ois Documentation

Release 0.1a0

Martin Beroiz

Nov 26, 2022

Contents

OIS is a Python package and a C command-line program to perform optimal image subtraction on astronomical images.

It offers different methods to subtract images:

- Modulated multi-Gaussian kernel (as described in [alard1998])
- Delta basis kernel (as described in [bramich2008])
- Adaptive Delta Basis kernel (as described in [miller2008])

Main features:

- Each method can (optionally) simultaneously fit and remove common background.
- Each method can resolve small translations on the image
- Adaptive Bramich can resolve small relative rotations on the images

(See *Methods*)

CHAPTER 1

Installation

Install it directly from PyPI using pip:

```
pip install ois
```

1.1 Contents:

1.1.1 Usage

Optimal Subtraction

If your image has a relatively narrow field of view where your PSF doesn't change significatively across the field, you can use `optimal_system` on the default settings:

```
>>> from ois import optimal_system
>>> diff_image, optimal_image, kernel, background = optimal_system(test_image,␣
↪refimage)
```

Where (See *Theory*):

- `test_image` is the image we want to analize, I
- `refimage` is an archive or reference image from the same location in the sky, R
- `diff_image` is $D = I - (R \otimes K + B_{kg})$
- `optimal_image` is $R \otimes K + B_{kg}$
- `kernel` is K
- `background` is B_{kg}

Note: `test_image` must be previously aligned with `refimage`

Note: The subtraction works best when `refimage` is of better quality than `test_image`.

The default method for kernel fit is Bramich (2008), which uses the information of all pixels in the image and fits for every pixel in the convolution kernel independently. The available methods are `"Alard-Lupton"`, `"Bramich"` and `"AdaptiveBramich"` (see *Methods*).

Refer to the *Module API* for a complete description of the method.

Working with bad pixels (masks)

If your reference image or test image have sections of bad pixels, it can distort the kernel estimation. This is especially true when the fitting algorithm uses all the pixels in the image. Saturated stars can also confuse the fit of the convolution kernel.

To let *ois* know which pixels are good, you can create a numpy masked array, with `True` on bad pixels. The ois subtraction methods will ignore completely the information on those bad pixels.

The returned image, will have a combined `OR` mask from the mask in `test_image` and the mask on `refimage` expanded to exclude pixels that would have used defective pixels in the convolution.

If no mask is provided in both `test_image` and `refimage`, the returned image will be a plain numpy array (no mask).

1.1.2 Theory

General Assumptions

All of the methods assume we have a reference image R and a science image I that can be approximately modelled as:

$$I \approx R \otimes K + B_{kg}$$

for some background B_{kg} and some kernel K.

The optimal image subtraction D is then:

$$D = I - (R \otimes K + B_{kg})$$

The methods differ in their modelling of K.

Warning: In the ideal case of perfect subtraction, D should contain only noise and optical transients. In practice, tiny image misalignments, saturated stars and poor PSF fitting can leave subtraction artifacts near sources.

Methods

Alard & Lupton

Alard and Lupton [alard1998] introduced a method to simultaneously fit a background B_{kg} and a convolution kernel K that will minimize the difference between a reference image R and an another science image I.

$$I \approx R \otimes K + B_{kg}$$

This method assumes that the convolution kernel can be approximated by a linear combination of fixed Gaussians, where the linear coefficients are left free to vary. In addition, each Gaussian is modulated with a polynomial of an arbitrary degree.

$$
\begin{aligned}
K &= \sum_i a_i B_i \\
&= \sum_n a_n \times \left[\exp\left(-\frac{(u-u_0)^2}{2\sigma_u^2} + \frac{(v-v_0)^2}{2\sigma_v^2} \right) \sum_{d_n^x} \sum_{d_n^y} u^{d_n^x} v^{d_n^y} \right]
\end{aligned}
$$

Note: The centre, width and orientation of the Gaussians are fixed beforehand as well as the number of Gaussians to use in the expansion.

In OIS this is specified with a list of dictionaries, one for each gaussian we want to use. Below is an example of a basis with 3 Gaussians:

```
gausslist=[{center: (5, 5), sx: 2., sy: 2., modPolyDeg: 3},
           {sx: 1.0, sy: 2.5, modPolyDeg: 1},
           {sx: 3.0, sy: 1.0},]
```

Here `center` is the (row, column) of the center pixel of the Gaussian in the kernel. For example, for a kernel of shape `(11, 17)`, `center=(5, 8)` would yield a centered Gaussian. Center coordinates may be `float` values. If not specified, `center` defaults to the kernel's center.

The parameters `sx` and `sy` are the σ_u and σ_v of the Gaussian profile.

The degree of the 2D polynomial that modulates the Gaussian is set by the parameter `modPolyDeg`.

Bramich

The method developed in [bramich2008] modifies Alard-Lupton making each pixel of the kernel an independent parameter to fit. This is equivalent as having a vector basis consisting of *Delta kernels*. It can also simultaneously fit a polynomial background.

$$
K = \sum_n a_n \times \delta_{nn'}
$$

This method does not make assumptions on the kernel shape and can thus model completely arbitrary kernels. It can also correct for small translations between the images.

While more accurate, this method is computationally more expensive than *Alard & Lupton*'s.

Warning: Since each pixel is treated independently, a 11 by 11 kernel will have 121 free parameters just for the kernel. It grows quadratically with the kernel side. This needs to be taken in consideration for large kernels.

Adaptive Bramich

Like Bramich, this method also treats each pixel independently, but it will also multiply each pixel by a polynomial on the coordinates of the image.

This requires a special type of convolution where the kernel varies point to point in the image.

It is especially suited for situations where the PSF varies significatively across the image.

The method is described in more detail in [miller2008].

Warning: Just like Bramich, the number of free parameters scales quadratically with the kernel side. Furthermore, the degree of the polynomial multiplies the number of parameters by (deg + 1) * (deg + 2) / 2. This needs to be taken in consideration for large kernels.

1.1.3 Command-line program

The C command-line program *ois* is somewhat limited compared to its Python counterpart. It can only perform *Adaptive Bramich* method without simultanous background fit. For that reason we suggest removing the background on the images to be used.

OIS on the command line will read the reference and science images from FITS files on the file system. It will only output the difference image. It will not output the kernel or optimal image.

Note: To perform Bramich method instead of Adaptive Bramich, set the command line argument -kd to 0

Installation

To compile and execute the C command-line program:

```
$ git clone https://github.com/toros-astro/ois.git
$ cd ois
$ make ois
$ ./ois --help
```

Usage

```
$ ois -ks, --kernel-side <int> -kd, --kernel-poly-deg <int> -ref <filename> -sci
↪<filename> [-o <filename>] [-h, --help] [--version]
```

Command-line arguments:

-ks, --kernel-side
The side in pixels of the kernel to calculate the optimal difference. Must be an odd number.

-kd, --kernel-poly-deg
Degree of the interpolating polynomial for the variable kernel.

-ref
The reference image path.

-sci
The science image path.

-o
[Optional] The path where the subtraction FITS file will be written. Default value is "diff_img.fits".

-h, --help
Print usage help and exit.

--version
Print version information and exit.

1.1.4 Module API

OIS is a package to perform optimal image subtraction on astronomical images. It offers different methods to subtract images:

- Modulated multi-Gaussian kernel
- Delta basis kernel
- Adaptive Delta Basis kernel

Each method can (optionally) simultaneously fit and remove common background.

Usage:

```
>>> from ois import optimal_system
>>> difference, optimalImage, optimalKernel, background =
                                  optimal_system(image, referenceImage)
```

(c) Martin Beroiz email: <martinberoiz@gmail.com> University of Texas at San Antonio

exception ois.**EvenSideKernelError**

ois.**convolve2d_adaptive**(*image*, *kernel*, *poly_degree*)
Convolve image with the adaptive kernel of *poly_degree* degree.

ois.**eval_adpative_kernel**(*kernel*, *x*, *y*)
Return the adaptive kernel at position (x, y) = (col, row).

ois.**optimal_system**(*image*, *refimage*, *kernelshape=(11, 11)*, *bkgdegree=None*, *method='Bramich'*, *gridshape=None*, ***kwargs*)
Do Optimal Image Subtraction and return optimal image, kernel and background.

This is an implementation of a few Optimal Image Subtraction algorithms. They all (optionally) simultaneously fit a background.

Parameters

- **gridshape** – A tuple containing the number of vertical and horizontal divisions of a grid. Subtraction will be performed on each grid element. `None` is equivalent to a `(1, 1)` grid (no grid).
- **kernelshape** – Shape of the kernel to use. Must be of odd size.
- **bkgdegree** – Degree of the polynomial to fit the background. To turn off background fitting set this to `None`.
- **method** – One of the following strings.
  - `"Bramich"`: A Delta basis for the kernel (all pixels fit independently)
  - `"AdaptiveBramich"`: Same as Bramich, but with a polynomial variation across the image. It needs the parameter `poly_degree`, which is the polynomial degree of the variation.
  - `"Alard-Lupton"`: A modulated multi-Gaussian kernel. It needs the gausslist keyword.
- **poly_degree** – Needed only for AdaptiveBramich. It is the degree of the polynomial for the kernel spatial variation.

- **gausslist** – Needed only for Alard-Lupton. A list of dictionaries with info for the modulated multi-Gaussian. Dictionary keys are:
  - center: a (row, column) tuple for the center of the Gaussian. Default: kernel center.
  - modPolyDeg: the degree of the modulating polynomial. Default: 2
  - sx: sigma in x direction. Default: 2.
  - sy: sigma in y direction. Deafult: 2.

  All keys are optional. Example:

```
gausslist=[{center: (5, 5), sx: 2., sy: 2., modPolyDeg: 3},
           {sx: 1.0, sy: 2.5, modPolyDeg: 1},
           {sx: 3.0, sy: 1.0},]
```

Returns difference, optimal_image, kernel, background

Raises *EvenSideKernelError* – If any dimension of `kernelshape` is even.

Bibliography

[alard1998] "A Method for Optimal Image Subtraction" - C. Alard, R. H. Lupton, 1997.

[bramich2008] "A New Algorithm For Difference Image Analysis" - D.M. Bramich, 2008.

[miller2008] "Optimal Image Subtraction Method: Summary Derivations, Applications, and Publicly Shared Application Using IDL" - J. PATRICK MILLER et al., 2008.

Python Module Index

o

Index

Symbols

C

E

O