Subject: Re: Scale the psf on images.
Posted by on Thu, 15 Jan 2015 10:19:59 GMT
View Forum Message <> Reply to Message

```
Den torsdag 15 januari 2015 kl. 09:04:18 UTC+1 skrev anes.tz...@gmail.com:
>  On Thursday, January 15, 2015 at 3:58:18 PM UTC+8, Mats Löfdahl wrote:
>>  Den torsdag 15 januari 2015 kl. 08:13:19 UTC+1 skrev anes.tz...@gmail.com:
>>>  Well perhaps i will get the award for the dumbest person in the world for this.
>>>
>>>  On the following link there is an fft example
>>>  http://www.exelisvis.com/docs/fftreducebackgroundnoise.html
>>>
>>>  As you said Mats they do something to reduce the noise. The problem is  by using this method, again the final image doesnt make sense. There is no structure actually and the pixel values vary from -1e7 to 1e7. This is the code
>>>
>>>  fits_read, 'Image1.fits',image1,header1
>>>  fits_read, 'Image1.fits.psf.1.fits',image2,header2
>>>  fits_read, 'Image2.fits',image3,header3
>>>  fits_read, 'Image2.fits.psf.1.fits',image4,header4
>>>
>>>
>>>  ;Compute the difference kernel
>>>  p1=fft(image2)
>>>  p2=fft(image4)
>>>
>>>  ; Compute the power spectrum of the transform and apply a log scale.
>>>
>>>  powerSpectrum = ABS(p1)^2
>>>  scaledPowerSpect = ALOG10(powerSpectrum)
>>>
>>>  powerSpectrum2 = ABS(p2)^2
>>>  scaledPowerSpect2 = ALOG10(powerSpectrum2)
>>>
>>>
>>>  ; Scale the power spectrum to make its maximum value equal to 0.
>>>  scaledPS = scaledPowerSpect - MAX(scaledPowerSpect)
>>>  scaledPS2 = scaledPowerSpect2 - MAX(scaledPowerSpect2)
>>>
>>>  surface,powerSpectrum2
>>>
>>>  mask = REAL_PART(scaledPowerSpect) GT -5
>>>  psf1_trans = p1*mask
>>>  mask2 = REAL_PART(scaledPowerSpect2) GT -5
>>>  psf2_trans = p2*mask2
>>>  kernel = REAL_PART(FFT(psf1_trans/psf2_trans, /INVERSE, /CENTER))
>>>
>>>
```

```
>>>  image3_prime = convol(image3, kernel)
>>>  diff = image1 - image3_prime
>>>
>>>
>>>  WRITEFITS,'im_conv.fits',diff,header3
>>>  end
>>
```

>> You are aware that you are trying to construct noise filters based on the power spectra of the PSFs?
>
> To be honest I am totally lost with it. This is the first time I am doing something like this, so actually I have no clue on what each command does. A bit more light is appreciated. Thanks a lot for your time.

I think you need to think of this as a mathematics problem and not only as a coding problem. If you don't understand what the commands do and what to expect from the data, there is no way you are going to be able to adapt code snippets you find to your own problem.

For example, the code you found at exelisvis only does filtering but there are no PSFs or kernels involved so you can't apply it blindly to your problem.

A maths thing you missed: The misnamed variables psf1_trans and psf2_trans (they are not psfs, they are the Fourier transforms of psfs, i.e., otfs) have zeroes where mask1 and mask2 have zeroes, so when you do psf1_trans/psf2_trans you divide by zero where mask2 has zeroes.

So this is not what you want to do, you want a single low-pass filter (or mask, as you name it) that you use to get rid of data in frequencies where you would otherwise amplify noise rather than signal in (the Fourier transform of) the image you are modifying, image2 in this case (going by the file name, you variable is called image3). So base your filter on where image2 is noise dominated.

Understanding what quantities you are dealing with and naming the variables properly and consistently might help some. You could start your program like this:

```
; Read image 1 and its psf
fits_read, 'Image1.fits', image1, image_header1
fits_read, 'Image1.fits.psf.1.fits',psf1, psf_header1

; Read image 2 and its psf
fits_read, 'Image2.fits', image2, image_header2
fits_read, 'Image2.fits.psf.1.fits',psf2, psf_header2

;Compute the optical transfer functions
otf1 = fft(psf1)
otf2 = fft(psf2)
```

Just so you don't confuse yourself with what is what later in the code.

And then do read up on image processing, what to expect the Fourier transform of an image to look like (where to find signal and where to find noise), and how to construct a proper filter. Make sure you understand Fourier transforms enough that your convolution operation, including the low-pass filtering, can be expressed as multiplication and division in the Fourier domain. Saves you the call to convol() and makes it easier to see where you are dividing by zero or almost zero.

Or try the dirty trick I suggested in my first reply. The good thing about it is that since you don't do any convolutions that might amplify noise, you don't have to bother with noise filters. The assumptions behind it is that the otfs are mainly real-valued and that the images are similar enough that the differences you are looking for do not change the power spectrum significantly.

And you need the two images to be already registered and have the same spatial image scale, of course. But that goes for what you are trying to do now as well.
