ZOGY in Parallel Documentation

Release latest

Feb 26, 2020

Contents

CHAPTER 1

ZOGY in Parallel

A python package for image subtraction, registration, and co-addition

1.1 How to install

ZiP can be installed with pip

```
pip install zogyp
```

Dependancies

- numpy
- scipy
- astropy
- sep
- SExtractor
- PSFex

1.2 About

ZOGY in Parallell (ZiP) is a fast(ish) computation of proper image subtraction B.Zackay, E.Ofek, A.Gal-Yam (2016). Inspired by Ofek (2014) and pmvreeswijk. ZiP offers a faster subtraction at the expense of a more comprehensive input. I.e. The program should be tailored for one telescope or input of images. This code has a parallell function, however it requires 6+ cores to operate. This particular Case is optimised for the Gravitational-Wave Optical Transient Observer (GOTO) However, simple fudging of the parameters should make it possible to make this work for other telescopes.

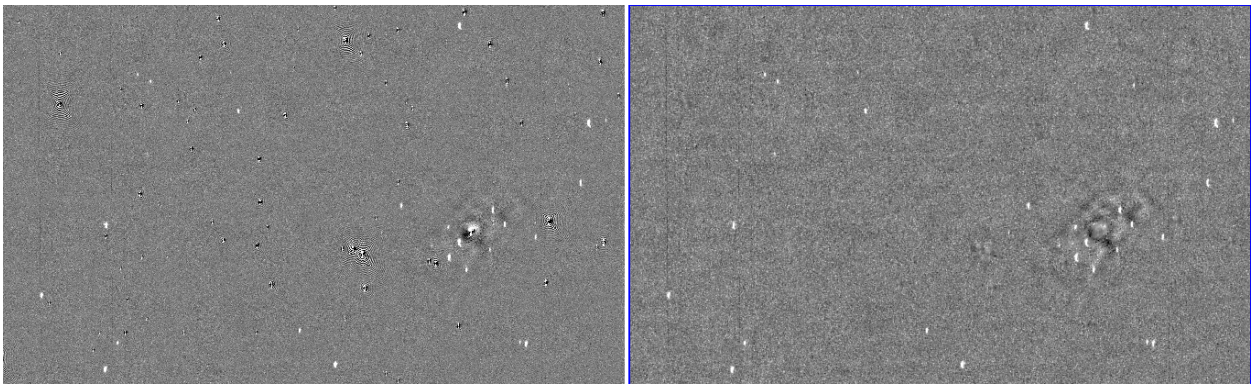


Fig. 1: Left: is the D-output image. Right: Scorr image.

An internal version of Spalipy has been added as the alignment algorithm. This uses sextractor to find source locations in two images and then aligns them with an affine transform. Residuals are then used to build a 2D spline surface to correct for warping due to large field distortions.

Finally, a parallel version of proper coadition is used for stacking images. It still isn't increadibly fast for on the spot coaddition; so a meidian combine tool is also included.

1.3 Basic Usage

ZiP can now be used in the terminal:

```
ZOGY file.fits ref-file.fits

#-------------------------------------------------
#Takes Comprehensive inputs too now

ZOGY file.fits ref-file.fits -sub_imagex 2 -sub_imagey 2
```

In the shell use:

```
ZOGY -help
```

For a list of helpful parameters

You can also import zogyp into python:

```
#You can also make your own scripts

import zogyp

zogyp.zip.run_ZOGY(sci,ref)
```

This will do the subtraction (sci - ref) and return the subtracted in an output directory called **Zoutput**.

The output consists of the D_image, S_image, and Scorr_image.

There is a comprehensive tutorial on GitHub

```
Ry Cutter

Version 1.6.3 (25/02/2020)
```

CHAPTER 2

ZOGY

This is the subtraction code, below describes the algorithm and how to use it

2.1 The Maths

To find the raw subtraction *D*:

$$\widehat{D} = \frac{\left(F_r \widehat{P_r} \widehat{N} - F_n \widehat{P_n} \widehat{R}\right)}{\sqrt{\sigma_n^2 F_r^2 |\widehat{P_r}|^2 - \sigma_r^2 F_n^2 |\widehat{P_n}|^2}}$$

Looking at the statistical image:

$$S = F_d D \otimes \overleftarrow{P_D}$$

and finally, the correlated noise statistical image:

$$S_{corr} = \frac{S}{\sqrt{V(N_s) + V(R_s) + V_{fwhm}(N_s) + V_{fwhm}(R_s)}}$$

As described in B.Zackay, E.Ofek, A.Gal-Yam (2016).

2.2 Running ZOGY Tutorial

Install the standards

```
import glob
import ntpath
import time
import shutil
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
from astropy.io import fits
import numpy as np

#ZOGY in Parallel routines
from zogyp.zip import run_ZOGY
from zogyp.zip import rdfits
from zogyp.zip import config_loc

#Image alignment
#from zogyp.spali2 import spalipy

#Stacking
#from zogyp.zo_coadd import med_comb
#from zogyp.zo_coadd import prop_coad
```

Find the directory of the configuration files, these will need to be edited for best peformance

```
print(config_loc())
```

The two files of interest are default.sex and psfex.conf

Get the test images

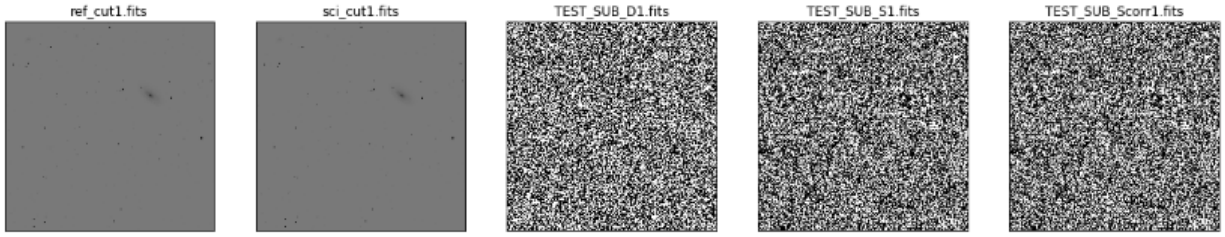
```
t_f = LOC.replace('configfls','test')
T = [i for i in glob.glob(t_f+'/*')]
```

Basic Subtraction

```
run_ZOGY(T[0], T[1], outname='TEST_SUB') #ZOGY image subtraction T[0] - T[1]
```

Let's plot the subtractions

```
file_names = glob.glob('Zoutput/*')
fig, axs = plt.subplots(1, 5, figsize=(20,45))
for i in range(len(file_names)):
    D = fits.getdata(file_names[i])
    axs[i].imshow(D, cmap = 'gray_r', vmin = -np.mean(D)*20, vmax= np.mean(D)*20)
    axs[i].set_xticks([] , [])
    axs[i].set_yticks([] , [])
    axs[i].set_title(ntpath.basename(file_names[i]))
plt.show()
```



ZiP is a fast image subtraction tool. If speed is a vital component for your subtraction, pay attention to the clean_ref and clean_sci parameters. They exist to clean up small order over fitting of the PSF from PSFex. The smaller the parameter the faster the subtraction, (usually 0.25 should suffice). As this is on a normalised PSF, the cleaning parameter has to be between 0 and 1 and will remove any pixel values in the kernel smaller than the cleaning param

```
Array = []
PLOTS = []
for i in range(11):
    t = time.time()
    X = i/10
    run_ZOGY(T[0], T[1], clean_ref = X, clean_sci = X)
    Array.append([X, (time.time() - t)])
    plt.imshow(fits.getdata('Zoutput/data_D1.fits')[630:690, 1254:1300])
    plt.xticks([], [])
    plt.yticks([], [])
    plt.
    show()
```

Use PSF slices to do subtractions even faster. Slices are used primarily to apply a varrying PSF across the field. This can be done in parallel and the slices are smaller which speeds up the subtraction

```
run_ZOGY(T[0], T[1], xslice=2, yslice=2)
```

Sub images (not to be confused with the PSF slices from above) split the main image. These sub images can be subtracted simultaneously, and is the backbone of what makes ZiP so fast. These output images should be treated as seperate (independant images) when doing analyses.

```
run_ZOGY(T[0], T[1], sub_imagex=1, sub_imagey=2)
```

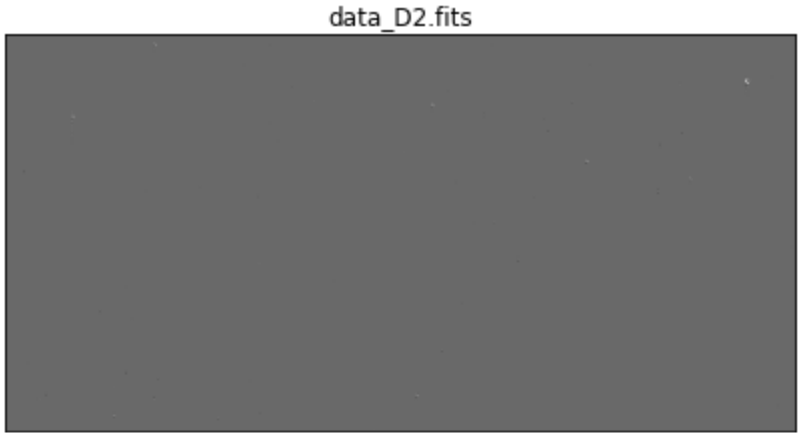
Plot the sub-images

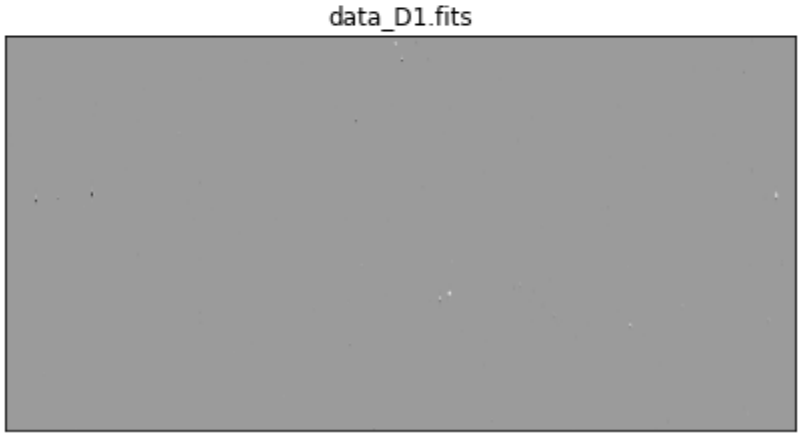
```
file_names = glob.glob('Zoutput/data_D*')
fig, axs = plt.subplots(2, 1, figsize=(10,8))
for i in range(len(file_names)):
    D = fits.getdata(file_names[i])
    axs[i].imshow(D, cmap='gray')
    axs[i].set_xticks([] , [])
    axs[i].set_yticks([] , [])
    axs[i].set_title(ntpath.basename(file_names[i]))
plt.show()
```

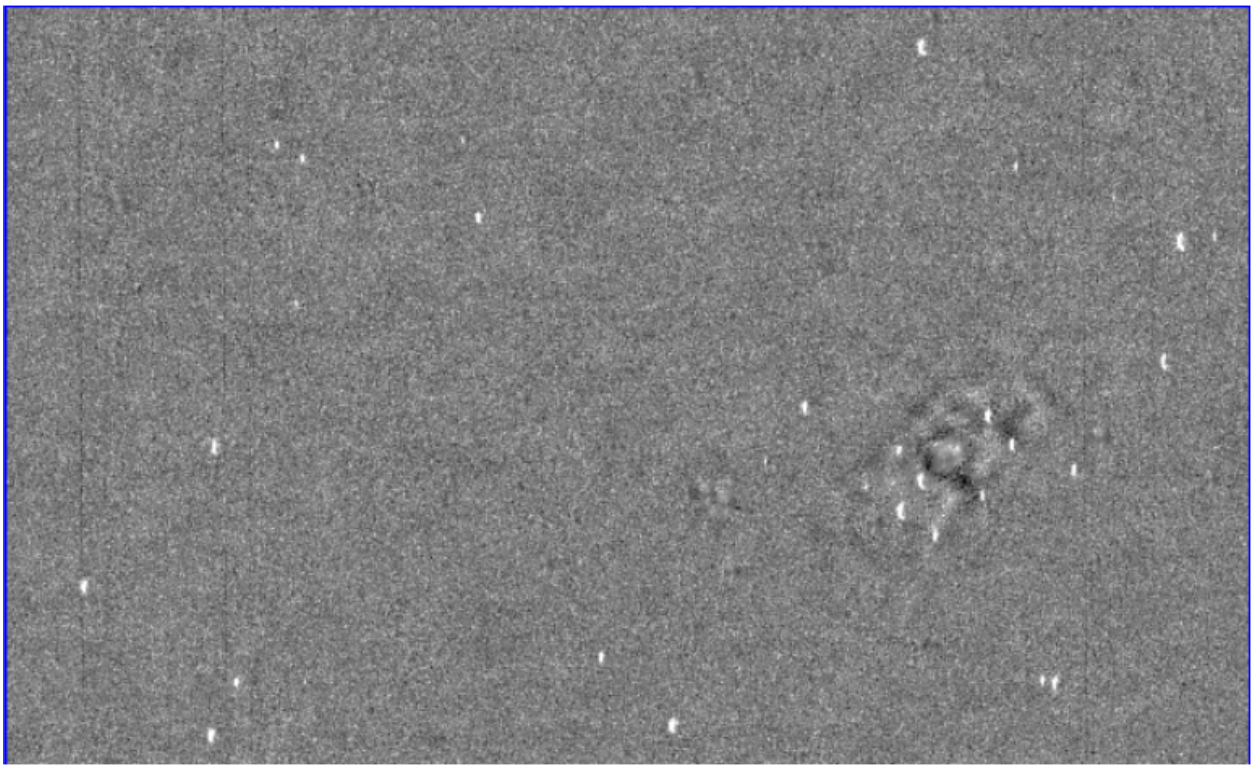
Finally, put it all together!

```
run_ZOGY(T[0], T[1], sub_imagex=1, sub_imagey=2,
         xslice=2, yslice=2, blackout=True,
         clean_ref = 0.75, clean_sci = 0.75,
         outname = 'FINAL')

plt.imshow(fits.getdata('Zoutput/FINAL_Scorr1.fits'), cmap='gray_r' ,vmin=-0.25,␣
↪vmax=0.25)
plt.xticks([],[])
plt.yticks([],[])
plt.show()
```


data_D2.fits


data_D1.fits



2.3 Parameters

Table 1: Title

Parameter	Purpose	Range
sub_imagex	Number of sub-images from slices on the x-axis	>0
sub_imagey	Number of sub-images from slices on the y-axis	>0
xslice	PSF kernels built for each sub-image x-axis	>0
yslice	PSF kernels built for each sub-image y-axis	>0
blackout	Clears any part of the reference image that does not overlap with the science image	True / False
clean_sci	cleans the science PSF kernel, anything less than the param = 0	0<, <1
clean_ref	cleans the reference PSF kernel, anything less than the param = 0	0<, <1
outname	Name for the output files	str
align	If true, will align the science image to the reference	True / False

2.4 Tips and Tricks

To Do

CHAPTER 3

Spali2

3.1 A robust image registration technique for wide fields of view

spali2 is the successor to the detection based image registration algorithm spalipy . *Here, this section will cover the functionality and how to use spali2*

3.2 The Algorithm

Firstly, two catalogs are built using *sextractor*. These catalogs will detail source positions and fluxes. Quads are built in both catalogs (Hogg et al 2010) and are then compared to one another. Using the position, rotation, and scaling differences, an affine transform is built (scipy). **!!! WARNING: the affine transformation uses scipy.ndimage.interpolation.affine_transform which does not handle NaNs properly. Replace all NaN values in the input image prior to running spalipy !!!**

Second, once the affine transform is done, the remaining residuals are compared to see where higher order transfroms are needed. This builds a 2D-spline surface to represent the residuals in x and y axes. The spline surfaces are then implemeneted to properly register the images.

3.3 Using Spali2

As usual, import stuff

```
import glob
import ntpath
import time
import shutil
import subprocess
import matplotlib.pyplot as plt
from astropy.io import fits
```

(continues on next page)

(continued from previous page)

```
import numpy as np

#ZOGY in Parallel routines
from zogyp.zip import run_ZOGY
from zogyp.zip import rdfits
from zogyp.zip import config_loc

#Image alignment
from zogyp.spali2 import spalipy

#Stacking
#from zogyp.zo_coadd import med_comb
#from zogyp.zo_coadd import prop_coad
```

To align image1 to image2

```
Output = spalipy(image1, image2, name='SPLINE.fits')
```

If your images are not sextractor compliant, you can remake the fits files and try again

```
R = rdfits(image1)
D = rdfits(image2)
Output = spalipy(R, D, name='SPLINE.fits')
subprocess.call(['rm', R, D])
```

Finally, if your images are small and not prone to wide distorions or you are really pressed for time, you can take out the spline step.

```
ali = spalipy(image1, image2, spline=False)
```

CHAPTER 4

Proper Coadition

This is the Co-Addition (or image stacking) tool, below describes the algorithm and how to use it The following was developed from B.Zackay and E.O. Ofek. (2015a) and B.Zackay and E.O. Ofek. (2015b).

4.1 The Maths

To find proper image from co-addition, *R*:

$$\widehat{R} = \frac{\sum_j \frac{F_j}{\sigma_j} \overline{\widehat{P_j}} \widehat{M_j}}{\sum_j \sqrt{\frac{F_j^2}{\sigma_j^2} |\widehat{P_j}|}}$$

The derrivation can be found in B.Zackay, et al. (2016).

4.2 Using Proper Co-Addition

Import everything!

```
import glob
import ntpath
import time
import shutil
import subprocess
import matplotlib.pyplot as plt
from astropy.io import fits
import numpy as np

#ZOGY in Parallel routines
#from zogyp.zip import run_ZOGY
#from zogyp.zip import rdfits
```

(continues on next page)

(continued from previous page)

```
from zogyp.zip import config_loc

#Image alignment
#from zogyp.spali2 import spalipy

#Stacking
from zogyp.zo_coadd import med_comb
from zogyp.zo_coadd import prop_coad
```

The coad funcitons take a list of fits files and stacks them (Assuming they are aligned)

This can be done using either the directory of the images needing to be stacked:

```
Ref_data = prop_coad(["path/to/Directory/"])
```

or using the individual file names:

```
Ref_data = prop_coad(["list","of","aligned","fits","files"])
```

you can save the reference image too

```
out_file, R = prop_coad(["Directory"], make_fits=True)
```

Finally all of this can be done with median combination also (which is significantly faster, but leaves a lower quality reference)

```
out_file, R = med_comb(["Directory"], make_fits=True)
```

CHAPTER 5

Pipelines