image$_{r}egistration Documentation$

Release 0.2.5.dev328

Adam Ginsburg

Nov 22, 2017

Contents

Github: Image Registration

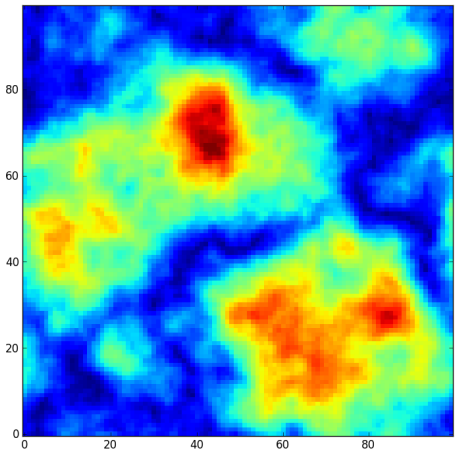
TL;DR version

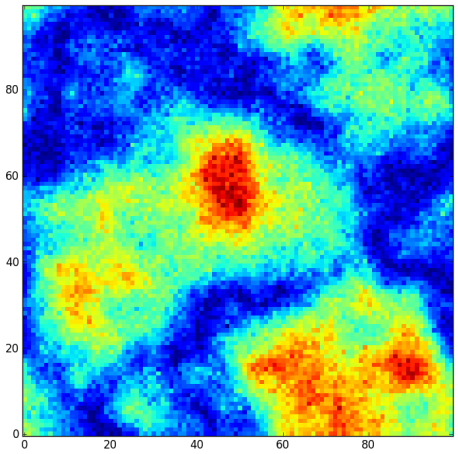
Use `chi2_shift` to match images.

A toolkit for registering images of astronomical images containing primarily extended flux (e.g., nebulae, radio and millimeter maps).*[0]

There are related packages scattered throughout the internet that do the same thing, but with different features.

The general goal is to align images that look kind of like these:





[0] Apparently astronomical images look a lot like microscopic images. So maybe this code is good for coaligning bacteria!

Part I

Quick Example

Given two images, shift the second to match the first and check that they are the same. This notebook also shows examples.

```
from image_registration import chi2_shift
from image_registration.fft_tools import shift
xoff, yoff, exoff, eyoff = chi2_shift(image, offset_image, noise,
...                                   return_error=True, upsample_factor='auto')
corrected_image2 = shift.shiftnd(offset_image, -yoff, -xoff)
```

Part II

Module APIs:

image_registration Package

fft_tools Package (and a description of the *Fourier Image Manipulation Tools*)

The most successful of the methods implemented here is `chi2_shift`

There is an ipython notebook demonstration of the code here and in pdf here

Part III

Related Methods

There are many other approaches to performing image registration. Some are summarized here. Note that this package is intended for image registration where the brightness is "extended" or "spread out" - stellar images are best to register by treating the stars as control points.

The methods below have various advantages and deficiencies. The most dangerous approach that should be avoided is that of fitting a gaussian to the peak of a cross-correlation image: this is the only other method that allows for measurements of the errors on the best-fit shift, but it is likely to be systematically wrong. The peak-fitting approach is unstable to fitting cross-correlated structure (which may be "beam-shaped") instead of the cross-correlation shift peak (which may have effectively no shape because it is sub-pixel).

The main advantage of the `chi2_shift` approach is that it can return *statistical errors* on the best-fit shift. It is also fast and efficient for many image types and sizes.

CHAPTER 1

Programs implementing these methods in various languages:

Varosi + Landsman astrolib correl_optimize :
Uses cross-correlation with "reduction" and "magnification" factors for speed and accuracy respectively; this method is relatively slow when using the complete information in the image (the magnification process increases the size of the image directly)

Brian Welsch's cross-cor taylor :
Uses the cross-correlation peak to measure the pixel peak of the offset, then does a 2nd order taylor-expansion around that peak to achieve sub-pixel accuracy. Is fast and generally quite accurate, but can be subject to bias.

Manuel Guizar's Efficient Sub-Pixel Registration :
A matlab version of the main method implemented in this code. Is fast and accurate. The speed comes from making use of the fourier zoom / fourier scaling property.

Marshall Perrin's sub-pixel image registration :
Implements many cross-correlation based methods, with sub-pixel registration based off of centroiding, gaussian fitting, and many variations thereupon. The gaussian approach is also implemented here, but is highly biased and inaccurate in general. As a sidenote, I tried using the "gaussian fit after high-pass filter" approach, but found that it really didn't work - it helped remove SOME of the large-scale junk, but it didn't end up changing the shape of the peak in a helpful way.

Christoph Gohlke's python fft image registration code :
Allows for rescaling and rotation.

Related bibliographic items (with no attached code): "Sub-pixel image registration with a maximum likelihood estimator" The method they describe is statistically equivalent to what I have implemented, though my method measures the *error* on the maximum-likelihood offset in addition to the ML offset.

This is an Astropy affiliated package.

Part IV

Indices and tables

- genindex
- modindex
- search
- *FITS_tools Package*
- *fft_tools Package*
- *image_registration Package*

CHAPTER 2

Spectral Cross-Correlation

Cross-correlation is frequently used to measure redshift by comparing a measured spectrum to a template. The Sloan survey made use of this technique.

However, the "standard" approach to determining errors

http://www.sdss.org/dr5/algorithms/spectemplates/spDR2-028.fit

CHAPTER 3

Fourier Image Manipulation Tools

There are a few interesting Fourier-based image manipulation tools implemented in this package.

3.1 Shift Theorem

The Fourier shift theorem allows an image to be shifted in any directions by an arbitrary amount, including sub-pixel shifts. It is more computationally efficient than most interpolation techniques (or at least, so I think). It is described well in this lecture and in the NRAO interferometry course.

$$FT[f(t-t_0)](x) = e^{-2\pi i x t_0} F(x)$$

The shift code is in `image_registration.fft_tools.shift`.

3.2 Similarity Theorem

The similarity theorem, or scale theorem, allows you to upsample timestreams. You cannot gain information below the image scale, but it is useful for getting sub-pixel information about gaussian peaks, for example. This NRAO lecture details the math.

$$f(ax) \Leftrightarrow \frac{F(s/a)}{|a|}$$

The zoom and upsample methods are in `image_registration.fft_tools.scale` and `image_registration.fft_tools.zoom`.

3.3 Resources

I made use of a lot of not particularly easy to find documents when writing this code.

- Image Resampling by Neil Dodgson is a book

- Comparison of Interpolation Methods for Image Resampling
- The Similarity Theorem

As stated elsewhere, though, the main inspiration for this work was Manuel Guizar's DFT upsampling technique

CHAPTER 4

image_registration Package

4.1 image_registration Package

4.2 image_registration Package

This is an Astropy affiliated package.

4.2.1 Functions

cross_correlation_shifts(image1, image2[, ...])	Use cross-correlation and a 2nd order taylor expansion to measure the
cross_correlation_shifts_FITS(fitsfile1, ...)	Determine the shift between two FITS images using the cross-correlation technique.
register_images(im1, im2[, usfac, ...])	Sub-pixel image registration (see dftregistration for lots of details)
test([package, test_path, args, plugins, ...])	Run the tests using py.test.

chi2_shift

image_registration.**chi2_shift**(*im1*, *im2*, *err=None*, *upsample_factor='auto'*, *boundary='wrap'*, *nthreads=1*, *use_numpy_fft=False*, *zeromean=False*, *nfitted=2*, *verbose=False*, *return_error=True*, *return_chi2array=False*, *max_auto_size=512*, *max_nsig=1.1*)

Find the offsets between image 1 and image 2 using the DFT upsampling method (http://www.mathworks.com/matlabcentral/fileexchange/18401-efficient-subpixel-image-registration-by-cross-correlation/content/html/

efficient_subpixel_registration.html) combined with χ^2 to measure the errors on the fit

Equation 1 gives the χ^2 value as a function of shift, where Y is the model as a function of shift:

$$\chi^2(dx,dy) = \Sigma_{ij} \frac{(X_{ij} - Y_{ij}(dx,dy))^2}{\sigma_{ij}^2}$$

Equation 2-4: blahha

$$\text{Term 1 : } f(dx,dy) = \Sigma_{ij} \frac{X_{ij}^2}{\sigma_{ij}^2} \tag{4.1}$$

$$f(dx,dy) = f(0,0), \forall dx, dy \tag{4.2}$$

$$\text{Term 2 : } g(dx,dy) = -2\Sigma_{ij} \frac{X_{ij} Y_{ij}(dx,dy)}{\sigma_{ij}^2} = -2\Sigma_{ij} \left(\frac{X_{ij}}{\sigma_{ij}^2}\right) Y_{ij}(dx,dy) \tag{4.3}$$

$$\text{Term 3 : } h(dx,dy) = \Sigma_{ij} \frac{Y_{ij}(dx,dy)^2}{\sigma_{ij}^2} = \Sigma_{ij} \left(\frac{1}{\sigma_{ij}^2}\right) Y_{ij}^2(dx,dy) \tag{4.4}$$

The cross-correlation can be computed with fourier transforms, and is defined

$$CC_{m,n}(x,y) = \Sigma_{ij} x_{ij}^* y_{(n+i)(m+j)}$$

which can then be applied to our problem, noting that the cross-correlation has the same form as term 2 and 3 in χ^2 (term 1 is a constant, with no dependence on the shift)

$$\text{Term 2 : } CC(X/\sigma^2, Y)[dx,dy] = \Sigma_{ij} \left(\frac{X_{ij}}{\sigma_{ij}^2}\right)^* Y_{ij}(dx,dy) \tag{4.5}$$

$$\text{Term 3 : } CC(\sigma^{-2}, Y^2)[dx,dy] = \Sigma_{ij} \left(\frac{1}{\sigma_{ij}^2}\right)^* Y_{ij}^2(dx,dy) \tag{4.6}$$

Technically, only terms 2 and 3 has any effect on the resulting image, since term 1 is the same for all shifts, and the quantity of interest is $\Delta\chi^2$ when determining the best-fit shift and error.

Parameters

im1 : np.ndarray

im2 : np.ndarray

The images to register.

err : np.ndarray

Per-pixel error in image 2

boundary : 'wrap','constant','reflect','nearest'

Option to pass to map_coordinates for determining what to do with shifts outside of the boundaries.

upsample_factor : int or 'auto'

upsampling factor; governs accuracy of fit (1/usfac is best accuracy) (can be "automatically" determined based on chi^2 error)

return_error : bool

Returns the "fit error" (1-sigma in x and y) based on the delta-chi2 values

return_chi2_array : bool

Returns the x and y shifts and the chi2 as a function of those shifts in addition to other returned parameters. i.e., the last return from this function will be a tuple (x, y, chi2)

zeromean : bool

Subtract the mean from the images before cross-correlating? If no, you may get a 0,0 offset because the DC levels are strongly correlated.

verbose : bool

Print error message if upsampling factor is inadequate to measure errors

use_numpy_fft : bool

Force use numpy's fft over fftw? (only matters if you have fftw installed)

nthreads : bool

Number of threads to use for fft (only matters if you have fftw installed)

nfitted : int

number of degrees of freedom in the fit (used for chi^2 computations). Should probably always be 2.

max_auto_size : int

Maximum zoom image size to create when using auto-upsampling

Returns

dx,dy : float,float

Measures the amount im2 is offset from im1 (i.e., shift im2 by -1 * these #'s to match im1)

errx,erry : float,float

optional, error in x and y directions

xvals,yvals,chi2n_upsampled : ndarray,ndarray,ndarray,

x,y positions (in original chi^2 coordinates) of the chi^2 values and their corresponding chi^2 value

Examples

Create a 2d array, shift it in both directions, then use chi2_shift to determine the shift

```
>>> import image_registration
>>> rr = ((np.indices([100,100]) - np.array([50.,50.])[:,None,None])**2).sum(axis=0)**0.5
>>> image = np.exp(-rr**2/(3.**2*2.)) * 20
>>> shifted = np.roll(np.roll(image,12,0),5,1) + np.random.randn(100,100)
>>> dx,dy,edx,edy = chi2_shift(image, shifted, upsample_factor='auto')
>>> shifted2 = image_registration.fft_tools.shift2d(image,3.665,-4.25) + np.random.randn(100,100)
>>> dx2,dy2,edx2,edy2 = chi2_shift(image, shifted2, upsample_factor='auto')
```

chi2_shift_iterzoom

image_registration.**chi2_shift_iterzoom**(*im1, im2, err=None, upsample_factor='auto', boundary='wrap', nthreads=1, use_numpy_fft=False, zeromean=False, verbose=False, return_error=True, return_chi2array=False, zoom_shape=[10, 10], rezoom_shape=[100, 100], rezoom_factor=5, mindiff=1, **kwargs*)

Find the offsets between image 1 and image 2 using an iterative DFT upsampling method combined with χ^2 to measure the errors on the fit

A simpler version of `chi2_shift()` that only computes the χ^2 array on the largest scales, then uses a fourier upsampling technique to zoom in.

Parameters

im1 : np.ndarray

im2 : np.ndarray

The images to register.

err : np.ndarray

Per-pixel error in image 2

boundary : 'wrap','constant','reflect','nearest'

Option to pass to map_coordinates for determining what to do with shifts outside of the boundaries.

upsample_factor : int or 'auto'

upsampling factor; governs accuracy of fit (1/usfac is best accuracy) (can be "automatically" determined based on chi^2 error)

zeromean : bool

Subtract the mean from the images before cross-correlating? If no, you may get a 0,0 offset because the DC levels are strongly correlated.

verbose : bool

Print error message if upsampling factor is inadequate to measure errors

use_numpy_fft : bool

Force use numpy's fft over fftw? (only matters if you have fftw installed)

nthreads : bool

Number of threads to use for fft (only matters if you have fftw installed)

nfitted : int

number of degrees of freedom in the fit (used for chi^2 computations). Should probably always be 2.

zoom_shape : [int,int]

Shape of iterative zoom image

rezoom_shape : [int,int]

Shape of the final output chi^2 map to use for determining the errors

rezoom_factor : int

Amount to zoom above the last zoom factor. Should be <= rezoom_shape/zoom_shape

Returns

dx,dy : float,float

Measures the amount im2 is offset from im1 (i.e., shift im2 by -1 * these #'s to match im1)

errx,erry : float,float

optional, error in x and y directions

xvals,yvals,chi2n_upsampled : ndarray,ndarray,ndarray,

x,y positions (in original chi^2 coordinates) of the chi^2 values and their corresponding chi^2 value

Other Parameters

return_error : bool

Returns the "fit error" (1-sigma in x and y) based on the delta-chi2 values

return_chi2_array : bool

Returns the x and y shifts and the chi2 as a function of those shifts in addition to other returned parameters. i.e., the last return from this function will be a tuple (x, y, chi2)

Examples

Create a 2d array, shift it in both directions, then use chi2_shift_iterzoom to determine the shift

```
>>> import image_registration
>>> np.random.seed(42) # so the doctest will pass
>>> image = np.random.randn(50,55)
>>> shifted = np.roll(np.roll(image,12,0),5,1)
>>> dx,dy,edx,edy = chi2_shift_iterzoom(image, shifted, upsample_factor='auto')
>>> shifted2 = image_registration.fft_tools.shift2d(image,3.665,-4.25)
>>> dx2,dy2,edx2,edy2 = chi2_shift_iterzoom(image, shifted2, upsample_factor='auto')
```

chi2n_map

image_registration.**chi2n_map**(*im1*, *im2*, *err=None*, *boundary='wrap'*, *nthreads=1*, *zeromean=False*, *use_numpy_fft=False*, *return_all=False*, *reduced=False*)

Parameters

im1 : np.ndarray

im2 : np.ndarray

The images to register.

err : np.ndarray

Per-pixel error in image 2

boundary : 'wrap','constant','reflect','nearest'

Option to pass to map_coordinates for determining what to do with shifts outside of the boundaries.

zeromean : bool

Subtract the mean from the images before cross-correlating? If no, you may get a 0,0 offset because the DC levels are strongly correlated.

nthreads : bool

Number of threads to use for fft (only matters if you have fftw installed)

reduced : bool

Return the reduced χ^2 array, or unreduced? (assumes 2 degrees of freedom for the fit)

Returns

chi2n : np.ndarray

the χ^2 array

term1 : float

Scalar, term 1 in the χ^2 equation

term2 : np.ndarray

Term 2 in the equation, -2 * cross-correlation(x/sigma^2,y)

term3 : np.ndarray | float

If error is an array, returns an array, otherwise is a scalar float corresponding to term 3 in the equation

cross_correlation_shifts

image_registration.**cross_correlation_shifts**(*image1*, *image2*, *errim1=None*, *errim2=None*, *maxoff=None*, *verbose=False*, *gaussfit=False*, *return_error=False*, *zeromean=True*, ***kwargs*)

Use cross-correlation and a 2nd order taylor expansion to measure the offset between two images

Given two images, calculate the amount image2 is offset from image1 to sub-pixel accuracy using 2nd order taylor expansion.

Parameters

image1: np.ndarray

The reference image

image2: np.ndarray

The offset image. Must have the same shape as image1

errim1: np.ndarray [optional]

The pixel-by-pixel error on the reference image

errim2: np.ndarray [optional]

The pixel-by-pixel error on the offset image.

maxoff: int

Maximum allowed offset (in pixels). Useful for low s/n images that you know are reasonably well-aligned, but might find incorrect offsets due to edge noise

zeromean : bool

Subtract the mean from each image before performing cross-correlation?

verbose: bool

Print out extra messages?

gaussfit : bool

Use a Gaussian fitter to fit the peak of the cross-correlation?

return_error: bool

Return an estimate of the error on the shifts. WARNING: I still don't understand how to make these agree with simulations. The analytic estimate comes from http://adsabs.harvard.edu/abs/2003MNRAS.342.1291Z At high signal-to-noise, the analytic version overestimates the error by a factor of about 1.8, while the gaussian version overestimates error by about 1.15. At low s/n, they both UNDERestimate the error. The transition zone occurs at a *total* S/N ~ 1000 (i.e., the total signal in the map divided by the standard deviation of the map - it depends on how many pixels have signal)

****kwargs are passed to correlate2d, which in turn passes them to convolve.**

The available options include image padding for speed and ignoring NaNs.

References

From http://solarmuri.ssl.berkeley.edu/~welsch/public/software/cross_cor_taylor.pro

Examples

```
>>> import numpy as np
>>> im1 = np.zeros([10,10])
>>> im2 = np.zeros([10,10])
>>> im1[4,3] = 1
>>> im2[5,5] = 1
>>> import image_registration
>>> yoff,xoff = image_registration.cross_correlation_shifts(im1,im2)
>>> im1_aligned_to_im2 = np.roll(np.roll(im1,int(yoff),1),int(xoff),0)
>>> assert (im1_aligned_to_im2-im2).sum() == 0
```

cross_correlation_shifts_FITS

image_registration.**cross_correlation_shifts_FITS**(*fitsfile1*, *fitsfile2*, *return_cropped_images=False*, *quiet=True*, *sigma_cut=False*, *register_method=<function cross_correlation_shifts>*, ***kwargs*)

Determine the shift between two FITS images using the cross-correlation technique. Requires reproject

Parameters

fitsfile1: str

Reference fits file name

fitsfile2: str

Offset fits file name

return_cropped_images: bool

Returns the images used for the analysis in addition to the measured offsets

quiet: bool

Silence messages?

sigma_cut: bool or int

Perform a sigma-cut before cross-correlating the images to minimize noise correlation?

register_images

image_registration.**register_images**(*im1*, *im2*, *usfac=1*, *return_registered=False*, *return_error=False*, *zeromean=True*, *DEBUG=False*, *maxoff=None*, *nthreads=1*, *use_numpy_fft=False*)

Sub-pixel image registration (see dftregistration for lots of details)

Parameters

im1 : np.ndarray

im2 : np.ndarray

The images to register.

usfac : int

upsampling factor; governs accuracy of fit (1/usfac is best accuracy)

return_registered : bool

Return the registered image as the last parameter

return_error : bool

Does nothing at the moment, but in principle should return the "fit error" (it does nothing because I don't know how to compute the "fit error")

zeromean : bool

Subtract the mean from the images before cross-correlating? If no, you may get a 0,0 offset because the DC levels are strongly correlated.

maxoff : int

Maximum allowed offset to measure (setting this helps avoid spurious peaks)

DEBUG : bool

Test code used during development. Should DEFINITELY be removed.

Returns

dx,dy : float,float

REVERSE of dftregistration order (also, signs flipped) for consistency with other routines. Measures the amount im2 is offset from im1 (i.e., shift im2 by these #'s to match im1)

test

image_registration.**test**(*package=None*, *test_path=None*, *args=None*, *plugins=None*, *verbose=False*, *pastebin=None*, *remote_data=False*, *pep8=False*, *pdb=False*, *coverage=False*, *open_files=False*, ***kwargs*)

Run the tests using py.test. A proper set of arguments is constructed and passed to pytest.main.

Parameters

package : str, optional

The name of a specific package to test, e.g. 'io.fits' or 'utils'. If nothing is specified all default tests are run.

test_path : str, optional

Specify location to test by path. May be a single file or directory. Must be specified absolutely or relative to the calling directory.

args : str, optional

Additional arguments to be passed to pytest.main in the `args` keyword argument.

plugins : list, optional

Plugins to be passed to pytest.main in the `plugins` keyword argument.

verbose : bool, optional

Convenience option to turn on verbose output from py.test. Passing True is the same as specifying `'-v'` in `args`.

pastebin : {'failed','all',None}, optional

Convenience option for turning on py.test pastebin output. Set to `'failed'` to upload info for failed tests, or `'all'` to upload info for all tests.

remote_data : bool, optional

Controls whether to run tests marked with @remote_data. These tests use online data and are not run by default. Set to True to run these tests.

pep8 : bool, optional

Turn on PEP8 checking via the pytest-pep8 plugin and disable normal tests. Same as specifying `'--pep8 -k pep8'` in `args`.

pdb : bool, optional

Turn on PDB post-mortem analysis for failing tests. Same as specifying `'--pdb'` in `args`.

coverage : bool, optional

Generate a test coverage report. The result will be placed in the directory htmlcov.

open_files : bool, optional

Fail when any tests leave files open. Off by default, because this adds extra run time to the test suite. Requires the psutil package.

parallel : int, optional

When provided, run the tests in parallel on the specified number of CPUs. If parallel is negative, it will use the all the cores on the machine. Requires the pytest-xdist plugin installed. Only available when using Astropy 0.3 or later.

kwargs

Any additional keywords passed into this function will be passed on to the astropy test runner. This allows use of test-related functionality implemented in later versions of astropy without explicitly updating the package template.

Continued on next page

Table 4.2 – continued from previous page

4.3 register_images Module

4.4 image_registration.register_images Module

4.4.1 Functions

register_images(im1, im2[, usfac, ...])	Sub-pixel image registration (see dftregistration for lots of details)

register_images

image_registration.register_images.**register_images**(*im1*, *im2*, *usfac=1*, *return_registered=False*, *return_error=False*, *zeromean=True*, *DEBUG=False*, *maxoff=None*, *nthreads=1*, *use_numpy_fft=False*)

Sub-pixel image registration (see dftregistration for lots of details)

Parameters

im1 : np.ndarray

im2 : np.ndarray

The images to register.

usfac : int

upsampling factor; governs accuracy of fit (1/usfac is best accuracy)

return_registered : bool

Return the registered image as the last parameter

return_error : bool

Does nothing at the moment, but in principle should return the "fit error" (it does nothing because I don't know how to compute the "fit error")

zeromean : bool

Subtract the mean from the images before cross-correlating? If no, you may get a 0,0 offset because the DC levels are strongly correlated.

maxoff : int

Maximum allowed offset to measure (setting this helps avoid spurious peaks)

DEBUG : bool

Test code used during development. Should DEFINITELY be removed.

Returns

dx,dy : float,float

REVERSE of dftregistration order (also, signs flipped) for consistency with other routines. Measures the amount im2 is offset from im1 (i.e., shift im2 by these #'s to match im1)

Sub-pixel image registration (see dftregistration for lots of details)

Parameters

im1 : np.ndarray

im2 : np.ndarray

The images to register.

usfac : int

upsampling factor; governs accuracy of fit (1/usfac is best accuracy)

return_registered : bool

Return the registered image as the last parameter

return_error : bool

Does nothing at the moment, but in principle should return the "fit error" (it does nothing because I don't know how to compute the "fit error")

zeromean : bool

Subtract the mean from the images before cross-correlating? If no, you may get a 0,0 offset because the DC levels are strongly correlated.

maxoff : int

Maximum allowed offset to measure (setting this helps avoid spurious peaks)

DEBUG : bool

Test code used during development. Should DEFINITELY be removed.

Returns

dx,dy : float,float

REVERSE of dftregistration order (also, signs flipped) for consistency with other routines. Measures the amount im2 is offset from im1 (i.e., shift im2 by these #'s to match im1)

4.5 chi2_shifts Module

4.6 image_registration.chi2_shifts Module

4.6.1 Chi^2 shifts

Various tools for calculating shifts based on the chi^2 method

4.6.2 Functions

chi2_shift

image_registration.chi2_shifts.**chi2_shift**(*im1*, *im2*, *err=None*, *upsample_factor='auto'*, *boundary='wrap'*, *nthreads=1*, *use_numpy_fft=False*, *zeromean=False*, *nfitted=2*, *verbose=False*, *return_error=True*, *return_chi2array=False*, *max_auto_size=512*, *max_nsig=1.1*)

Find the offsets between image 1 and image 2 using the DFT upsampling method (http://www.mathworks.com/matlabcentral/fileexchange/18401-efficient-subpixel-image-registration-by-cross-correlation/content/html/efficient_subpixel_registration.html) combined with χ^2 to measure the errors on the fit

Equation 1 gives the χ^2 value as a function of shift, where Y is the model as a function of shift:

$$\chi^2(dx,dy) = \Sigma_{ij} \frac{(X_{ij} - Y_{ij}(dx,dy))^2}{\sigma_{ij}^2}$$

Equation 2-4: blahha

$$\text{Term 1 }: f(dx,dy) = \Sigma_{ij} \frac{X_{ij}^2}{\sigma_{ij}^2} \tag{4.7}$$

$$f(dx,dy) = f(0,0), \forall dx,dy \tag{4.8}$$

$$\text{Term 2 }: g(dx,dy) = -2\Sigma_{ij} \frac{X_{ij} Y_{ij}(dx,dy)}{\sigma_{ij}^2} = -2\Sigma_{ij} \left(\frac{X_{ij}}{\sigma_{ij}^2}\right) Y_{ij}(dx,dy) \tag{4.9}$$

$$\text{Term 3 }: h(dx,dy) = \Sigma_{ij} \frac{Y_{ij}(dx,dy)^2}{\sigma_{ij}^2} = \Sigma_{ij} \left(\frac{1}{\sigma_{ij}^2}\right) Y_{ij}^2(dx,dy) \tag{4.10}$$

The cross-correlation can be computed with fourier transforms, and is defined

$$CC_{m,n}(x,y) = \Sigma_{ij} x_{ij}^* y_{(n+i)(m+j)}$$

which can then be applied to our problem, noting that the cross-correlation has the same form as term 2 and 3 in χ^2 (term 1 is a constant, with no dependence on the shift)

$$\text{Term 2 }: CC(X/\sigma^2, Y)[dx,dy] = \Sigma_{ij} \left(\frac{X_{ij}}{\sigma_{ij}^2}\right)^* Y_{ij}(dx,dy) \tag{4.11}$$

$$\text{Term 3 }: CC(\sigma^{-2}, Y^2)[dx,dy] = \Sigma_{ij} \left(\frac{1}{\sigma_{ij}^2}\right)^* Y_{ij}^2(dx,dy) \tag{4.12}$$

Technically, only terms 2 and 3 has any effect on the resulting image, since term 1 is the same for all shifts, and the quantity of interest is $\Delta\chi^2$ when determining the best-fit shift and error.

Parameters

im1 : np.ndarray

im2 : np.ndarray

The images to register.

err : np.ndarray

Per-pixel error in image 2

boundary : 'wrap','constant','reflect','nearest'

Option to pass to map_coordinates for determining what to do with shifts outside of the boundaries.

upsample_factor : int or 'auto'

upsampling factor; governs accuracy of fit (1/usfac is best accuracy) (can be "automatically" determined based on chi^2 error)

return_error : bool

Returns the "fit error" (1-sigma in x and y) based on the delta-chi2 values

return_chi2_array : bool

Returns the x and y shifts and the chi2 as a function of those shifts in addition to other returned parameters. i.e., the last return from this function will be a tuple (x, y, chi2)

zeromean : bool

Subtract the mean from the images before cross-correlating? If no, you may get a 0,0 offset because the DC levels are strongly correlated.

verbose : bool

Print error message if upsampling factor is inadequate to measure errors

use_numpy_fft : bool

Force use numpy's fft over fftw? (only matters if you have fftw installed)

nthreads : bool

Number of threads to use for fft (only matters if you have fftw installed)

nfitted : int

number of degrees of freedom in the fit (used for chi^2 computations). Should probably always be 2.

max_auto_size : int

Maximum zoom image size to create when using auto-upsampling

Returns

dx,dy : float,float

Measures the amount im2 is offset from im1 (i.e., shift im2 by -1 * these #'s to match im1)

errx,erry : float,float

optional, error in x and y directions

xvals,yvals,chi2n_upsampled : ndarray,ndarray,ndarray,

x,y positions (in original chi^2 coordinates) of the chi^2 values and their corresponding chi^2 value

Examples

Create a 2d array, shift it in both directions, then use chi2_shift to determine the shift

```
>>> import image_registration
>>> rr = ((np.indices([100,100]) - np.array([50.,50.])[:,None,None])**2).sum(axis=0)**0.5
>>> image = np.exp(-rr**2/(3.**2*2.)) * 20
>>> shifted = np.roll(np.roll(image,12,0),5,1) + np.random.randn(100,100)
>>> dx,dy,edx,edy = chi2_shift(image, shifted, upsample_factor='auto')
```

```
>>> shifted2 = image_registration.fft_tools.shift2d(image,3.665,-4.25) + np.random.randn(100,100)
>>> dx2,dy2,edx2,edy2 = chi2_shift(image, shifted2, upsample_factor='auto')
```

chi2_shift_iterzoom

image_registration.chi2_shifts.**chi2_shift_iterzoom**(*im1, im2, err=None, upsample_factor='auto', boundary='wrap', nthreads=1, use_numpy_fft=False, zeromean=False, verbose=False, return_error=True, return_chi2array=False, zoom_shape=[10, 10], rezoom_shape=[100, 100], rezoom_factor=5, mindiff=1, **kwargs*)

Find the offsets between image 1 and image 2 using an iterative DFT upsampling method combined with χ^2 to measure the errors on the fit

A simpler version of `chi2_shift()` that only computes the χ^2 array on the largest scales, then uses a fourier upsampling technique to zoom in.

Parameters

im1 : np.ndarray

im2 : np.ndarray

The images to register.

err : np.ndarray

Per-pixel error in image 2

boundary : 'wrap','constant','reflect','nearest'

Option to pass to map_coordinates for determining what to do with shifts outside of the boundaries.

upsample_factor : int or 'auto'

upsampling factor; governs accuracy of fit (1/usfac is best accuracy) (can be "automatically" determined based on chi^2 error)

zeromean : bool

Subtract the mean from the images before cross-correlating? If no, you may get a 0,0 offset because the DC levels are strongly correlated.

verbose : bool

Print error message if upsampling factor is inadequate to measure errors

use_numpy_fft : bool

Force use numpy's fft over fftw? (only matters if you have fftw installed)

nthreads : bool

Number of threads to use for fft (only matters if you have fftw installed)

nfitted : int

number of degrees of freedom in the fit (used for chi^2 computations). Should probably always be 2.

zoom_shape : [int,int]

Shape of iterative zoom image

rezoom_shape : [int,int]

Shape of the final output chi^2 map to use for determining the errors

rezoom_factor : int

Amount to zoom above the last zoom factor. Should be <= rezoom_shape/zoom_shape

Returns

dx,dy : float,float

Measures the amount im2 is offset from im1 (i.e., shift im2 by -1 * these #'s to match im1)

errx,erry : float,float

optional, error in x and y directions

xvals,yvals,chi2n_upsampled : ndarray,ndarray,ndarray,

x,y positions (in original chi^2 coordinates) of the chi^2 values and their corresponding chi^2 value

Other Parameters

return_error : bool

Returns the "fit error" (1-sigma in x and y) based on the delta-chi2 values

return_chi2_array : bool

Returns the x and y shifts and the chi2 as a function of those shifts in addition to other returned parameters. i.e., the last return from this function will be a tuple (x, y, chi2)

Examples

Create a 2d array, shift it in both directions, then use chi2_shift_iterzoom to determine the shift

```
>>> import image_registration
>>> np.random.seed(42) # so the doctest will pass
>>> image = np.random.randn(50,55)
>>> shifted = np.roll(np.roll(image,12,0),5,1)
>>> dx,dy,edx,edy = chi2_shift_iterzoom(image, shifted, upsample_factor='auto')
>>> shifted2 = image_registration.fft_tools.shift2d(image,3.665,-4.25)
>>> dx2,dy2,edx2,edy2 = chi2_shift_iterzoom(image, shifted2, upsample_factor='auto')
```

chi2n_map

image_registration.chi2_shifts.**chi2n_map**(*im1*, *im2*, *err=None*, *boundary='wrap'*, *nthreads=1*, *zeromean=False*, *use_numpy_fft=False*, *return_all=False*, *reduced=False*)

Parameters

im1 : np.ndarray

im2 : np.ndarray

The images to register.

err : np.ndarray

Per-pixel error in image 2

boundary : 'wrap','constant','reflect','nearest'

Option to pass to map_coordinates for determining what to do with shifts outside of the boundaries.

zeromean : bool

Subtract the mean from the images before cross-correlating? If no, you may get a 0,0 offset because the DC levels are strongly correlated.

nthreads : bool

Number of threads to use for fft (only matters if you have fftw installed)

reduced : bool

Return the reduced χ^2 array, or unreduced? (assumes 2 degrees of freedom for the fit)

Returns

chi2n : np.ndarray

the χ^2 array

term1 : float

Scalar, term 1 in the χ^2 equation

term2 : np.ndarray

Term 2 in the equation, -2 * cross-correlation(x/sigma^2,y)

term3 : np.ndarray | float

If error is an array, returns an array, otherwise is a scalar float corresponding to term 3 in the equation

4.7 cross_correlation_shifts Module

4.8 image_registration.cross_correlation_shifts Module

4.8.1 Functions

cross_correlation_shifts(image1, image2[, ...])	Use cross-correlation and a 2nd order taylor expansion to measure the
cross_correlation_shifts_FITS(fitsfile1, ...)	Determine the shift between two FITS images using the cross-correlation technique.

cross_correlation_shifts

image_registration.cross_correlation_shifts.**cross_correlation_shifts**(*image1*, *image2*, *errim1=None*, *errim2=None*, *maxoff=None*, *verbose=False*, *gaussfit=False*, *return_error=False*, *zeromean=True*, ***kwargs*)

Use cross-correlation and a 2nd order taylor expansion to measure the offset between two images

Given two images, calculate the amount image2 is offset from image1 to sub-pixel accuracy using 2nd order taylor expansion.

Parameters

image1: np.ndarray

The reference image

image2: np.ndarray

The offset image. Must have the same shape as image1

errim1: np.ndarray [optional]

The pixel-by-pixel error on the reference image

errim2: np.ndarray [optional]

The pixel-by-pixel error on the offset image.

maxoff: int

Maximum allowed offset (in pixels). Useful for low s/n images that you know are reasonably well-aligned, but might find incorrect offsets due to edge noise

zeromean : bool

Subtract the mean from each image before performing cross-correlation?

verbose: bool

Print out extra messages?

gaussfit : bool

Use a Gaussian fitter to fit the peak of the cross-correlation?

return_error: bool

Return an estimate of the error on the shifts. WARNING: I still don't understand how to make these agree with simulations. The analytic estimate comes from http://adsabs.harvard.edu/abs/2003MNRAS.342.1291Z At high signal-to-noise, the analytic version overestimates the error by a factor of about 1.8, while the gaussian version overestimates error by about 1.15. At low s/n, they both UNDERestimate the error. The transition zone occurs at a *total* S/N ~ 1000 (i.e., the total signal in the map divided by the standard deviation of the map - it depends on how many pixels have signal)

****kwargs are passed to correlate2d, which in turn passes them to convolve.**

The available options include image padding for speed and ignoring NaNs.

References

From http://solarmuri.ssl.berkeley.edu/~welsch/public/software/cross_cor_taylor.pro

Examples

```
>>> import numpy as np
>>> im1 = np.zeros([10,10])
>>> im2 = np.zeros([10,10])
>>> im1[4,3] = 1
>>> im2[5,5] = 1
>>> import image_registration
>>> yoff,xoff = image_registration.cross_correlation_shifts(im1,im2)
>>> im1_aligned_to_im2 = np.roll(np.roll(im1,int(yoff),1),int(xoff),0)
>>> assert (im1_aligned_to_im2-im2).sum() == 0
```

cross_correlation_shifts_FITS

image_registration.cross_correlation_shifts.**cross_correlation_shifts_FITS**(*fitsfile1*, *fitsfile2*, *return_cropped_images=False*, *quiet=True*, *sigma_cut=False*, *register_method=<function cross_correlation_shifts>*, ***kwargs*)

Determine the shift between two FITS images using the cross-correlation technique. Requires reproject

Parameters

fitsfile1: str

Reference fits file name

fitsfile2: str

Offset fits file name

return_cropped_images: bool

Returns the images used for the analysis in addition to the measured offsets

quiet: bool

Silence messages?

sigma_cut: bool or int

Perform a sigma-cut before cross-correlating the images to minimize noise correlation?

image_registration.cross_correlation_shifts.**cross_correlation_shifts**(*image1*, *image2*, *errim1=None*, *errim2=None*, *maxoff=None*, *verbose=False*, *gaussfit=False*, *return_error=False*, *zeromean=True*, ***kwargs*)

Use cross-correlation and a 2nd order taylor expansion to measure the offset between two images

Given two images, calculate the amount image2 is offset from image1 to sub-pixel accuracy using 2nd order taylor expansion.

Parameters

image1: np.ndarray

The reference image

image2: np.ndarray

The offset image. Must have the same shape as image1

errim1: np.ndarray [optional]

The pixel-by-pixel error on the reference image

errim2: np.ndarray [optional]

The pixel-by-pixel error on the offset image.

maxoff: int

Maximum allowed offset (in pixels). Useful for low s/n images that you know are reasonably well-aligned, but might find incorrect offsets due to edge noise

zeromean : bool

Subtract the mean from each image before performing cross-correlation?

verbose: bool

Print out extra messages?

gaussfit : bool

Use a Gaussian fitter to fit the peak of the cross-correlation?

return_error: bool

Return an estimate of the error on the shifts. WARNING: I still don't understand how to make these agree with simulations. The analytic estimate comes from http://adsabs.harvard.edu/abs/2003MNRAS.342.1291Z At high signal-to-noise, the analytic version overestimates the error by a factor of about 1.8, while the gaussian version overestimates error by about 1.15. At low s/n, they both UNDERestimate the error. The transition zone occurs at a *total* S/N ~ 1000 (i.e., the total signal in the map divided by the standard deviation of the map - it depends on how many pixels have signal)

****kwargs are passed to correlate2d, which in turn passes them to convolve.**

The available options include image padding for speed and ignoring NaNs.

References

From http://solarmuri.ssl.berkeley.edu/~welsch/public/software/cross_cor_taylor.pro

Examples

```
>>> import numpy as np
>>> im1 = np.zeros([10,10])
>>> im2 = np.zeros([10,10])
>>> im1[4,3] = 1
>>> im2[5,5] = 1
>>> import image_registration
>>> yoff,xoff = image_registration.cross_correlation_shifts(im1,im2)
>>> im1_aligned_to_im2 = np.roll(np.roll(im1,int(yoff),1),int(xoff),0)
>>> assert (im1_aligned_to_im2-im2).sum() == 0
```

4.9 Subpackages

4.9.1 fft_tools Package

fft_tools Package

image_registration.fft_tools Package

Functions

correlate2d(im1, im2[, boundary])	Cross-correlation of two images of arbitrary size.
dftups(inp[, nor, noc, usfac, roff, coff])	Translated from matlab:
get_ffts([nthreads, use_numpy_fft])	Returns fftn,ifftn using either numpy's fft or fftw
shift2d(data, deltax, deltay[, phase, ...])	2D version: obsolete - use ND version instead
shiftnd(data, offset[, phase, nthreads, ...])	FFT-based sub-pixel image shift.
smooth(image[, kernelwidth, kerneltype, ...])	Returns a smoothed image using a gaussian, boxcar, or tophat kernel
upsample_image(image[, upsample_factor, ...])	Use dftups to upsample an image (but takes an image and returns an image with all reals)

correlate2d

image_registration.fft_tools.**correlate2d**(*im1*, *im2*, *boundary='wrap'*, ***kwargs*)

Cross-correlation of two images of arbitrary size. Returns an image cropped to the largest of each dimension of the input images

Parameters

return_fft - if true, return fft(im1)*fft(im2[::-1,::-1]), which is the power

spectral density

fftshift - if true, return the shifted psd so that the DC component is in

the center of the image

pad - Default on. Zero-pad image to the nearest 2^n

crop - Default on. Return an image of the size of the largest input image.

If the images are asymmetric in opposite directions, will return the largest image in both directions.

boundary: str, optional

A flag indicating how to handle boundaries:

- **'fill'**
  [set values outside the array boundary to fill_value] (default)
- 'wrap' : periodic boundary

WARNING: Normalization may be arbitrary if you use the PSD

dftups

image_registration.fft_tools.**dftups**(*inp*, *nor=None*, *noc=None*, *usfac=1*, *roff=0*, *coff=0*)

Translated from matlab:

- Original Source
- Manuel Guizar - Dec 13, 2007
- Modified from dftus, by J.R. Fienup 7/31/06

Upsampled DFT by matrix multiplies, can compute an upsampled DFT in just a small region.

This code is intended to provide the same result as if the following operations were performed:

- Embed the array "in" in an array that is usfac times larger in each dimension. ifftshift to bring the center of the image to (1,1).
- Take the FFT of the larger array
- Extract an [nor, noc] region of the result. Starting with the [roff+1 coff+1] element.

It achieves this result by computing the DFT in the output array without the need to zeropad. Much faster and memory efficient than the zero-padded FFT approach if [nor noc] are much smaller than [nr*usfac nc*usfac]

Parameters

usfac : int

Upsampling factor (default usfac = 1)

nor,noc : int,int

Number of pixels in the output upsampled DFT, in units of upsampled pixels (default = size(in))

roff, coff : int, int

Row and column offsets, allow to shift the output array to a region of interest on the DFT (default = 0)

get_ffts

image_registration.fft_tools.**get_ffts**(*nthreads=1*, *use_numpy_fft=True*)

Returns fftn,ifftn using either numpy's fft or fftw

shift2d

image_registration.fft_tools.**shift2d**(*data*, *deltax*, *deltay*, *phase=0*, *nthreads=1*, *use_numpy_fft=False*, *return_abs=False*, *return_real=True*)

2D version: obsolete - use ND version instead (though it's probably easier to parse the source of this one)

FFT-based sub-pixel image shift. Will turn NaNs into zeros

Shift Theorem:

$$FT[f(t-t_0)](x) = e^{-2\pi i x t_0} F(x)$$

Parameters

data : np.ndarray

2D image

phase : float

Phase, in radians

shiftnd

image_registration.fft_tools.**shiftnd**(*data*, *offset*, *phase=0*, *nthreads=1*, *use_numpy_fft=False*, *return_abs=False*, *return_real=True*)

FFT-based sub-pixel image shift. Will turn NaNs into zeros

Shift Theorem:

$$FT[f(t-t_0)](x) = e^{-2\pi i x t_0} F(x)$$

Parameters

data : np.ndarray

Data to shift

offset : (int,)*ndim

Offsets in each direction. Must be iterable.

phase : float

Phase, in radians

Returns

The input array shifted by offsets

Other Parameters

use_numpy_fft : bool

Force use numpy's fft over fftw? (only matters if you have fftw installed)

nthreads : bool

Number of threads to use for fft (only matters if you have fftw installed)

return_real : bool

Return the real component of the shifted array

return_abs : bool

Return the absolute value of the shifted array

smooth

image_registration.fft_tools.**smooth**(*image*, *kernelwidth=3*, *kerneltype='gaussian'*, *traps-lope=None*, *silent=True*, *psf_pad=True*, *interp_nan=False*, *nwidths='max'*, *min_nwidths=6*, *return_kernel=False*, *normal-ize_kernel=<function sum>*, *downsample=False*, *downsam-ple_factor=None*, *ignore_edge_zeros=False*, ***kwargs*)

Returns a smoothed image using a gaussian, boxcar, or tophat kernel

Parameters

kernelwidth :

width of kernel in pixels (see definitions below)

kerneltype : gaussian, boxcar, or tophat

- a gaussian, uses a gaussian with sigma = kernelwidth (in pixels) out to [nwidths]-sigma
- a boxcar is a kernelwidth x kernelwidth square
- a tophat is a flat circle with radius = kernelwidth

psf_pad : [True]

will pad the input image to be the image size + PSF. Slows things down but removes edge-wrapping effects (see convolve) This option should be set to false if the edges of your image are symmetric.

interp_nan : [False]

Will replace NaN points in an image with the smoothed average of its neighbors (you can still simply ignore NaN values by setting ignore_nan=True but leaving interp_nan=False)

silent : [True]

turn it off to get verbose statements about kernel types

return_kernel : [False]

If set to true, will return the kernel as the second return value

nwidths : ['max']

number of kernel widths wide to make the kernel. Set to 'max' to match the image shape, otherwise use any integer

min_nwidths : [6]

minimum number of gaussian widths to make the kernel (the kernel will be larger than the image if the image size is < min_widths*kernelsize)

normalize_kernel :

Should the kernel preserve the map sum (i.e. kernel.sum() = 1) or the kernel peak (i.e. kernel.max() = 1) ? Must be a *function* that can operate on a numpy array

downsample :

downsample after smoothing?

downsample_factor :

if None, default to kernelwidth

ignore_edge_zeros : bool

Ignore the zero-pad-created zeros. This will effectively decrease the kernel area on the edges but will not re-normalize the kernel. This parameter may result in 'edge-brightening' effects if you're using a normalized kernel

Notes

Note that the kernel is forced to be even sized on each axis to assure no offset when smoothing.

upsample_image

image_registration.fft_tools.**upsample_image**(*image*, *upsample_factor=1*, *output_size=None*, *nthreads=1*, *use_numpy_fft=False*, *xshift=0*, *yshift=0*)

Use dftups to upsample an image (but takes an image and returns an image with all reals)

convolve_nd Module

image_registration.fft_tools.convolve_nd Module

Functions

convolvend(array, kernel[, boundary, ...])	Convolve an ndarray with an nd-kernel.

convolvend

image_registration.fft_tools.convolve_nd.**convolvend**(*array*, *kernel*, *boundary='fill'*, *fill_value=0*, *crop=True*, *return_fft=False*, *fftshift=True*, *fft_pad=True*, *psf_pad=False*, *interpolate_nan=False*, *quiet=False*, *ignore_edge_zeros=False*, *min_wt=0.0*, *normalize_kernel=False*, *use_numpy_fft=True*, *nthreads=1*)

Convolve an ndarray with an nd-kernel. Returns a convolved image with shape = array.shape. Assumes image & kernel are centered.

Also note that the astropy.convolution convolver is a more up-to-date version of this one.

Parameters

array: 'numpy.ndarray'

Array to be convolved with *kernel*

kernel: 'numpy.ndarray'

Will be normalized if *normalize_kernel* is set. Assumed to be centered (i.e., shifts may result if your kernel is asymmetric)

boundary: str, optional

A flag indicating how to handle boundaries:

- **'fill'**
  [set values outside the array boundary to fill_value] (default)
- 'wrap' : periodic boundary

interpolate_nan: bool

attempts to re-weight assuming NAN values are meant to be ignored, not treated as zero. If this is off, all NaN values will be treated as zero.

ignore_edge_zeros: bool

Ignore the zero-pad-created zeros. This will effectively decrease the kernel area on the edges but will not re-normalize the kernel. This parameter may result in 'edge-brightening' effects if you're using a normalized kernel

min_wt: float

If ignoring NANs/zeros, force all grid points with a weight less than this value to NAN (the weight of a grid point with *no* ignored neighbors is 1.0). If `min_wt` == 0.0, then all zero-weight points will be set to zero instead of NAN (which they would be otherwise, because 1/0 = nan). See the examples below

normalize_kernel: function or boolean

if specified, function to divide kernel by to normalize it. e.g., normalize_kernel=np.sum means that kernel will be modified to be: kernel = kernel / np.sum(kernel). If True, defaults to normalize_kernel = np.sum

fft_pad: bool

Default on. Zero-pad image to the nearest 2^n

psf_pad: bool

Default off. Zero-pad image to be at least the sum of the image sizes (in order to avoid edge-wrapping when smoothing)

crop: bool

Default on. Return an image of the size of the largest input image. If the images are asymmetric in opposite directions, will return the largest image in both directions. For example, if an input image has shape [100,3] but a kernel with shape [6,6] is used, the output will be [100,6].

return_fft: bool

Return the fft(image)*fft(kernel) instead of the convolution (which is ifft(fft(image)*fft(kernel))). Useful for making PSDs.

fftshift: bool

If return_fft on, will shift & crop image to appropriate dimensions

nthreads: int

if fftw3 is installed, can specify the number of threads to allow FFTs to use. Probably only helpful for large arrays

use_numpy_fft: bool

Force the code to use the numpy FFTs instead of FFTW even if FFTW is installed

Returns

default: `array` convolved with `kernel`

if return_fft: fft(`array`) * fft(`kernel`)

- if fftshift: Determines whether the fft will be shifted before returning

if not('crop') : Returns the image, but with the fft-padded size

instead of the input size

Examples

```
>>> convolvend([1,0,3],[1,1,1])
array([ 1.,  4.,  3.])
```

```
>>> convolvend([1,np.nan,3],[1,1,1],quiet=True)
array([ 1.,  4.,  3.])
```

```
>>> convolvend([1,0,3],[0,1,0])
array([ 1.,  0.,  3.])
```

```
>>> convolvend([1,2,3],[1])
array([ 1.,  2.,  3.])
```

```
>>> convolvend([1,np.nan,3],[0,1,0], interpolate_nan=True)
array([ 1.,  0.,  3.])
```

```
>>> convolvend([1,np.nan,3],[0,1,0], interpolate_nan=True, min_wt=1e-8)
array([  1.,  nan,   3.])
```

```
>>> convolvend([1,np.nan,3],[1,1,1], interpolate_nan=True)
array([ 1.,  4.,  3.])
```

```
>>> convolvend([1,np.nan,3],[1,1,1], interpolate_nan=True, normalize_kernel=True, ignore_edge_
↪zeros=True)
array([ 1.,  2.,  3.])
```

correlate2d Module

image_registration.fft_tools.correlate2d Module

Functions

correlate2d(im1, im2[, boundary])	Cross-correlation of two images of arbitrary size.

correlate2d

image_registration.fft_tools.correlate2d.**correlate2d**(*im1*, *im2*, *boundary='wrap'*, ***kwargs*)

Cross-correlation of two images of arbitrary size. Returns an image cropped to the largest of each dimension of the input images

Parameters

return_fft - if true, return fft(im1)*fft(im2[::-1,::-1]), which is the power

spectral density

fftshift - if true, return the shifted psd so that the DC component is in

the center of the image

pad - Default on. Zero-pad image to the nearest 2^n

crop - Default on. Return an image of the size of the largest input image.

If the images are asymmetric in opposite directions, will return the largest image in both directions.

boundary: str, optional

A flag indicating how to handle boundaries:

- **'fill'**
  [set values outside the array boundary to fill_value] (default)
- 'wrap' : periodic boundary

WARNING: Normalization may be arbitrary if you use the PSD

`fast_ffts` Module

image_registration.fft_tools.fast_ffts Module

Functions

get_ffts([nthreads, use_numpy_fft])	Returns fftn,ifftn using either numpy's fft or fftw

get_ffts

image_registration.fft_tools.fast_ffts.**get_ffts**(*nthreads=1*, *use_numpy_fft=True*)
Returns fftn,ifftn using either numpy's fft or fftw

`shift` Module

image_registration.fft_tools.shift Module

Shift

Fourier-transform based shifting. scipy.fftpack.shift does about the same thing, but only in one dimension

Functions

shift2d(data, deltax, deltay[, phase, ...])	2D version: obsolete - use ND version instead
shiftnd(data, offset[, phase, nthreads, ...])	FFT-based sub-pixel image shift.

shift2d

image_registration.fft_tools.shift.**shift2d**(*data*, *deltax*, *deltay*, *phase=0*, *nthreads=1*, *use_numpy_fft=False*, *return_abs=False*, *return_real=True*)
2D version: obsolete - use ND version instead (though it's probably easier to parse the source of this one)

FFT-based sub-pixel image shift. Will turn NaNs into zeros

Shift Theorem:

$$FT[f(t-t_0)](x) = e^{-2\pi i x t_0} F(x)$$

Parameters
data : np.ndarray

2D image

phase : float

Phase, in radians

shiftnd

image_registration.fft_tools.shift.**shiftnd**(*data*, *offset*, *phase=0*, *nthreads=1*, *use_numpy_fft=False*, *return_abs=False*, *return_real=True*)

FFT-based sub-pixel image shift. Will turn NaNs into zeros

Shift Theorem:

$$FT[f(t-t_0)](x) = e^{-2\pi i x t_0} F(x)$$

Parameters
data : np.ndarray

Data to shift

offset : (int,)*ndim

Offsets in each direction. Must be iterable.

phase : float

Phase, in radians

Returns
The input array shifted by offsets

Other Parameters
use_numpy_fft : bool

Force use numpy's fft over fftw? (only matters if you have fftw installed)

nthreads : bool

Number of threads to use for fft (only matters if you have fftw installed)

return_real : bool

Return the real component of the shifted array

return_abs : bool

Return the absolute value of the shifted array

zoom Module

image_registration.fft_tools.zoom Module

Functions

zoom1d(inp[, usfac, outsize, offset, ...])	Zoom in to the center of a 1D array using Fourier upsampling
zoom_on_pixel(inp, coordinates[, usfac, ...])	Zoom in on a 1D or 2D array using Fourier upsampling

Continued on next page

Table 4.10 – continued from previous page

zoomnd(inp[, offsets, middle_convention])	Zoom in to the center of a 1D or 2D array using Fourier upsampling

zoom1d

image_registration.fft_tools.zoom.**zoom1d**(*inp*, *usfac=1*, *outsize=None*, *offset=0*, *nthreads=1*, *use_numpy_fft=False*, *return_xouts=False*, *return_real=True*)

Zoom in to the center of a 1D array using Fourier upsampling

Parameters

inp : np.ndarray

Input 1D array

usfac : int

Upsampling factor

outsize : int

Number of pixels in output array

offset : float

Offset from center *in original pixel units*

Returns

The input array upsampled by a factor `usfac` with size `outsize`.

If `return_xouts`, returns a tuple (xvals, zoomed)

Other Parameters

return_xouts : bool

Return the X indices of the output array in addition to the scaled array

return_real : bool

Return the real part of the zoomed array (if True) or the complex

zoom_on_pixel

image_registration.fft_tools.zoom.**zoom_on_pixel**(*inp*, *coordinates*, *usfac=1*, *outshape=None*, *nthreads=1*, *use_numpy_fft=False*, *return_real=True*, *return_xouts=False*)

Zoom in on a 1D or 2D array using Fourier upsampling (in principle, should work on N-dimensions, but does not at present!)

Parameters

inp : np.ndarray

Input 1D array

coordinates : tuple of floats

Pixel to zoom in on

usfac : int

Upsampling factor

outshape : int

Number of pixels in output array

Returns

The input array upsampled by a factor `usfac` with size `outshape`.

If `return_xouts`, returns a tuple (xvals, zoomed)

Other Parameters

return_xouts : bool

Return the X indices of the output array in addition to the scaled array

return_real : bool

Return the real part of the zoomed array (if True) or the complex

zoomnd

`image_registration.fft_tools.zoom.`**zoomnd**(*inp*, *offsets=()*, *middle_convention=<type 'float'>*, ***kwargs*)

Zoom in to the center of a 1D or 2D array using Fourier upsampling (in principle, should work on N-dimensions, but does not at present!)

Parameters

inp : np.ndarray

Input 1D array

offsets : tuple of floats

Offset from center *in original pixel units*"

middle_convention : function

What convention to use for the "Middle" of the array. Should be either float (i.e., can be half-pixel), floor, or ceil. I don't think round makes a ton of sense... should just be ceil.

usfac : int

Upsampling factor (passed to `zoom_on_pixel()`)

outshape : int

Number of pixels in output array (passed to `zoom_on_pixel()`)

Returns

The input array upsampled by a factor `usfac` with size `outshape`.

If `return_xouts`, returns a tuple (xvals, zoomed)

Other Parameters

return_xouts : bool

Return the X indices of the output array in addition to the scaled array (passed to `zoom_on_pixel()`)

return_real : bool

Return the real part of the zoomed array (if True) or the complex (passed to `zoom_on_pixel()`)

Continued on next page

Table 4.11 – continued from previous page

scale Module

image_registration.fft_tools.scale Module

Functions

fourier_interp1d(data, out_x[, data_x, ...])	Use the fourier scaling theorem to interpolate (or extrapolate, without raising any exceptions) data.
fourier_interp2d(data, outinds[, nthreads, ...])	Use the fourier scaling theorem to interpolate (or extrapolate, without raising any exceptions) data.
fourier_interpnd(data, outinds[, nthreads, ...])	Use the fourier scaling theorem to interpolate (or extrapolate, without raising any exceptions) data.

fourier_interp1d

image_registration.fft_tools.scale.**fourier_interp1d**(*data*, *out_x*, *data_x=None*, *nthreads=1*, *use_numpy_fft=False*, *return_real=True*)

Use the fourier scaling theorem to interpolate (or extrapolate, without raising any exceptions) data.

Parameters

data : ndarray

The Y-values of the array to interpolate

out_x : ndarray

The X-values along which the data should be interpolated

data_x : ndarray | None

The X-values corresponding to the data values. If an ndarray, must have the same shape as data. If not specified, will be set to np.arange(data.size)

nthreads : int

Number of threads for parallelized FFTs (if available)

use_numpy_fft : bool

Use the numpy version of the FFT before any others? (Default is to use fftw3)

Returns

The real component of the interpolated 1D array, or the full complex array

if return_real is False

Raises

ValueError if output indices are the wrong shape or the data X array is the

wrong shape

fourier_interp2d

image_registration.fft_tools.scale.**fourier_interp2d**(*data*, *outinds*, *nthreads=1*, *use_numpy_fft=False*, *return_real=True*)

Use the fourier scaling theorem to interpolate (or extrapolate, without raising any exceptions) data.

Parameters

data : ndarray

The data values of the array to interpolate

outinds : ndarray

The coordinate axis values along which the data should be interpolated CAN BE: `ndim x [n,m,...]` like np.indices OR (less memory intensive, more processor intensive) `([n],[m],...)`

fourier_interpnd

`image_registration.fft_tools.scale.`**`fourier_interpnd`**(*data*, *outinds*, *nthreads=1*, *use_numpy_fft=False*, *return_real=True*)

Use the fourier scaling theorem to interpolate (or extrapolate, without raising any exceptions) data. * DOES NOT WORK FOR ANY BUT 2 DIMENSIONS *

Parameters

data : ndarray

The data values of the array to interpolate

outinds : ndarray

The coordinate axis values along which the data should be interpolated CAN BE `ndim x [n,m,...]` like np.indices OR (less memory intensive, more processor intensive) `([n],[m],...)`

`upsample` Module

Fourier upsampling (or interpolation, scaling, zooming) is achieved via DFTs using a dot product rather than the usual fft, as there is (probably?) no way to perform FFTs with a different kernel.

This notebook demonstrates 1-d Fourier upsampling.

image_registration.fft_tools.upsample Module

Functions

`dftups`(inp[, nor, noc, usfac, roff, coff])	Translated from matlab:
`dftups1d`(inp[, usfac, outsize, offset, ...])	
`odddftups`(inp[, nor, noc, usfac, roff, coff])	
`upsample_image`(image[, upsample_factor, ...])	Use dftups to upsample an image (but takes an image and returns an image with all reals)

dftups

`image_registration.fft_tools.upsample.`**`dftups`**(*inp*, *nor=None*, *noc=None*, *usfac=1*, *roff=0*, *coff=0*)

Translated from matlab:

- Original Source
- Manuel Guizar - Dec 13, 2007
- Modified from dftus, by J.R. Fienup 7/31/06

Upsampled DFT by matrix multiplies, can compute an upsampled DFT in just a small region.

This code is intended to provide the same result as if the following operations were performed:

- Embed the array "in" in an array that is usfac times larger in each dimension. ifftshift to bring the center of the image to (1,1).
- Take the FFT of the larger array
- Extract an [nor, noc] region of the result. Starting with the [roff+1 coff+1] element.

It achieves this result by computing the DFT in the output array without the need to zeropad. Much faster and memory efficient than the zero-padded FFT approach if [nor noc] are much smaller than [nr*usfac nc*usfac]

Parameters

usfac : int

Upsampling factor (default usfac = 1)

nor,noc : int,int

Number of pixels in the output upsampled DFT, in units of upsampled pixels (default = size(in))

roff, coff : int, int

Row and column offsets, allow to shift the output array to a region of interest on the DFT (default = 0)

dftups1d

image_registration.fft_tools.upsample.**dftups1d**(*inp*, *usfac=1*, *outsize=None*, *offset=0*, *return_xouts=False*)

odddftups

image_registration.fft_tools.upsample.**odddftups**(*inp*, *nor=None*, *noc=None*, *usfac=1*, *roff=0*, *coff=0*)

upsample_image

image_registration.fft_tools.upsample.**upsample_image**(*image*, *upsample_factor=1*, *output_size=None*, *nthreads=1*, *use_numpy_fft=False*, *xshift=0*, *yshift=0*)

Use dftups to upsample an image (but takes an image and returns an image with all reals)

Continued on next page

Table 4.13 – continued from previous page

4.9.2 FITS_tools Package

FITS_tools Package

image_registration.FITS_tools Package

Functions

fits_overlap(file1, file2, **kwargs)	Create a header containing the exact overlap region between two .fits files
get_cd(wcs[, n])	Get the value of the change in world coordinate per pixel across a linear axis.
header_overlap(hdr1, hdr2[, max_separation, ...])	Create a header containing the exact overlap region between two .fits files
load_data(data)	Attempt to load an image specified as an HDU, a string pointing to a FITS
load_header(header)	Attempt to load a header specified as a header, a string pointing to a FITS
match_fits(fitsfile1, fitsfile2[, header, ...])	Project one FITS file into another's coordinates
project_to_header(fitsfile, header, **kwargs)	Reproject an image to a header.
register_fits(fitsfile1, fitsfile2[, ...])	Determine the shift between two FITS images using the cross-correlation technique.

fits_overlap

image_registration.FITS_tools.**fits_overlap**(*file1*, *file2*, ***kwargs*)

Create a header containing the exact overlap region between two .fits files

Does NOT check to make sure the FITS files are in the same coordinate system!

Parameters

file1,file2 : str,str

files from which to extract header strings

get_cd

image_registration.FITS_tools.**get_cd**(*wcs*, *n=1*)

Get the value of the change in world coordinate per pixel across a linear axis. Defaults to wcs.wcs.cd if present. Does not support rotated headers (e.g., with nonzero CDm_n where m!=n)

header_overlap

image_registration.FITS_tools.**header_overlap**(*hdr1*, *hdr2*, *max_separation=180*, *overlap='union'*)

Create a header containing the exact overlap region between two .fits files

Does NOT check to make sure the FITS files are in the same coordinate system!

Parameters

hdr1,hdr2 : pyfits.Header

Two pyfits headers to compare

max_separation : int

Maximum number of degrees between two headers to consider before flipping signs on one of them (this to deal with the longitude=0 region)

overlap: 'union' or 'intersection'

Which merger to do

load_data

image_registration.FITS_tools.**load_data**(*data*)

Attempt to load an image specified as an HDU, a string pointing to a FITS file, an HDUlist, or an actual data array

load_header

image_registration.FITS_tools.**load_header**(*header*)

Attempt to load a header specified as a header, a string pointing to a FITS file, or a string pointing to a Header text file, or a string that contains the actual header, or an HDU

match_fits

image_registration.FITS_tools.**match_fits**(*fitsfile1*, *fitsfile2*, *header=None*, *sigma_cut=False*, *return_header=False*, ***kwargs*)

Project one FITS file into another's coordinates If sigma_cut is used, will try to find only regions that are significant in both images using the standard deviation

Parameters

fitsfile1: str

Reference fits file name

fitsfile2: str

Offset fits file name

header: pyfits.Header

Optional - can pass a header to projet both images to

sigma_cut: bool or int

Perform a sigma-cut on the returned images at this level

Returns

image1,image2,[header] : Two images projected into the same space, and

optionally the header used to project them

project_to_header

image_registration.FITS_tools.**project_to_header**(*fitsfile*, *header*, ***kwargs*)

Reproject an image to a header. Simple wrapper of reproject.reproject_interp

Parameters

fitsfile : string

a FITS file name

header : pyfits.Header

A pyfits Header instance with valid WCS to project to

quiet : bool

Silence Montage's output

Returns

np.ndarray image projected to header's coordinates

register_fits

image_registration.FITS_tools.**register_fits**(*fitsfile1*, *fitsfile2*, *errfile=None*, *return_error=True*, *register_method=<function chi2_shift_iterzoom>*, *return_cropped_images=False*, *return_shifted_image=False*, *return_header=False*, ***kwargs*)

Determine the shift between two FITS images using the cross-correlation technique. Requires montage or hcongrid.

kwargs are passed to `register_method()`

Parameters

fitsfile1: str

Reference fits file name

fitsfile2: str

Offset fits file name

errfile : str [optional]

An error image, intended to correspond to fitsfile2

register_method : function

Can be any of the shift functions in `image_registration`. Defaults to `chi2_shift_iterzoom()`

return_errors: bool

Return the errors on each parameter in addition to the fitted offset

return_cropped_images: bool

Returns the images used for the analysis in addition to the measured offsets

return_shifted_images: bool

Return image 2 shifted into image 1's space

return_header : bool

Return the header the images have been projected to

quiet: bool

Silence messages?

sigma_cut: bool or int

Perform a sigma-cut before cross-correlating the images to minimize noise correlation?

Returns

xoff,yoff : (float,float)

pixel offsets

xoff_wcs,yoff_wcs : (float,float)

world coordinate offsets

exoff,eyoff : (float,float) (only if `return_errors` is True)

Standard error on the fitted pixel offsets

exoff_wcs,eyoff_wcs : (float,float) (only if `return_errors` is True)

Standard error on the fitted world coordinate offsets

proj_image1, proj_image2 : (ndarray,ndarray) (only if `return_cropped_images` is True)

The images projected into the same coordinates

shifted_image2 : ndarray (if `return_shifted_image` is True)

The second image projected *and shifted* to match image 1.

header : `pyfits.Header` (only if `return_header` is True)

The header the images have been projected to

`match_images` Module

image_registration.FITS_tools.match_images Module

Functions

`project_to_header`(fitsfile, header, **kwargs)	Reproject an image to a header.
`match_fits`(fitsfile1, fitsfile2[, header, ...])	Project one FITS file into another's coordinates
`register_fits`(fitsfile1, fitsfile2[, ...])	Determine the shift between two FITS images using the cross-correlation technique.

project_to_header

image_registration.FITS_tools.match_images.**project_to_header**(*fitsfile*, *header*, ***kwargs*)

Reproject an image to a header. Simple wrapper of reproject.reproject_interp

Parameters

fitsfile : string

a FITS file name

header : pyfits.Header

A pyfits Header instance with valid WCS to project to

quiet : bool

Silence Montage's output

Returns

np.ndarray image projected to header's coordinates

match_fits

image_registration.FITS_tools.match_images.**match_fits**(*fitsfile1*, *fitsfile2*, *header=None*, *sigma_cut=False*, *return_header=False*, ***kwargs*)

Project one FITS file into another's coordinates If sigma_cut is used, will try to find only regions that are significant in both images using the standard deviation

Parameters

fitsfile1: str

Reference fits file name

fitsfile2: str

Offset fits file name

header: pyfits.Header

Optional - can pass a header to projet both images to

sigma_cut: bool or int

Perform a sigma-cut on the returned images at this level

Returns

image1,image2,[header] : Two images projected into the same space, and

optionally the header used to project them

register_fits

image_registration.FITS_tools.match_images.**register_fits**(*fitsfile1*, *fitsfile2*, *errfile=None*, *return_error=True*, *register_method=<function chi2_shift_iterzoom>*, *return_cropped_images=False*, *return_shifted_image=False*, *return_header=False*, ***kwargs*)

Determine the shift between two FITS images using the cross-correlation technique. Requires montage or hcongrid.

kwargs are passed to `register_method()`

Parameters

fitsfile1: str

Reference fits file name

fitsfile2: str

Offset fits file name

errfile : str [optional]

An error image, intended to correspond to fitsfile2

register_method : function

Can be any of the shift functions in `image_registration`. Defaults to `chi2_shift_iterzoom()`

return_errors: bool

Return the errors on each parameter in addition to the fitted offset

return_cropped_images: bool

Returns the images used for the analysis in addition to the measured offsets

return_shifted_images: bool

Return image 2 shifted into image 1's space

return_header : bool

Return the header the images have been projected to

quiet: bool

Silence messages?

sigma_cut: bool or int

Perform a sigma-cut before cross-correlating the images to minimize noise correlation?

Returns

xoff,yoff : (float,float)

pixel offsets

xoff_wcs,yoff_wcs : (float,float)

world coordinate offsets

exoff,eyoff : (float,float) (only if `return_errors` is True)

Standard error on the fitted pixel offsets

exoff_wcs,eyoff_wcs : (float,float) (only if `return_errors` is True)

Standard error on the fitted world coordinate offsets

proj_image1, proj_image2 : (ndarray,ndarray) (only if `return_cropped_images` is True)

The images projected into the same coordinates

shifted_image2 : ndarray (if `return_shifted_image` is True)

The second image projected *and shifted* to match image 1.

header : `pyfits.Header` (only if `return_header` is True)

The header the images have been projected to

`fits_overlap` Module

image_registration.FITS_tools.fits_overlap Module

Functions

`fits_overlap`(file1, file2, **kwargs)	Create a header containing the exact overlap region between two .fits files
`header_overlap`(hdr1, hdr2[, max_separation, ...])	Create a header containing the exact overlap region between two .fits files

fits_overlap

image_registration.FITS_tools.fits_overlap.**fits_overlap**(*file1*, *file2*, ***kwargs*)

Create a header containing the exact overlap region between two .fits files

Does NOT check to make sure the FITS files are in the same coordinate system!

Parameters

file1,file2 : str,str

files from which to extract header strings

header_overlap

image_registration.FITS_tools.fits_overlap.**header_overlap**(*hdr1*, *hdr2*, *max_separation=180*, *overlap='union'*)

Create a header containing the exact overlap region between two .fits files

Does NOT check to make sure the FITS files are in the same coordinate system!

Parameters

hdr1,hdr2 : pyfits.Header

Two pyfits headers to compare

max_separation : int

Maximum number of degrees between two headers to consider before flipping signs on one of them (this to deal with the longitude=0 region)

overlap: 'union' or 'intersection'

Which merger to do

hcongrid Module

image_registration.FITS_tools.hcongrid Module

i

C

D

F

G

H

I

L

M

O

P

R

S

T

U

Z