pyDIA: GPU-accelerated wide-field difference-imaging photometry

Michael D. Albrow
Department of Physics and Astronomy
University of Canterbury

Introduction

pyDIA is a modular python package for performing star detection, difference imaging and photometry on astronomical images. The guiding philosophy was to write as much of the software as possible in easy-to-read python, while using embedded CUDA C (or externally compiled C) to perform the most computationally-intensive tasks. This approach was adopted to allow the software to be easily used and extended for data pipelines. The code can be run either using an NVDIA GPU (in which case the CUDA version is used) or on a conventional CPU (using python ctypes).

The difference-imaging part of this software implements the algorithm of Bramich et al. (2013)[1] with extended delta basis functions as defined in that paper. This allows independent control of the degrees of spatial variation for the differential photometric scaling and differential PSF variations between images. This part of the software uses embedded CUDA or external C code.

The photometry part of the software is also written in CUDA and C and designed to be very fast. Typical throughput is ~ 5000 flux measurements per second using a GPU.

Hardware Requirements

PC or Mac computer (for CPU version).

To use the GPU version, you must also have an NVIDIA GPU that is available for computation (i.e. not being used to drive the display). The code has been tested with NVIDIA GTX970, C2075 and K40 GPU's.

[1] Bramich, D.M., Horne, K., Albrow, M.D., et al., 2013, MNRAS, 428, 2275

Software Requirements

Linux or OSX operating system.

Python 2.7 with installed packages:

- numpy
- scipy
- astropy
- pyraf
- pyCUDA (for GPU version)
- scikit-learn
- scikit-images

I use both the Ureka astronomy software distribution from http://ssb.stsci.edu/ureka/, and the Anaconda python distribution from https://www.continuum.io/downloads with the AstroConda environment from http://astroconda.readthedocs.io/en/latest/. These both contain python2.7 with numpy, scipy, astropy and pyraf.

Pycuda is available from http://mathema.tician.de/software/pycuda/, and requires preinstallation of the CUDA software from https://developer.nvidia.com/cuda-downloads.

If necessary, the scikit-learn and scikit-images packages can be installed using pip.

Installation

pyDIA can be obtained from MDA in the form of a compressed tar file, pyDIA.tar.gz, or from http://www2.phys.canterbury.ac.nz/~mda45/pyDIA/pyDIA.tar.gz. This can be unpacked to any suitable location on your computer by typing
tar xf pyDIA.tar

This will result in a simple directory structure

pyDIA:

- __init__.py
- DIA_common.py
- DIA_CPU_header.py
- DIA_GPU_header.py
- analysis_functions.py
- calibration_functions.py
- c_functions.c
- c_ interface_functions.py

cuda_functions.py
cuda_interface_functions.py
data_structures.py
detect.py
image_functions.py
io_functions.py
photometry_functions.py
Makefile

All are readable text files of python code, except for cuda_functions.py (which is CUDA C) and c_functions.c (which is the C equivalent).

To prepare both CPU and GPU versions, just type “make”.

Data structures

The are two fundamental object classes defined in data_structures.py.

class Parameters

This is used as a container to store all configurable parameters.

class Observation

An Observation instance is the primary data object. At initiation, it reads an image and computes its saturation mask, variance, seeing, background, etc.

High-level functions

The highest-level routines are found in DIA_common.py, and provide a data-reduction pipeline that is ready to use for many applications.

difference_image(ref_image, target_image, params, stamp_positions=None,
psf_image=None, star_positions=None,
star_group_boundaries=None, detector_mean_positions_x=None,
detector_mean_positions_y=None)

Create a difference image by mapping ref_image to target_image, with both of them aligned to registration_image.

make_reference(files, reg, params, reference_image='ref.fits')

Create a photometric reference image by combining the best-seeing images in a list of files.

imsub_all_fits(params, reference='ref.fits')

Create a reference image, detect stars, make difference images, measure difference fluxes. This function is the normal interface to be called your own script.

do_photometry(params, extname='newflux', star_file='star_positions',
psf_file='psf.fits', star_positions=None, reference_image='ref.fits')

Redo photometry of all stars or one or more new stars.

Imsub_all_fits is the basic interface for data reduction and can be run from a very simple python script, e.g.

```
#
#  Set the package install location
#
import sys
import os
import fileinput
sys.path.append('/home/mda45/PythonPackages')

#
# Import the high-level pipeline routines
#
use_GPU = False
if params.use_GPU:
   from pyDIA import DIA_GPU as DIA
else:
   from pyDIA import DIA_CPU as DIA

from pyDIA import calibration_functions as cal

#
# Load the default parameters
#
params = DIA.Parameters()
params.use_GPU = use_GPU

#
# Override parameter defaults if necessary
#
params.gain = 1.55
params.readnoise = 5.0
params.pixel_min = 10
```

```
params.pixel_max = 140000
params.name_pattern = 'Input_images/k_*.fits'
params.datekey = 'PHJDMID'
params.pdeg = 1
params.sdeg = 1
params.bdeg = 1
params.use_stamps = True
params.nstamps = 100
params.loc_output = 'Output_dir'

#
# Perform the difference imaging, photometry and calibration
#
if not(os.path.exists(params.loc_output)):
   DIA.imsub_all_fits(params)
if not(os.path.exists(params.loc_output+'/calibration.png')):
   cal.calibrate(params.loc_output)
```

The default values for all data and reduction parameters are set in the file, data_structures.py, and described later in this document.

Adding new images

If you have run pyDIA on some initial set of images, and now have one or more new images, then simply copy them into the data directory and rerun imsub_all_fits. The code will automatically re-use the existing reference image and PSF, and skip over any images it has already processed.

Re-doing photometry

If you have processed a set of images, but want to rerun the photometry without re-making the difference images (say if you want to try the photometry with a different PSF fit radius), then make a copy of your run script and replace the imsub_all_fits command with do_photometry. Note that you can change the output file extension so as to not overwrite your original results. The do_photometry command can also be used to do photometry at a one or more star positions that weren't in your original list.

Mid-level functions

The files image_functions.py, io_functions.py and photometry_functions.py contain mid-level functions, accessible by importing the relevant python module. These routines can be combined to build a data processing pipeline.

image_functions.py

positional_shift(R, T)

Cross-correlate two images to find a small xy offset.

register(R, T , params)

Align image T to image R by an integer XY shift.

compute_bleed_mask(d, radius, params)

Detect and mask bleeding along columns from saturated pixels.

compute_saturated_pixel_mask(im, radius, params)

Compute a mask image (good pixel = 1, bad pixel = 0) where bad pixels are defined as those within a radius of a saturated or low value.

compute_saturated_pixel_mask_2(im1, im2, radius, params)

As above, but use saturated or low pixel values from either of two images.

cosmic_ray_clean(data, params)

Clean cosmic rays from data using the LA Cosmic package. Requires separate installation. Not used.

kappa_clip(mask, norm, threshold)

Iteratively mask pixels exceeding threshold standard deviations from zero.

boxcar_blur(im)

Convolve image with a square 3x3 kernel.

convolve_undersample(im)

Convolve image with a 3x3 pixel tophat.

convolve_gauss(im,fwhm)

Convolve image with a 2-d Gaussian.

apply_photometric_scale(d, c, pdeg)

Divide image d by a spatial polynomial of degree pdeg with coefficients c.

undo_photometric_scale(d, c, pdeg)

Multiply image d by a spatial polynomial of degree pdeg with coefficients c.

compute_fwhm(f, params, width=20, seeing_file='seeing')

Use autocorrelation to compute the seeing FWHM for an image.

subtract_sky(image, params)

Subtract the background from an image.

mask_cluster(image, mask, params)

Mask the central region of a globular cluster.

define_kernel_pixels(rad, INNER_RADIUS=7)

Define the pixel centres for a mixed-resolution circular kernel.

io_functions.py

get_date(file, key='JD'):

Read a date from the fiits header of file.

read_fits_file(file, slice=None):

Read the data and header from a fits file.

write_image(image, file):

Write image to fits file.

write_kernel_table(file, kernel_index, extended_basis, coeffs, params)

Write a convolution kernel to a FITS table file.

read_kernel_table(file, params)

Read a convolution kernel from a FITS table.

photometry_functions.py

compute_xy_shift(pos1 , pos2, threshold, dx=0.0, dy=0.0**)**

Compute the xy offset between two sets of positions.

detect_stars(f, params)

Detect star positions and magnitudes.

choose_stamps(f, params)

Select image regions around bright stars.

rewrite_psg(file1, file2)

Remove PSF stars from a DAOphot psg file.

compute_psf_image(params, g , psf_deg=1, psf_rad=8, star_file='phot.mags',
psf_image='psf.fits')

Detect stars and compute the PSF for image g. Return a list of stars.

group_stars_ccd(params, star_positions, reference)

Arrange list of star positions into tile groups.

Low-level functions

These routines in cuda_interface_functions.py and cuda_functions.py are not recommended for access directly. They are better used through the high-level function, difference_image, in DIA.py. They should not be edited unless you are sure you know what you are doing (i.e. you have a good knowledge of multithreaded programming and the CUDA language). Small changes to these files can cause the software to break.

c_interface_functions.py, cuda_interface_functions.py

These are python functions to call the C or CUDA C routines in c_functions.c and cuda_functions.py

numpy3d_to_array (np_array, allow_surface_bind=True)
array_to_numpy3d(cuda_array)

Convert back and forth between python and CUDA C array data. Taken from the pyCUDA examples documentation at http://wiki.tiker.net/PyCUDA/Examples/Demo3DSurface

compute_matrix_and_vector_cuda(R, RB, T, Vinv, mask, kernelIndex,
extendedBasis, kernelRadius, params,
stamp_positions=None)

compute_model_cuda(image_size, texref, c,k ernelIndex, extendedBasis,
params)

photom_all_stars(diff, inv_variance, positions, psf_image,c , kernelIndex,
extendedBasis, kernelRadius, params,
star_group_boundaries=None,
detector_mean_positions_x=None,
detector_mean_positions_y=None)

convolve_image_with_psf(psf_image, image1, image2, c, kernelIndex,
extendedBasis, kernelRadius, params)

cuda_functions.py

These are embedded CUDA C functions for the GPU.

deconvolve3_columns(int width,int height,int rowstride,
double *data,double *buffer,double a,double b)
deconvolve3_rows(int width,int height,int rowstride,double *data,
double *buffer,double a,double b)
resolve_coeffs_2d(int width, int height, int rowstride, double *data)
interpolate_2d(double x,double y,int rowstride,double *coeff)

Functions for interpolation using cubic-OMOMS polynomials. Taken from the Gwiddion software for scanning probe microscopy (http://gwyddion.net/), which is distributed under the GNU General Public License.

integrated_profile(int profile_type, int idx, int idy, float xpos,
float ypos, float *psf_parameters, float *lut_0,
float *lut_xd, float *lut_yd)

Compute a pixellated version of the PSF.

convolve_image_psf(int profile_type, int nx, int ny, int dx, int dy,
int dp, int ds, int n_coeff, int nkernel,
int kernel_radius,int *kxindex,
int *kyindex, int* ext_basis, float *psf_parameters,
float *psf_0, float *psf_xd, float *psf_yd,
float *coeff,float *cim1, float* cim2)

Convolve two images with the PSF.

cu_photom(int profile_type, int nx, int ny, int dp, int ds, int n_coeff, int nkernel,
int kernel_radius,
int *kxindex, int *kyindex, int* ext_basis, float *psf_parameters,
float *psf_0, float *psf_xd, float *psf_yd, float *posx, float *posy,
float *coeff, float *flux, float *dflux)

Read PSF, convolve it, map to subpixel star location, fit to image. Each star is a single CUDA block. Each block is a 16 x 16 array of threads.

cu_compute_model(int dp, int ds, int db, int *kxindex, int *kyindex, int* ext_basis,
int nkernel, float *coefficient, float *M)

Convolve an image with a previously-determined kernel.

cu_compute_vector(int dp, int ds, int db, int nx, int ny, int *kxindex, int *kyindex,
int *ext_basis, int nkernel, int kernelRadius,float *V)

Use full images to compute the vector described in Eqn 21 of Bramich et al. (2013).

cu_compute_vector_stamps(int dp, int ds, int db, int nx, int ny, int nstamps,
int stamp_half_width, float *stamp_xpos,
float* stamp_ypos, int *kxindex, int *kyindex,
int *ext_basis, int nkernel, int kernelRadius,float *V)

Use image sections to compute the vector described in Eqn 21 of Bramich et al. (2013).

cu_compute_matrix(int dp, int ds, int db, int nx, int ny, int *kxindex, int *kyindex,
int *ext_basis, int nkernel, int kernelRadius,float *H)

Use full images to compute the matrix defined in Eqn 20 of Bramich et al. (2013). Each CUDA block is a matrix element. Threads range over image pixels.

cu_compute_matrix_stamps(int dp, int ds, int db, int nx, int ny, int nstamps,
int stamp_half_width, float *stamp_xpos,
float* stamp_ypos, int *kxindex, int *kyindex,
int *ext_basis, int nkernel, int kernelRadius,float *H)

Use image sections to compute the matrix defined in Eqn 20 of Bramich et al. (2013).

c_functions.c

These are C versions of the CUDA routines from cuda_functions.py for the CPU version of the code. Apart from cu_photom, these are almost exact copies.

Calibration, Variable-object detection and Analysis functions

These subpackages are under active development. Their documentation may be obsolete and is certainly incomplete.

The subpackage **calibration_functions.py** provides a set of functions to calibrate the DAOphot magnitudes to the pyDIA reference flux measurments, and to produce blending-corrected versions of the reference-fluxes.

locate_intercept(x,y)

Filter outliers and fit a straight line to y(x).

calibrate(dir, plotfile='calibration.png')
Calibrate photometry.

makeCMD(dirI, dirV, bandwidth = 0.25)

Make colour-magnitude diagrams and estimate red clump centroid.

The subpackage **analysis_functions.py** provides a growing number of routines for lightcurve analysis.

readflux(data_params)

Read reference and difference fluxes from *.flux files.

fix_flux(ref,mag_range)

Convert reference fluxes to DAOphot magnitude basis.

extract_lightcurves(files,ref):

Convert fluxes to relative differential measurements and magnitudes and return these as 2-dimensional arrays. Also return the dates as an array.

find_star_at(xpos,ypos,ref)

Find the closest star to a given location.

plot_lightcurve(d,y,ye)

Plot a lightcurve.

detect_variables(files,sn_file='SN.fits')

Detect variable objects using coadded difference images.

trend_filter(y,n,iterations=1,star_number=None,mode='mean')

Filter lightcurves using the TFA algorithm of Kovacs, Bakos & Noyes (2005). The subpackage **detect.py** provides routines for the dection of variable objects from difference images, and their photometry.

gauss(x, p)

Gaussian function evaluation.

final_variable_photometry(files, params, coords=None, coord_file=None, psf_file=None)

Photometry centred at coords using routines from photometry_functions.py.

detect_variables(params, psf_file=None, ccoords=None, time_sigma=4.0)

Detect features in data cube of difference images. Converge their positions, do photometry.

do_photometry_variables(files, params, position, extname='vflux',
psf_file=None, reference_image='ref.fits', iterations=10)

Photometry with centroid convergence using Albrow et al. (2009) algorithm.

Parameters

All parameters are initially defined and assigned default values in file data_structures.py. They need to be imported and can be reassigned if necessary.

Parameter	Default value	Description
bdeg	0	Degree of spatial variation of the differential bakground
ccd_group_size	100	For CCD code version, only reevaluate the PSF for position changes greater than this parameter.
cluster_mask_radius	50	Radius of circular region to mask if mask_cluster = True.
datekey	'MJD-OBS'	Date field in FITS headers
detect_threshold	4.0	Sigma threshold for variable object detection.
diff_std_threshold	10.0	Threshold for good images to use for variable object detection
do_photometry	True	Measure stellar fluxes from the difference images
fwhm_mult	5	Multiplier to determine kernel size
fwhm_section	None	Array of 4 numbers describing the bottom-left and top-right corners of a rectangular section of each image to use for FWHM estimation
gain	1.0	Inverse-gain of the CCD (e-/ADU)
image_list_file	'images'	Write image names and dates.
iterations	1	Number of kernel iterations
kernel_maximum_radius	20.0	Maximum radius for the convolution kernel
kernel_minimum_radius	5.0	Minimum radius for the convolution kernel
loc_data	'.'	Absolute or relative path to the directory containing the input images.
loc_output	'.'	Absolute or relative path to the

		directory to store the output files. Will be created if it doesn't exist.
mask_cluster	False	If True, the attempt to detect and mask the highest concentration of stars in each image.
min_ref_images	3	Minimum number of images to combine for the reference.
n_parallel	1	Number of parallel CPU parallel processes to run. Typically set this to equal the number of CPU cores available.
name_pattern	'*.fits'	Pattern describing data file names
nstamps	200	How many stamps to use
pdeg	0	Degree of spatial variation of the kernel photmetric scale
pixel_max	50,000	Maximum valid pixel value
pixel_min	0.0	Minumum valid pixel value
pixel_rejection_threshold	3.0	Threshold for masking outlying pixels after each kernel iteration.
psf_fit_radius	3.0	Radius (in pixels) for PSF photometry.
psf_profile_type	'gaussian'	The only option at present.
readnoise	1.0	CCD readout noise (e)
ref_image_list	'ref.images'	List of images used to create the reference.
ref_exclude_file	None	If this file exists, it should contain a list of images to exclude from the photometric reference
ref_include_file	None	If this file exists, it should contain a list of images to use for the photometric reference
reference_min_seeing	1.3	Minimum FWHM for images to be included in the photometric reference
reference_seeing_factor	1.1	Include images in the photometric reference that have FWHM less than this factor times the lowest FWHM.
reference_sky_factor	1.3	Include images in the photometric reference that have backgrounds less than this factor times that of the image with the lowest FWHM.
registration_image	None	Use this FITS file as the astrometric reference. Otherwise use the best-

		seeing image.
sdeg	0	Degree of spatial variation of the kernel shape variation
sky_degree	0	Degree of spatial variation allowed for the sky background model
sky_subtract_mode	'percent'	If this parameter is set to 'default', fit a 2D polynomial model for the sky. If this parameter is set to 'percent', then subtract a constant percentage value from each image.
sky_subtract_percent	0.01	Sky percentage to subtract if sky_subtract_mode = 'percent'
stamp_edge_distance	40	Minimum distance in pixels from the centre of a stamp to the edge of the detector.
stamp_half_width	20	Size of each stamp
star_detect_sigma	12	Minimum signal-to-noise for star detection.
star_file	None	If provided, use this catalogue of star positions for photometry. The first 3 columns must be star_number, x_position, y_position. Rows starting with # are ignored.
star_file_has_magnitudes	False	If true, assume column 4 of star_file contains magnitudes.
star_file_is_one_based	True	If true, assume the coordinates in star_file are based on (1,1) being the lower left pixel of the detector.
subtract_sky	True	Pre-subtract the sky background from each image
use_GPU	True	Flag to indicate GPU or CPU use.
use_stamps	False	Use small image sections to compute the kernel (as opposed to using the whole image)

Output files

Most of the output files have prefixes or suffixes appended to the corresponding input file names. The important files are:

File name	Description
ref.fits	Reference image
k_*.fits	FITS ascii table of the convolution kernel
d_*.fits	Difference image normalised to ref.fits
n_*.fits	Difference image normalised to pixel noise
r_*.fits	Registered image
sm_*.fits	Saturation mask image
m_*.fits	Model image
z_*.fits	Mask image
psf.fits	PSF image determined by DAOphot
ref.mags	Stellar magnitudes and positions determined by DAOphot
ref.flux	Stellar fluxes measured from ref.fits
*.flux	Stellar differential fluxes measured from difference image

Known problems

None at present. Please report bugs to Michael.Albrow@canterbury.ac.nz