the data from each file in each filter, so we will use a small trick: exec().

```python
lnf = len(filters)
nn = ['new', 'norm']
dntgtdh = ['data', 'name', 'time', 'gain', 'temp', 'dateobs', 'hdr']
for i in range(len(dntgtdh)): # fl Li
    mycode_0 = str('fl Li' + nn[i] + '_q = []')
    exec(mycode_0)

for j in range(lnf):
    mycode_1 = str('fl Li[i] + '_J' + filters[j] + '_q = []')
    exec(mycode_1)

    for k in range(len(dntgtdh)):
        mycode_2 = str('fl Li[i] + '_J' + filters[j] + '_J' + dntgtdh[k] + '_J' + [ ]; _J + fl Li[i] + '_J' + filters[j] + '_q.append(_J + fl Li[i] + '_J' + filters[j] + '_q' + dntgtdh[k] + 
exec(mycode_2)

    mycode_3 = str('_q.append(_J + fl Li[i] + '_J' + filters[j] + '_q')
    exec(mycode_3)

    exec(mycode_4)
```

END RESULT: all_q = [dark_q, bias_q, flat_B_q, flat_V_q, light_B_q, light_V_q]

Note that this is just an example, where two filters were used: B & V.

data_gwp = []
flat_data_master_q = []
for i in range(lnf):
    mycode_5 = str('fl Li[i] + '_J' + filters[j] + '_J' + data_master = []); flat_data_master_q.append(flat + filters[j] + '_data_master')
    exec(mycode_5)

    mycode_6 = str('data_imgwp' + filters[j] + '= []); data_gwp.append(data_imgwp + filters[j] + ')'
exec(mycode_6)

The same of the files as given in the header.
axis_1 = []
axis_2 = []
axis_12 = []

# Now we will create a function that will take all the data from the headers that we need and store it in separate lists (one for each category).
def g(x):
    all_q[x][0].append(the_data)
    all_q[x][1].append(all_files_names[i])
    all_q[x][2].append(int(the_hdr['EXPOSURE']))
    all_q[x][3].append(float(the_hdr['EGAIN']))
    all_q[x][4].append(float(the_hdr['CCD-TEMP']))
    all_q[x][5].append(str(the_hdr['DATE-OBS']))
# Now we want to use the previously defined function. For that we need to classify our files.

dark_search = ['Dark Frame', 'dark frame', 'DARK FRAME', 'Dark', 'dark', 'DARK']
flat_search = ['Flat Field', 'flat field', 'FLAT FIELD', 'Flat', 'flat', 'FLAT']
light_search = ['Light Frame', 'light frame', 'LIGHT FRAME', 'Light', 'light', 'LIGHT']

for i in range(len(all_files)):
    hdu = fits.open(all_files[i])
    the_data = hdu[0].data
    the_hdr = hdu[0].header

    if len(the_data) == 0:
        print('There is no data in', all_files[i], '.')
    elif len(the_hdr) == 0:
        print('There is no header in', all_files[i], '.')
    elif the_hdr['IMAGETYP'] in dark_search:
        g(0)
    elif the_hdr['IMAGETYP'] in bias_search:
        g(1)
    elif the_hdr['IMAGETYP'] in flat_search:
        g2 += filters.index(the_hdr['FILTER'].upper())
    elif the_hdr['IMAGETYP'] in light_search:
        g2 += lnf + filters.index(the_hdr['FILTER'].upper())

# Since we need all the files to have the same shape, we ought to check it before we lose time running uselessly the code.
# In the case the files do not match in shape, a list with all the files and their shapes will be displayed in order to help the
# user see what he/she is dealing with.
axis_12 = list(zip(axis_1, axis_2))
if len(set(axis_12)) == 1:
    print('Sorry, but not all the files have the same length... You need to either omit some or correct them.')
else:
    print('This is a list with all the files and their sizes: ')
    for i in range(lnf):
        print(all_files[i], ' ' * (29 - len(all_files[i])), axis_12[i])
        print('Please restart the program with the new files.')

## Goal: master bias
# Note that we use the median of all the bias frames AS AN ARRAY.
bias_data_master = np.median(bias_data, axis=0)
# Secondary goal: master dark (Fus Ro Daah)
# Note that we use again the median AS AN ARRAY.
dark_data_master = np.median(dark_data_new, axis=0)

# The dark/sec is given by the division of the master dark by the maximum (which is now 'standard') exposure time.
dark_per_sec = dark_data_master / np.max(dark_time)

### Goal: master flat & corrected images()
# We now get to work with the flat files and the light frames. Since we need to work with them one filter at a time,
# a for loop would be helpful. We will start with the flat files.
# We need to subtract the bias and perform the dark subtraction.
for i in range(lnf):
    for j in range(len(all_q[2+i][0])):
        flat_data_new_q[i].append(all_q[2+lnf][0][j] - bias_data_master - (dark_per_sec * np.array(all_q[2+lnf][2][j])))

# We measure the light as a signal: more light, bigger numbers.
# So when one shoots the flat fields, the longer it takes for the
# set to be shot, the more light will be in the images (daaaaah...).
# But there is one thing that stands out: since they are shot either in the
# evening or in the morning (not too much light, not too less), they will definitely have either less
# and less signal (sun setting down) or vice-versa (sun rising up).
# This is why, after we cleaned-up the images, we have to divide them by their
# own median in order to expect any other frame besides the one in the centre to
# be the median.
for j in range(len(flat_data_new_q[i])):
    flat_data_norm_q[i].append(flat_data_new_q[i][j] / np.median(flat_data_new_q[i][j]))

# Divide the median (AS AN ARRAY) by its own mean.
flat_data_master_q[i] = (np.median(flat_data_norm_q[i])) / np.mean(np.median(flat_data_norm_q[i]))

# 'Clean' the image(s) (remove bias, perform the dark subtraction), then divide by the master flat.
for j in range(len(all_q[2+lnf+i][0])):
    data_gwp[i].append(all_q[2+lnf+i][0][j] - bias_data_master - (dark_per_sec * np.array(all_q[2+lnf+i][2][j])) / flat_data_master_q[i])

# Because democracy...
color = input('Do you want to see the images using "jet" or just in black and white (j/b)?')
if color == 'j': color='jet'
if color == 'b': color='gray'

centoventi = input('(If you will opt to visualize any images later on, would you want to see them unaltered or with
'a specific percentage of their upper and lower values being cut (this makes the dimmer objects in
'the frame much brighter, but will take a few extra minutes) (u/a)?')
Thanos_2.1 Last Checkpoint: 09/30/2019 (autosaved)
- the ability to use files from different folders and auto-select those that are viable

- the ability to work with any filters, regardless of their type or number, by creating individual, specially named lists containing the respective data sets, thus making easy for the user to search through them

- performing data reduction

- sorting them based on the their type (bias frames, dark frames, flat fields or light frames) and (for the last three ones) also on the filters used when capturing them

- viewing the obtained frames with the option to cut a percentage of the top/lower values in each, thus making the analyzing process easier by not occupying a range of the colors that can be displayed with unimportant values (counts)

- shifting the obtained frames and (after letting the user chose one for reference) compiling them into a single one

- repeating the above procedure for the resulted shifted frames and obtaining a single, final, frame

- saving by choice any of the obtained frame with a conclusive name
In [1]: from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))

In [322]: %matplotlib inline
import os
from astropy.io import fits
from matplotlib.pyplot import figure, show
import matplotlib.pyplot as plt
import numpy as np
from scipy.ndimage import interpolation as interp
from mpl_toolkits.mplot3d import Axes3D
from skimage.feature.register_translation import (register_translation, _upsampled_dft)
import matplotlib

# Here we just want to get the path/paths to the directory/directories where the files are located.
path_list = []
print('This program will filter a provided set of images (light frames).')
print()
y0 = input('Is this program located in the same folder as (all) the files (y/n)?')
if y0 == 'y':
  path_list = ['.']
if y0 == 'n':
  y1 = input('Then, at least, are all the files located into the same folder (y/n)?
if y1 == 'y':
  path_list.append(str(input('Then what is the (relative of full) path to that folder?')))  
if y1 == 'n':
  print('Then what is the (relative or full) path to each folder (one at a time and when done, press enter)?)
  while True:
    answ_0 = str(input())
    if answ_0 == '/': # juuuust to make sure...
      path_list.append(answ_0)
    else:
      answ_0 = '/'
      path_list.append(answ_0)
    else:
      break
  print('There are', len(path_list), 'paths given above.')

# Now we want to get the files from this/those directory/directories.
all_files = []
for d in os.walk(path_list[i]):
  d, f in os.walk(path_list[i]): # r=root, d=directories, f=files
    for file in f:
      all_files.append(file)
Thanos - the show must go on

- a more complete description of the individual files in their headers
- the ability to perform photometry
- the addition of other correcting techniques
Thank you... ^^