Package ‘ProFound’

November 28, 2018

Type Package

Title Photometry Tools

Version 1.3.4

Date 2018-11-12

Author Aaron Robotham

Maintainer Aaron Robotham <aaron.robotham@uwa.edu.au>

Description Core package containing all the tools for simple and advanced source extraction. This is used to create inputs for 'ProFit', or for source detection, extraction and photometry in its own right.

License LGPL-3

Depends R (>= 3.1), FITSio, magicaxis (>= 2.0.3)

Imports RColorBrewer, data.table, celestial (>= 1.4.1), foreach, doParallel, Rcpp

Suggests ProFit, knitr, rmarkdown, EBImage, akima, imager, LaplacesDemon

VignetteBuilder knitr

SystemRequirements C++11

LinkingTo Rcpp

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-11-28 13:40:03 UTC

R topics documented:

`ProFound-package` *Photometry Tools*

Description

Core package containing all the tools for simple and advanced source extraction. This is used to create inputs for 'ProFit', or for source detection, extraction and photometry in its own right.

Details

Package:	ProFound
Type:	Package
Version:	1.3.4
Date:	2018-11-12
License:	LGPL-3
Depends:	R (>= 3.0), FITSio, magicaxis (>= 2.0.3)
Imports:	RColorBrewer, data.table, celestial (>= 1.4.1)
Suggests:	ProFit, knitr, rmarkdown, EBImage, akima, imager, LaplacesDemon

Author(s)

Aaron Robotham

Maintainer: Aaron Robotham <aaron.robotham@uwa.edu.au>

References

Robotham A.S.G., et al., 2018, MNRAS, 476, 3137

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits',
package="ProFound"))

profound=profoundProFound(image, skycut=1.5, magzero=30, verbose=TRUE, plot=TRUE)

## End(Not run)
```

FPtest *False Positive Reference Data*

Description

This data consists of 1,000 runs of a random 1000 x 1000 noise matrix through `profoundProFound`. The catalogue is a concatenation of all the segstats outputs for all of these run.

Usage

```
data("FPtest")
```

Format

A data frame with 7012 observations on the following 56 variables. See `profoundProFound` for a detailed discussion on each of these parameters.

`segID` a numeric vector

`uniqueID` a numeric vector

`xcen` a numeric vector

`ycen` a numeric vector

`xmax` a numeric vector

`ymax` a numeric vector

`RAcen` a logical vector

`Deccen` a logical vector

`RAmax` a logical vector

`Decmax` a logical vector

`sep` a numeric vector

`flux` a numeric vector

`mag` a numeric vector

`cenfrac` a numeric vector

`N50` a numeric vector

`N90` a numeric vector

`N100` a numeric vector

`R50` a numeric vector

`R90` a numeric vector

`R100` a numeric vector

`SB_N50` a numeric vector

`SB_N90` a numeric vector

`SB_N100` a numeric vector

`xsd` a numeric vector

`ysd` a numeric vector

`covxy` a numeric vector

`corxy` a numeric vector

`con` a numeric vector

`asymm` a logical vector

`flux_reflect` a logical vector

`mag_reflect` a logical vector

`semimaj` a numeric vector

`semimin` a numeric vector

`axrat` a numeric vector

`ang` a numeric vector

`signif` a numeric vector

`FPlim` a numeric vector

`flux_err` a numeric vector

`mag_err` a numeric vector

`flux_err_sky` a numeric vector

`flux_err_skyRMS` a numeric vector

`flux_err_shot` a numeric vector

`sky_mean` a numeric vector

`sky_sum` a numeric vector

`skyRMS_mean` a numeric vector

`Nedge` a logical vector

`Nsky` a logical vector

`Nobject` a logical vector

`Nborder` a logical vector

`Nmask` a logical vector

`edge_frac` a logical vector

`edge_excess` a logical vector

`flag_border` a logical vector

`iter` a numeric vector

`origfrac` a numeric vector

`flag_keep` a logical vector

Details

Specifically we ran with defaults the following command 1,000 times in a loop:

profoundProFound(matrix(rnorm(1e6),1e3))

The output is then a reference of the false positive rate, since we have not injected any sources into the images. The fact we find 7,012 false detections mean we expect 7 false positives per 1e6 pixels (the size in pixels of the input matrix). To compare against any target data we need to adjust the magnitudes by the sky RMS magnitude level, i.e. add on profoundFlux2Mag(skyRMS, 0) (if the zero point is 0 for our target data). See Examples for a comparison to our included VIKING data.

Source

FPtest=

for(i in 1:1000)FPtest=rbind(FPtest,profoundProFound(matrix(rnorm(1e6),1e3))$segstats)

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits', package="ProFound"))
profound=profoundProFound(image, magzero=30, rotstats=TRUE)
skyRMS=median(profound$skyRMS)
magoff=profoundFlux2Mag(skyRMS, 30)
totpix=prod(profound$dim)

#We can easily compute the expected number of false positives on an image this size:
data("FPtest")
dim(FPtest)[1]*totpix/1e6/1e3

#And plot the detections and expected false positive distributions:
maghist(profound$segstats$mag, seq(-11,-1,by=0.2)+magoff)
maghist(FPtest$mag+magoff, seq(-6,-1,by=0.2)+magoff, scale=totpix/1e6/1e3, add=TRUE,
border='red')

## End(Not run)
```

`plot.profound` *ProFound Diagnostic Grid*

Description

A useful visual grid of ProFound diagnostics. This is useful for checking if something very odd has occurred when running the code.

Usage

```
## S3 method for class 'profound'
plot(x, logR50 = TRUE, dmag=0.5, hist='sky', ...)
```

Arguments

`x`	Argument for the class dependent `plot.profound` function. An object of class profound as output by the `profoundProFound` function. This is the only structure that needs to be provided when executing plot(profound) class dependent plotting, which will use the plot.profound function.
`logR50`	Logical; specifies whether the bottom-centre panel uses a logarithmic y-axis for R50 (default is TRUE).
`dmag`	Numeric scaler; the magnitude binning scale to use (default 0.5 to reflect the axis binning). The magnitude histograms always use 0.5 magnitude bins, but this controls the y-axis scaling to give the correct normalisation as if the specified binning was used. I.e. the raw counts are scaled by an additional factor of 2 if 'dmag'=1 is specified.
`hist`	Character scalar; specifies the plot type for the bottom-left plot. Options are 'sky' (which is a sky pixel (image-sky)/skyRMS PDF using the objects_redo mask) or 'iters' (histogram of required iterations). Old default was 'iters', but now 'sky', since this is more useful in general.
`...`	Nothing to see here.

Details

Run for the side effect of generating a grid of useful diagnostic plots.

Value

Run for the side effect of generating a grid of useful diagnostic plots:

`Top-left`	Sky subracted image '`x$image`'-'`x$sky`', where blue is negative, yellow is 0, and red is positive
`Top-centre`	Output segmentation map '`x$segim`'
`Top-right`	Sky subracted and normalised image ('`x$image`'-'`x$sky`')/'`x$skyRMS`', with segment dilation extent shown in colour

`Middle-left`	Magnitude ('`x$segstats$mag`') counts histogram (max in red), scaled to counts per square degree if '`x$header`' is present
`Middle-centre`	Output '`x$sky`', where blue is negative, yellow is 0, and red is positive
`Middle-right`	Output '`x$skyRMS`', where dark is lower values and white larger values
`Bottom-left`	Sky pixel ('`x$image`'-'`x$sky`')/'`x$skyRMS`' PDF, or dilation iteration ('`x$segstats$iter`') histogram (depends of '`hist`')
`Bottom-centre`	Output mag ('`x$segstats$mag`') versus R50 ('`x$segstats$R50`')
`Bottom-right`	Output mag ('`x$segstats$mag`') versus axrat ('`x$segstats$axrat`')

Author(s)

Aaron Robotham

See Also

`profoundProFound`

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits', package="ProFound"))

profound=profoundProFound(image, skycut=1.5, magzero=30, verbose=TRUE, plot=TRUE)

plot(profound)

## End(Not run)
```

ProFound *ProFound Source Detection*

Description

This is the highest level source detection function provided in `ProFit`, calculating both the initial segmentation map and reasonable estimates for the total flux apertures for each source in an automatic manner.

Usage

```
profoundProFound(image = NULL, segim = NULL, objects = NULL, mask = NULL,  skycut=1,
pixcut=3, tolerance = 4, ext = 2, reltol = 0, cliptol = Inf, sigma = 1, smooth = TRUE,
SBlim, size = 5, shape = "disc", iters = 6, threshold = 1.05, converge = 'flux',
magzero = 0, gain = NULL, pixscale = 1, sky = NULL, skyRMS = NULL, redosegim = FALSE,
redosky = TRUE, redoskysize = 21, box = c(100,100), grid = box, type = "bicubic",
skytype = "median", skyRMStype = "quanlo", roughpedestal = FALSE, sigmasel = 1,
skypixmin = prod(box)/2, boxadd = box/2, boxiters = 0, deblend = FALSE, df = 3,
radtrunc = 2, iterative = FALSE, doclip = TRUE, shiftloc = FALSE, paddim = TRUE, header,
```

```
verbose = FALSE, plot = FALSE, stats = TRUE, rotstats = FALSE, boundstats = FALSE,
nearstats = boundstats, groupstats = boundstats, group = NULL, groupby = 'segim_orig',
offset = 1, haralickstats = FALSE, sortcol = "segID", decreasing = FALSE,
lowmemory = FALSE, keepim = TRUE, R50clean=0, watershed = 'EBImage', ...)
```

Arguments

`image`	Numeric matrix; required, the image we want to analyse. If 'image' is a list as created by readFITS, read.fits of magcutoutWCS then the image part of these lists is passed to 'image' and the correct header part is passed to 'header'. Note, image NAs are treated as masked pixels.
`segim`	Integer matrix; a specified segmentation map of the image. This matrix *must* be the same dimensions as 'image' if supplied. If this option is used then profoundProFound will not compute its initial segmentation map using profoundMakeSegim, which is then dilated. Instead it will use the one passed through 'segim'.
`objects`	Boolean matrix; optional, object mask where 1 is object and 0 is sky. If provided, this matrix *must* be the same dimensions as 'image'.
`mask`	Boolean matrix or integer scalar; optional, parts of the image to mask out (i.e. ignore). If a matrix is provided, this matrix *must* be the same dimensions as 'image' where 1 means mask out and 0 means use for analysis. if a scalar is provided it indicates the exact 'image' values that should be treated as masked (e.g. by setting masked pixels to 0 or -999). The latter achieves the same effect as setting masked 'image' pixels to NA, but allows for the fact not all programs can produce R legal NA values.
`skycut`	Numeric scalar; the lowest threshold to make on the 'image' in units of the skyRMS. Passed to profoundMakeSegim.
`pixcut`	Integer scalar; the number of pixels required to identify an object. Passed to profoundMakeSegim.
`tolerance`	Numeric scalar; the minimum height of the object in the units of skyRMS between its highest point (seed) and the point where it contacts another object (checked for every contact pixel). If the height is smaller than the tolerance, the object will be combined with one of its neighbours, which is the highest. The range 1-5 offers decent results usually. Passed to profoundMakeSegim.
`ext`	Numeric scalar; radius of the neighbourhood in pixels for the detection of neighbouring objects. Higher value smooths out small objects. Passed to profoundMakeSegim.
`reltol`	Numeric scalar; only relevant for 'watershed'='ProFound'. A modifier to the 'abstol', modifying it by the ratio of the segment peak flux divided by the saddle point flux to the power 'reltol'. The default means the 'reltol' has no effect since this modifier becomes 1. A larger value of 'reltol' means segments are more aggressively merged together.
`cliptol`	Numeric scalar; only relevant for 'watershed'='ProFound'. If ('image'-'sky')/optionskyRMS is above this level where segments touch then they are always merged, regardless of other criteria. When thinking in terms of sky RMS, values between 20-100 are probably appropriate for merging very bright parts of stars back together in optical data.

sigma
: Numeric scalar; standard deviation of the blur used when 'smooth'=TRUE. Passed to profoundMakeSegim.

smooth
: Logical; should smoothing be done on the target 'image'? Passed to profoundMakeSegim. If present, this will use the imblur function from the imager package. Otherwise it will use the gblur function from the EBImage package with a warning. These functions are very similar in output, but not strictly identical.

SBlim
: Numeric scalar; the mag/asec^2 surface brightness threshold to apply. This is always used in conjunction with 'skycut', so set 'skycut' to be very large (e.g. Inf) if you want a pure surface brightness threshold for the segmentation. 'magzero' and 'pixscale' must also be present for this to be used. Passed to profoundMakeSegim.

size
: Integer scalar; the size (e.g. width/diameter) of the dilation kernel in pixels. Should be an odd number else will be rounded up to the nearest odd number. See makeBrush. Passed to profoundMakeSegimDilate.

shape
: Character scalar; the shape of the dilation kernel. See makeBrush. Passed to profoundMakeSegimDilate.

iters
: Integer scalar; the maximum number of curve of growth dilations that should be made. This needs to be large enough to capture all the flux for sources of interest, but increasing this will increase the computation time for profoundProFound. If this is set to 0 then the undilated 'segim' image, whether provided or computed internally via profoundMakeSegim, will be used instead.

threshold
: Numeric scalar; After the curve of growth dilations, 'threshold' is the relative change of the converging property (see 'converge') that flags convergence. If consecutive iterations have a relative difference within this ratio then the dilation is stopped, and this iteration is used to define the segmentation of the object. The effect of this is that different objects will be dilated for a different number of iterations. Usually fainter sources require more.

converge
: Character scalar; the segmentation property to compare for relative convergence. The options are in principle any column that is output by profoundSegimStats, but in practice it should be something that increases slowly with dilation and tends to converge when the total flux is being captured. Good options are therefore 'flux' (default), 'R50' and 'R90'.

magzero
: Numeric scalar; the magnitude zero point. What this implies depends on the magnitude system being used (e.g. AB or Vega). If provided along with 'pixscale' then the flux and surface brightness outputs will represent magnitudes and mag/asec^2.

gain
: Numeric scalar; the gain (in photo-electrons per ADU). This is only used to compute object shot-noise component of the flux error (else this is set to 0).

pixscale
: Numeric scalar; the pixel scale, where pixscale=asec/pix (e.g. 0.4 for SDSS). If set to 1 (default), then the output is in terms of pixels, otherwise it is in arcseconds. If provided along with 'magzero' then the flux and surface brightness outputs will represent magnitudes and mag/asec^2.

sky
: User provided estimate of the absolute sky level. If this is not provided then it will be computed internally using profoundMakeSkyGrid. Can be a scalar or a matrix matching the dimensions of 'image' (allows values to vary per pixel). This will be subtracted off the 'image' internally, so only provide this if the sky does need to be subtracted!

skyRMS	User provided estimate of the RMS of the sky. If this is not provided then it will be computed internally using profoundMakeSkyGrid. Can be a scalar or a matrix matching the dimensions of 'image' (allows values to vary per pixel).
redosegim	Logical; should the segmentation map be modified based using the interim "better sky"? This means pixels falling below the new 'skycut' would be excluded from the final segmentation map. This is usually only required if the sky subtraction was radically poor and complex in the first place. Will be forced to FALSE if the user supplies a segmentation map. If the user wants to flag object pixels they should pass it to 'objects'.
redosky	Logical; should the sky and sky RMS grids be re-computed using the final segmentation map? This uses profoundMakeSkyGrid to compute the sky and sky RMS grids. If 'redosky'=TRUE then the output will include the aggressively masked 'objects_redo' image, if 'redosky'=FALSE then 'objects_redo' will be NA.
redoskysize	Integer scalar; the size (e.g. width/diameter) of the dilation kernel in pixels to apply to the 'object' mask before performing the initial and final aggressively masked sky estimates (the latter is only relevant if 'redosky'=TRUE). Should be an odd number else will be rounded up to the nearest odd number. See makeBrush. Dilation is done by profoundMakeSegimDilate. If 'redosky'=TRUE, the final dilated 'objects' mask is returned as 'objects_redo'. As a rule of thumb you probably want ~50% of your image pixels to be masked as objects, much more than this and you might not be able to sample enough sky pixels, much more less and the sky estimates might be biased by object flux in the wings.
box	Integer vector; the dimensions of the box car filter to estimate the sky with. For convenience, if length 1 then both dimensions of 'box' used internally are assumed to equal the specified 'box'. I.e. 200 would be interpreted as c(200,200). Dependent default arguments ('grid', 'boxadd' and 'skypixmin') are updated sensibly.
grid	Integer vector; the resolution of the background grid to estimate the sky with. By default this is set to be the same as the 'box'.
type	Character scalar; either "bilinear" for bilinear interpolation or "bicubic" for bicubic interpolation (default). The former creates sharper edges.
skytype	Character scalar; the type of sky level estimator used. Allowed options are 'median' (the default), 'mean' and 'mode' (see profoundSkyEstLoc for an explanation of what these estimators do). In all cases this is the estimator applied to unmasked and non-object pixels. If 'doclip'=TRUE then the pixels will be dynamically sigma clipped before the estimator is run.
skyRMStype	Character scalar; the type of sky level estimator used. Allowed options are 'quanlo' (the default), 'quanhi', 'quanboth', and 'sd' (see profoundSkyEstLoc for an explanation of what these estimators do). In all cases this is the estimator applied to unmasked and non-object pixels. If 'doclip'=TRUE then the pixels will be dynamically sigma clipped before the estimator is run.
roughpedestal	Logical; when the initial "rough sky" is computed, should only a pedestal (based on the median of the sky map) be used for the sky? This is a good option if the image is known to contain a *very* large (many times the 'box' size) galaxy that might otherwise be over subtracted by the initial rough sky map.

sigmasel	Numeric scalar; the quantile to use when trying to estimate the true standard-deviation of the sky distribution. If contamination is low then the default of 1 is about optimal in terms of S/N, but you might need to make the value lower when contamination is very high.
skypixmin	Numeric scalar; the minimum number of sky pixels desired in our cutout. The default is that we need half the original number of pixels in the 'box' to be sky.
boxadd	Integer vector; the dimensions to add to the 'box' to capture more pixels if 'skypixmin' has not been achieved.
boxiters	Integer scalar; the number of 'box'+'boxadd' iterations to attempt in order to capture 'skypixmin' sky pixels. The default means the box will not be grown at all.
deblend	Logical; should segment flux be deblended using profoundFluxDeblend and these columns appended to the end of the output 'segstats'?
df	Integer scalar; degrees of freedom for the non-parametric spline fitting. See profoundFluxDeblend.
radtrunc	Numeric scalar; the maximum allowed radius beyond the edge-most segment pixel to consider when deblending. Keeping this low (1-3) ensures segments do not gather flux from very distant regions of the group. See profoundFluxDeblend.
iterative	Logical; should each segment profile fit be subtracted as it goes along? TRUE tends to remove the pedestal from a large galaxy that has faint objects embedded on top. See profoundFluxDeblend.
doclip	Logical; should the unmasked non-object pixels used to estimate to local sky value be further sigma-clipped using magclip? Whether this is used or not is a product of the quality of the objects extraction. If all detectable objects really have been found and the dilated objects mask leaves only apparent sky pixels then an advanced user might be confident enough to set this to FALSE. If in doubt, leave as TRUE.
shiftloc	Logical; should the cutout centre for the sky shift from 'loc' of the desired 'box' size extends beyond the edge of the image? (See magcutout for details).
paddim	Logical; should the cutout be padded with image data until it meets the desired 'box' size (if 'shiftloc' is true) or padded with NAs for data outside the image boundary otherwise? (See magcutout for details).
header	Full FITS header in table or vector format. If this is provided then the segmentations statistics table will gain 'RAcen' and 'Decen' coordinate outputs. Legal table format headers are provided by the read.fitshdr function or the 'hdr' list output of read.fits in the astro package; the 'hdr' output of readFITS in the FITSio package or the 'header' output of magcutoutWCS. Missing header keywords are printed out and other header option arguments are used in these cases. See magWCSxy2radec.
verbose	Logical; should verbose output be displayed to the user? Since big image can take a long time to run, you might want to monitor progress.
plot	Logical; should a diagnostic plot be generated? This is useful when you only have a small number of sources (roughly a few hundred). With more than this it can start to take a long time to make the plot!

stats
: Logical; should statistics on the segmented objects be returned? If 'magzero' and 'pixscale' have been provided then some of the outputs are computed in terms of magnitude and mag/asec^2 rather than flux and flux/pix^2 (see Value).

rotstats
: Logical; if TRUE then the 'asymm', 'flux_reflect' and 'mag_reflect' are computed, else they are set to NA. This is because they are very expensive to compute compared to other photometric properties.

boundstats
: Logical; if TRUE then various pixel boundary statistics are computed ('Nedge', 'Nsky', 'Nobject', 'Nborder', 'edge_frac', 'edge_excess' and 'FlagBorder'). If FALSE these return NA instead (saving computation time).

nearstats
: Logical; if TRUE then the IDs of nearby segments is calculated via profoundSegimNear and output to the returned object 'near'. By default this option is linked to 'boundstats', i.e. it is assumed if you want boundary statistics then you probably also want nearby object IDs returned.

groupstats
: Logical; if TRUE then the IDs of grouped dilated segments (based on the output 'segim') is calculated via profoundSegimGroup and output to the list object 'group'. By default this option is linked to 'boundstats', i.e. it is assumed if you want boundary statistics then you probably also want grouped object information returned. If 'stats'=TRUE is also set then this flag will also create the 'groupstats' output of photometric properties of the groups.

group
: List; you can pass in the output from profoundSegimGroup directly, meaning groups will not be re-computed internally. This might be useful for speed in certain matched photometry applications.

groupby
: Character scalar; How should the grouped segmentation map be formed that will be used to produce the 'groupstats' output? Options are either via 'segim' or 'segim_orig'. 'segim' will create more groups, 'segim_orig' will have less.

offset
: Integer scalar; the distance to offset when searching for nearby segments (used in both profoundSegimStats and profoundSegimNear).

haralickstats
: Logical; if TRUE then the Haralick texture statistics are computed using the EBImage function computeFeatures.haralick. For more detail see the original paper: http://haralick.org/journals/TexturalFeatures.pdf, and a useful online EBImage document: http://earlglynn.github.io/RNotes/package/EBImage/Haralick-Textural-Features.html.

sortcol
: Character; name of the output column that the returned segmentation statistics data.frame should be sorted by (the default is segID, i.e. segment order). See below for column names and contents.

decreasing
: Logical; if FALSE (default) the segmentation statistics data.frame will be sorted in increasing order, if TRUE the data.frame will be sorted in decreasing order.

lowmemory
: Logical; if TRUE then a low memory mode of ProFound will be used. This limits the large 'image' pixel matched outputs to just 'segim', with 'segim_orig', 'objects' and 'objects_redo' set to NULL, and 'sky' and 'skyRMS' set to 0. Internally the sky and skyRMS are used as normal for flux estimates, but they are removed as soon as possible within the function in order to free up memory.

keepim
: Logical; if TRUE then the input 'image' and 'mask' matrices are passed through to the image output of the function. If FALSE then this is set to NULL.

R50clean	Numeric scalar; setting this to more than 0 cleans sources for spuriously small objects. This value should be in arc-seconds if pixel scale is provided or detected, and in pixels otherwise (or if 'pixscale'=1 is explictly set).
watershed	Character; the funciton to use to achieve the watershed deblend. Allowed options are 'EBImage' for EBImage::watershed, and 'ProFound' for the new Rcpp implementation included with the ProFound package.
...	Further arguments to be passed to magimage. Only relevant is 'plot'=TRUE.

Details

This high level function is both a source detection and a segmented aperture growing function. The latter is achieved through consecutive dilation and flux measurement operations. It is not super fast, but it is designed to be fairly robust and fast enough for most use cases.

profoundProFound initially makes a segmentation map using the profoundMakeSegim function. It then makes repeated dilations and flux measurements of this segmentation map using profoundMakeSegimDilate, and calculates the convergent flux segment for each source. These are combined to make a final segmentation map with associated source statistics (if requested).

The defaults should work reasonably well on modern survey data (see Examples), but should the solution not be ideal try modifying these parameters (in order of impact priority): 'skycut', 'pixcut', 'tolerance', 'sigma', 'ext'.

profoundMakeSegimDilate is similar in nature to the pixel growing objmask routine in IRAF (see the 'ngrow' and 'agrow' description at http://stsdas.stsci.edu/cgi-bin/gethelp.cgi?objmasks). This similarity was discovered after implementation, but it is worth noting that the higher level curve of growth function profoundProFound is not trivially replicated by other astronomy tools.

Value

An object list of class 'profound' containing:

segim	Integer matrix; the dilated and converged segmentation map matched pixel by pixel to 'image'.
segim_orig	Integer matrix; the pre-dilated segmentation map matched pixel by pixel to 'image'.
objects	Logical matrix; the object map matched pixel by pixel to 'image'. 1 means there is an object at this pixel, 0 means it is a sky pixel. Can be used as a mask in various other functions that require objects to be masked out.
objects_redo	Logical matrix; the dilated object map matched pixel by pixel to 'image'. See 'redosky' and 'redoskysize'. Can be used as a mask in various other functions that require objects to be masked out.
sky	The estimated sky level of the 'image'.
skyRMS	The estimated sky RMS of the 'image'.
image	The input 'image' matrix if 'keepim'=TRUE, else NULL.
mask	The input 'mask' matrix if 'keepim'=TRUE, else NULL.
segstats	If 'stats'=TRUE then contains the output of profoundSegimStats run using the final 'segim' (see below), otherwise NULL.

Nseg	The total number of segments extracted (dim(segstats)[1]).
near	If 'nearstats'=TRUE then contains the output of profoundSegimNear.
group	If 'groupstats'=TRUE then contains the output of profoundSegimGroup.
groupstats	If 'groupstats'=TRUE and 'stats'=TRUE then contains the output of profoundSegimStats run using the final 'group$groupim' (see below), otherwise NULL.
header	The header provided, if missing this is NULL.
SBlim	The surface brightness limit of detected objects. Requires at least 'magzero' to be provided and 'skycut'>0, else NULL. profoundMakeSegimExpand only.
magzero	The assumed magnitude zero point. This is relevant to various outputs returned by the segmentation statistics.
dim	The dimensions of the processed image.
pixscale	The assumed pixel scale. This is relevant to various outputs returned by the segmentation statistics.
gain	The assumed image gain (if NULL it was not used). This is relevant to various outputs returned by the segmentation statistics.
call	The original function call.
date	The date, more specifically the output of date.
time	The elapsed run time in seconds.
ProFound.version	The version of ProFound run, more specifically the output of packageVersion('ProFound').
R.version	The version of R run, more specifically the output of R.version.

If 'stats'=TRUE then the function profoundSegimStats is called and the 'segstats' part of the returned list will contain a data.frame with columns (else NULL):

segID	Segmentation ID, which can be matched against values in 'segim'
uniqueID	Unique ID, which is fairly static and based on the xmax and ymax position
xcen	Flux weighted x centre
ycen	Flux weighted y centre
xmax	x position of maximum flux
ymax	y position of maximum flux
RAcen	Flux weighted degrees Right Ascension centre (only present if a 'header' is provided)
Deccen	Flux weighted degrees Declination centre (only present if a 'header' is provided)
RAmax	Right Ascension of maximum flux (only present if a 'header' is provided)
Decmax	Declination of maximum flux (only present if a 'header' is provided)
sep	Radial offset between the cen and max definition of the centre (units of 'pixscale', so if 'pixscale' represents the standard asec/pix this will be asec)
flux	Total flux (calculated using 'image'-'sky') in ADUs
mag	Total flux converted to mag using 'magzero'

cenfrac	Fraction of flux in the brightest pixel
N50	Number of brightest pixels containing 50% of the flux
N90	Number of brightest pixels containing 90% of the flux
N100	Total number of pixels in this segment, i.e. contains 100% of the flux
R50	Approximate elliptical semi-major axis containing 50% of the flux (units of 'pixscale', so if 'pixscale' represents the standard asec/pix this will be asec)
R90	Approximate elliptical semi-major axis containing 90% of the flux (units of 'pixscale', so if 'pixscale' represents the standard asec/pix this will be asec)
R100	Approximate elliptical semi-major axis containing 100% of the flux (units of 'pixscale', so if 'pixscale' represents the standard asec/pix this will be asec)
SB_N50	Mean surface brightness containing brightest 50% of the flux, calculated as 'flux'*0.5/'N50' (if 'pixscale' has been set correctly then this column will represent mag/asec^2. Otherwise it will be mag/pix^2)
SB_N90	Mean surface brightness containing brightest 90% of the flux, calculated as 'flux'*0.9/'N90' (if 'pixscale' has been set correctly then this column will represent mag/asec^2. Otherwise it will be mag/pix^2)
SB_N100	Mean surface brightness containing all of the flux, calculated as 'flux'/'N100' (if 'pixscale' has been set correctly then this column will represent mag/asec^2. Otherwise it will be mag/pix^2)
xsd	Weighted standard deviation in x (always in units of pix)
ysd	Weighted standard deviation in y (always in units of pix)
covxy	Weighted covariance in xy (always in units of pix)
corxy	Weighted correlation in xy (always in units of pix)
con	Concentration, 'R50'/'R90'
asymm	180 degree flux asymmetry (0-1, where 0 is perfect symmetry and 1 complete asymmetry)
flux_reflect	Flux corrected for asymmetry by doubling the contribution of flux for asymmetric pixels (defined as no matching segment pixel found when the segment is rotated through 180 degrees)
mag_reflect	'flux_reflect' converted to mag using 'magzero'
semimaj	Weighted standard deviation along the major axis, i.e. the semi-major first moment, so ~2 times this would be a typical major axis Kron radius (always in units of pix)
semimin	Weighted standard deviation along the minor axis, i.e. the semi-minor first moment, so ~2 times this would be a typical minor axis Kron radius (always in units of pix)
axrat	Axial ratio as given by min/maj
ang	Orientation of the semi-major axis in degrees. This has the convention that 0= | (vertical), 45= \\, 90= - (horizontal), 135= /, 180= | (vertical)
signif	Approximate singificance of the detection using the Chi-Square distribution
FPlim	Approximate false-positive significance limit below which one such source might appear spuriously on an image this large

flux_err	Estimated total error in the flux for the segment
mag_err	Estimated total error in the magnitude for the segment
flux_err_sky	Sky subtraction component of the flux error
flux_err_skyRMS	Sky RMS component of the flux error
flux_err_shot	Object shot-noise component of the flux error (only if 'gain' is provided)
sky_mean	Mean flux of the sky over all segment pixels
sky_sum	Total flux of the sky over all segment pixels
skyRMS_mean	Mean value of the sky RMS over all segment pixels
Nedge	Number of edge segment pixels that make up the outer edge of the segment
Nsky	Number of edge segment pixels that are touching sky
Nobject	Number of edge segment pixels that are touching another object segment
Nborder	Number of edge segment pixels that are touching the 'image' border
Nmask	Number of edge segment pixels that are touching a masked pixel (note NAs in 'image' are also treated as masked pixels)
edge_frac	Fraction of edge segment pixels that are touching the sky i.e. 'Nsky''Nedge', higher generally meaning more robust segmentation statistics
edge_excess	Ratio of the number of edge pixels to the expected number given the elliptical geometry measurements of the segment. If this is larger than 1 then it is a sign that the segment geometry is irregular, and is likely a flag for compromised photometry
flag_border	A binary flag telling the user which 'image' borders the segment touches. The bottom of the 'image' is flagged 1, left=2, top=4 and right=8. A summed combination of these flags indicate the segment is in a corner touching two borders: bottom-left=3, top-left=6, top-right=12, bottom-right=9.
iter	The iteration number when the source was flagged as having convergent flux
origfrac	The ratio between the final converged flux and the initial profoundMakeSegim iso-contour estimate
Norig	Number of pixels in the non-dilated (i.e. original) segment. This will be >= 'pixcut' by construction.
flag_keep	A suggested flag for selecting good objects. Objects flagged FALSE have hit the iteration limit and have grown their flux by more than the median for all objects at the iteration limit.

Author(s)

Aaron Robotham

References

Robotham A.S.G., et al., 2018, MNRAS, 476, 3137 Haralick R.M., et al., 1973, IEEE, SMC-3 (6), 610

See Also

profoundMakeSegim, profoundMakeSegimDilate, profoundMakeSegimExpand, profoundMakeSegimPropagate, profoundSegimStats, profoundSegimPlot, profoundMultiBand

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits', package="ProFound"))

profound=profoundProFound(image, magzero=30, verbose=TRUE, plot=TRUE)

# You can check to see if the final objects mask is aggressive enough. Notice the halos
# surrounding bright sources when just using the objects mask.

temp=image$imDat
temp[profound$objects>0]=0
magimage(temp)
temp=image$imDat
temp[profound$objects_redo>0]=0
magimage(temp)

magplot(profound$segstats[,c("R50","SB_N90")], log='x', grid=TRUE)
magplot(profound$segstats[,c("R50","SB_N90")], log='x', grid=TRUE)

magplot(profound$segstats[,c("flux","origfrac")], log='x', grid=TRUE)

## An example of a large galaxy:

VST_r=readFITS(system.file("extdata", 'VST_r.fits', package="magicaxis"))

# Running on defaults results in the central galaxy subtracting itself:
plot(profoundProFound(VST_r))

# Setting boxters=2 fixes things nicely:
plot(profoundProFound(VST_r, boxiters=2))

## End(Not run)
```

---

profoundCatMerge *Catalogue Merging Tool*

Description

Merges segmentation and grouped segmentation catalogues based on which groups are preferred.

Usage

```
profoundCatMerge(segstats = NULL, groupstats = NULL, groupsegID = NULL,
groupID_merge = NULL, flag = TRUE, rowreset = FALSE)
```

Arguments

- `segstats` — Data.frame, segmentation catalogue output from ‘profoundProFound’.
- `groupstats` — Data.frame, grouped segmentation catalogue output from ‘profoundProFound’.
- `groupsegID` — List; group information as output by ‘profoundSegimGroup’ or ‘profoundProFound’. Must correspond to the supplied ‘segstats’ and ‘groupstats’.
- `groupID_merge` — Integer vector; group IDs that are preferred solutions. All segmented belonging to the corresponding group will be removed, and the new group photometry inserted instead.
- `flag` — Logical; should an extra column be added to the end specifying the origin of the photometry (either ’seg’ for the segmentation map, or ’group’ for the grouped segmentation map)?
- `rowreset` — Logical; should the data.frame row names be reset to be 1:Nrow of the data.frame? The default leaves a trace of the group segment selection (i.e. you can see the selected row numbers from the provided ‘segstats’).

Details

Handy tool to robustly merge catalogues based on preferred solutions.

Value

Merged catalogue. This will have the same number of columns as ‘segstats’, with an additional column at the end called ’origin’ that flags whether the object came from the segmentation catalogue (seg) or grouped segmentation catalogue (group).

Author(s)

Aaron Robotham

See Also

profoundSegimKeep

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits', package="ProFound"))
profound=profoundProFound(image, skycut=1.5, magzero=30, groupstats=TRUE, verbose=TRUE)

merge=profoundCatMerge(profound$segstats, profound$groupstats,
profound$group$groupsegID, 1)

profound$segstats[1,'mag']
merge[1,'mag'] #The merged object is brighter, as we should expect.

## End(Not run)
```

profoundDrawEllipse *Draw Ellipse*

Description

Draws multiple ellipses on a plot window.

Usage

```
profoundDrawEllipse(xcen = 0, ycen = 0, rad = 1, axrat = 1, ang = 0, box = 0, ...)
```

Arguments

`xcen`	Numeric vector; x centre/s of the ellipse/s.
`ycen`	Numeric vector; y centre/s of the ellipse/s.
`rad`	Numeric vector; the major axis extent of the ellipse/s.
`axrat`	Numeric vector; the axial ratio of the ellipse/s as given by 'radlo'/'radhi'.
`ang`	Numeric vector; the angle of the ellipse/s in the usual ProFit sense, see `profitMakeModel`.
`box`	Numeric vector; the boxiness of the ellipse/s in the usual ProFit sense, see `profitMakeModel`.
`...`	Further arguments to be passed to `lines` to draw the ellipse/s.

Details

This function uses all the standard `ProFit` conventions to define the input parameters

Value

No value is returned, this function is run purely for the side effect of drawing an ellipse.

Author(s)

Aaron Robotham

See Also

`profoundGetEllipsesPlot`, `profoundGetEllipses`, `profoundGetEllipse`

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits', package="ProFound"))

profound=profoundProFound(image, magzero=30, verbose=TRUE, plot=TRUE)
profoundDrawEllipse(profound$segstats$xcen, profound$segstats$ycen,
profound$segstats$R100/0.339, profound$segstats$axrat, profound$segstats$ang,
col='white', lty=2)
```

```
## End(Not run)
```

`profoundFlux2Mag` *Convert between fluxes and magnitudes.*

Description

Simple functions to concert between magnitudes and flux given a certain magnitude zero-point.

Usage

```
profoundFlux2Mag(flux = 1, magzero = 0)
profoundMag2Flux(mag = 0, magzero = 0)
profoundFlux2SB(flux = 1, magzero = 0, pixscale = 1)
profoundSB2Flux(SB = 0, magzero = 0, pixscale = 1)
```

Arguments

`flux`	Numeric scalar/vector; flux in ADUs given the 'magzero'.
`mag`	Numeric scalar/vector; magnitude given the 'magzero'.
`magzero`	Numeric scalar/vector; magnitude zero point. What this implies depends on the magnitude system being used (e.g. AB or Vega).
`SB`	Numeric scalar/vector; surface brightness in mag/asec^2.
`pixscale`	Numeric scalar/vector; the pixel scale, where pixscale=asec/pix (e.g. 0.4 for SDSS). If set to 1, then the output is in terms of pixels, otherwise it is in arcseconds.

Details

These functions are here to prevent silly mistakes, but the conversion is almost trivial.

Value

`profoundFlux2Mag`
Returns the magnitude, where 'mag' = -2.5 * log10('flux') + 'magzero')

`profoundMag2Flux`
Returns the flux, where 'flux' = 10^(-0.4 * ('mag' - 'magzero'))

HERE!!!

Author(s)

Aaron Robotham

See Also

`profoundGainConvert`

Examples

```
profoundFlux2Mag(1e5, 30)
profoundMag2Flux(17.5, 30)
```

`profoundFluxDeblend` *Mid Level Image Deblender*

Description

Given a target image, a segmentation map, image segstats and group properties, this function will attempt a non-parametric deblend based on local fitting of B-splines to create a weight map for each segment in a group. Flux is guaranteed to be conserved, and errors are appropriately rescaled.

Usage

```
profoundFluxDeblend(image = NULL, segim = NULL, segstats = NULL, groupim = NULL,
groupsegID = NULL, magzero = 0,  df = 3, radtrunc = 2, iterative = FALSE,
doallstats = TRUE)
```

Arguments

`image`	Numeric matrix; required, the image we want to analyse. As a convenience you can supply the output of `profoundProFound` of class profound, in which case any required input that is not explicitly set via the arguments will be inherited from the `profoundProFound` list.
`segim`	Integer matrix; a specified segmentation map of the image. This matrix *must* be the same dimensions as '`image`' if supplied. If this option is used then `profoundProFound` will not compute its initial segmentation map using `profoundMakeSegim`, which is then dilated. Instead it will use the one passed through '`segim`'.
`segstats`	Data.frame, segmentation catalogue output from '`profoundProFound`'.
`groupim`	Integer matrix; the grouped segmentation map. This matrix *must* be the same dimensions as '`image`'.
`groupsegID`	List; group information as output by '`profoundSegimGroup`' or '`profoundProFound`'. Must correspond to the supplied '`segstats`'.
`magzero`	Numeric scalar; the magnitude zero point.
`df`	Integer scalar; degrees of freedom for the non-parametric spline fitting. See `smooth.spline`.
`radtrunc`	Numeric scalar; the maximum allowed radius beyond the edge-most segment pixel to consider when deblending. Keeping this low (1-3) ensures segments do not gather flux from very distant regions of the group.
`iterative`	Logical; should each segment profile fit be subtracted as it goes along? TRUE tends to remove the pedestal from a large galaxy that has faint objects embedded on top.
`doallstats`	Logical; specifies whether the output catalogue is matched against all rows of the supplied '`segstats`' (TRUE), or only the rows containing grouped (and therefore deblended) galaxies are returned and the core flux columns (see below)s.

Details

This routine only deblends with detected groups, so it is quite fast if the number of groups is quite low. If the image is more confused then this process can be quite slow.

Value

A data.frame containing deblended flux information:

flux_db	Total flux (calculated using 'image'-'sky') in ADUs
mag_db	Total flux converted to mag using 'magzero'
N100_db	Total number of pixels in this segment, i.e. contains 100% of the flux

The below are only returned if 'doallstats'=TRUE:

flux_err_db	Estimated total error in the flux for the segment
mag_err_db	Estimated total error in the magnitude for the segment
flux_err_sky_db	Sky subtraction component of the flux error
flux_err_skyRMS_db	Sky RMS component of the flux error
flux_err_shot_db	Object shot-noise component of the flux error (only if 'gain' is provided)

Note

Given the large number of inputs required, this function effectively needs profoundProFound to be run first.

Author(s)

Aaron Robotham

See Also

profoundProFound, smooth.spline

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits', package="ProFound"))

profound=profoundProFound(image, magzero=30, verbose=TRUE, plot=TRUE)

deblend=profoundFluxDeblend(profound)

plot(profound$segstats$mag, deblend$mag_db)

## End(Not run)
```

profoundGainConvert *Convert gain between mag-zero points*

Description

Simple function to update the gain (electrons/ADU) when changing between magnitude zero points. These gains are what should be passed to e.g. profoundMakeSigma.

Usage

```
profoundGainConvert(gain = 1, magzero = 0, magzero_new = 0)
```

Arguments

gain	Numeric scalar or vector; the current gain/s in electrons/ADU.
magzero	Numeric scalar or vector; the current magnitude zero point/s.
magzero_new	Numeric scalar or vector; the new magnitude zero point/s.

Details

A simple function that is mostly here to avoid silly conversion mistakes. The conversion is calculated as: gain*10^(-0.4*(magzero_new - magzero)), where an object magnitude can be calculated from ADU flux as -2.5*log10(flux_ADU)+magzero.

Value

Numeric scalar or vector; the new gain/s.

Author(s)

Aaron Robotham

See Also

profoundMakeSigma, profoundFlux2Mag, profoundMag2Flux

Examples

```
#For optical survey data typically images with gain~1 have a magzero~30:
profoundGainConvert(1,30,0)
```

`profoundGainEst` *Image Gain Estimator*

Description

High level function to estimate a rough value for the image gain in cases where you have no idea what the true image gain is. In practice this tends to be accurate to an order of magnitude and provides a reasonable lower limit for the true gain, which is good enough to make a rough first attempt at a sigma map.

Usage

```
profoundGainEst(image = NULL, mask = 0, objects = 0, sky = 0, skyRMS = 1)
```

Arguments

`image`	Numeric matrix; required, the image we want to analyse.
`mask`	Boolean matrix; optional, non galaxy parts of the image to mask out, where 1 means mask out and 0 means use for analysis. If provided, this matrix *must* be the same dimensions as 'image'.
`objects`	Boolean matrix; optional, object mask where 1 is object and 0 is sky. Pixels set to 0 are interpreted as sky, and set to zero for calculating object shot-noise. If provided, this matrix *must* be the same dimensions as 'image'.
`sky`	Numeric scalar; user provided estimate of the absolute sky level. If this is not provided then it will be computed internally using `profoundSkyEst`.
`skyRMS`	Numeric scalar; user provided estimate of the RMS of the sky. If this is not provided then it will be computed internally using `profoundSkyEst`.

Details

This function makes use of the fact that a true Poisson distribution cannot generate samples below 0 and the distribution shape properties of the sky pixels. In practice this means the gain estimated is low as it can be. Once the ProFit fit has been made the gain estimated can be improved based on the residuals (assuming the model does a good job of subtracting the data).

Value

Numeric scalar; the estimated gain of the 'image'.

Author(s)

Aaron Robotham

See Also

`profoundMakeSegim`, `profoundMakeSegimExpand`, `profoundSkyEst`, `profoundMakeSigma`

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits',
package="ProFound"))
profound=profoundProFound(image)
profoundGainEst(image$imDat, objects=profound$objects_redo, sky=profound$sky)

## End(Not run)
```

`profoundGetEllipse` *Calculate single annulus properties of an iso-photal ellipse*

Description

Returns single ellipse properties for a specific set of pixels, assumed to be be a narrow range in flux (i.e. an iso-photal annulus).

Usage

```
profoundGetEllipse(x, y, z, xcen = NULL, ycen = NULL, scale = sqrt(2), pixscale = 1,
dobox = FALSE, plot=FALSE, ...)
```

Arguments

`x`	Numeric vector; x values of pixels. If this is a 3 column matrix then column 1 is used for '`x`', column 2 is used for '`y`' and column 3 is used for '`val`', see Examples.
`y`	Numeric vector; y values of pixels.
`z`	Numeric vector; z values of pixels. This is effectively the height, and would be the pixel flux for an image.
`xcen`	Numeric scalar; the desired x centre of the ellipse. If this is not provided it is calculated internally.
`ycen`	Numeric scalar; the desired y centre of the ellipse. If this is not provided it is calculated internally.
`scale`	How should the standard ellipse covariance be scaled to create a geometric ellipse. The default of sqrt(2) is appropriate to create an ellipse that represents the location of an iso-photal contour of a galaxy.
`pixscale`	Numeric scalar; the pixel scale, where pixscale=asec/pix (e.g. 0.4 for SDSS). If set to 1 (default), then the output '`radhi`', '`radlo`' and '`radav`' is in terms of pixels, otherwise they are in arcseconds.
`dobox`	Logical; should boxiness be computed? If FALSE then boxiness is fixed to be 0. If TRUE then boxiness is computed (and other parameters are refined) using a maximum likelihood method. This is more expensive to compute, so the default is FALSE.

plot
: Logical; should an ellipse be drawn on the current plot? This plot is generated by the profoundDrawEllipse function.

...
: Further arguments to be passed to profoundDrawEllipse. Only relevant is 'plot'=TRUE.

Details

The assumption is this function will largely be used by the profoundGetEllipses function, but it could be useful for computing the shape of a particular iso-photal contour (see Examples).

Value

A numeric vector with the following named elements:

xcen	The flux weighted x centre of the ellipse.
ycen	The flux weighted y centre of the ellipse.
radhi	The major axis extent of the ellipse (units of 'pixscale', so if 'pixscale' represents the standard asec/pix this will be asec).
radlo	The minor axis extent of the ellipse (units of 'pixscale', so if 'pixscale' represents the standard asec/pix this will be asec).
radav	The average radius of the ellipse (units of 'pixscale', so if 'pixscale' represents the standard asec/pix this will be asec).
axrat	The axial ratio of the ellipse as given by 'radlo'/'radhi'.
ang	The angle of the ellipse in the usual ProFit sense, see profitMakeModel.
box	The boxiness of the ellipse in the usual ProFit sense, see profitMakeModel.
xsd	The flux weighted standard deviation in x (always in units of pix).
xsd	The flux weighted standard deviation in y (always in units of pix).
covxy	The flux weighted covariance in xy (always in units of pix).
corxy	The flux weighted correlation in xy (always in units of pix).

Author(s)

Aaron Robotham

See Also

profoundGetEllipses, profoundGetEllipsesPlot

Examples

```
## Not run:
# We need the ProFit library to show the profile: library(ProFit)
image = readFITS(system.file("extdata", 'KiDS/G266035fitim.fits',
package="ProFit"))$imDat
tempxy=cbind(which(image>2e-11 & image<3e-11, arr.ind=TRUE)-0.5,
             image[image>2e-11 & image<3e-11])
magimage(image>2e-11 & image<3e-11)
```

```
points(tempxy[,1:2], pch='.', col='red')
tempellipse=profoundGetEllipse(tempxy)
profoundDrawEllipse(tempellipse['xcen'], tempellipse['ycen'], tempellipse['radhi'],
                    tempellipse['axrat'], tempellipse['ang'], col='blue')

## End(Not run)
```

---

profoundGetEllipses *Calculate multiple annulus properties of iso-photal ellipses*

Description

Returns multiple ellipse properties for an image, assumed to be monotonically decreasing in flux from a bright centre (i.e. a classic galaxy).

Usage

```
profoundGetEllipses(image = NULL, segim = NULL, segID = 1, levels = 10, magzero = 0,
pixscale = 1, fixcen = TRUE, dobox = FALSE, plot = TRUE, ...)
```

Arguments

image	Numeric matrix; required, the image we want to analyse.
segim	Integer matrix; optional, the segmentation map of the image. This matrix *must* be the same dimensions as 'image'.
segID	Integer scalar; optional, the desired 'segim' segment to extract from the 'image'.
levels	Integer scalar or vector. If a scalar this is the number of ellipse levels to extract from the 'image'. If a vector this specfies the extremes of all fractional levels, i.e. it should generally start at 0 and end at 1 to capture all isophotal levels.
magzero	Numeric scalar; the magnitude zero point. What this implies depends on the magnitude system being used (e.g. AB or Vega). If provided along with 'pixscale' then the surface brightness output will represent mag/asec^2.
pixscale	Numeric scalar; the pixel scale, where pixscale=asec/pix (e.g. 0.4 for SDSS). If set to 1 (default), then the output 'radhi', 'radlo' and 'radav' is in terms of pixels, otherwise they are in arcseconds. If provided along with 'magzero' then the surface brightness output will represent mag/asec^2.
fixcen	Logical; should the ellipse centres be fixed to a common flux weighted centre?
dobox	Logical; should boxiness be computed? If FALSE then boxiness is fixed to be 0. If TRUE then boxiness is computed (and other parameters are refined) using a maximum likelihood method. This is more expensive to compute, so the default is FALSE.
plot	Logical; should a diagnostic plot be generated? This plot is generated by the profoundGetEllipsesPlot function.
...	Further arguments to be passed to profoundGetEllipsesPlot. Only relevant if 'plot'=TRUE.

Details

This higher level function provides an easy way to extract iso-photal ellipses from an image of a galaxy. How it works somewhat replicates IRAF's ellipse, but it is really present to offer useful initial guesses for bulge and disk geometric properties. It certainly does not guarantee to return the same solution as IRAF (in fact I am not exactly aware of how IRAF computes its ellipses).

Internally it works by rank ordering the pixels of the galaxy and dividing these into equi-spaced quantiles of flux (so each annulus will approximately sum to the same amount of flux). This means that the error for each ellipse will be approximately constant. For each annulus it then runs `profoundGetEllipse` to compute the ellipse properties of what is assumed to be a fairly narrow annulus of pixels. The implicit assumption is that the galaxy flux more-or-less monotonically decreases from the centre, and dividing pixels like this will assure the extraction of common iso-photal ellipses. This assumption works well within the inner 90% of a galaxy's flux, but isophotes can be quite noisy once the galaxy flux gets close to the sky RMS level. This said, the ellipse returned will on average make sense, and ellipses tend to overlap only in very extreme cases (where the geometry is highly non-elliptical or there are close contaminants).

Value

A list containing:

`ellipses` A data.frame of ellipse properties ordered by radius (see below).

`segellipses` Integer matrix; the ellipse-wise segmentation map matched pixel by pixel to '`image`'. This allows you to see which specific pixels used to compute each ellipse annulus in '`ellipses`', where the number in the segmentation map refers to '`segellipseID`'.

'`ellipses`' is a data.frame of ellipse properties ordered by radius. It has the following columns

`segellipseID` The ellipse segment ID that refers to the segmentation map '`segellipses`'.

`fluxfrac` The approximate fraction of galaxy flux contained within this ellipse.

`xcen` The flux weighted x centre of the ellipse.

`ycen` The flux weighted y centre of the ellipse.

`radhi` The major axis extent of the ellipse (units of '`pixscale`', so if '`pixscale`' represents the standard asec/pix this will be asec)..

`radlo` The minor axis extent of the ellipse (units of '`pixscale`', so if '`pixscale`' represents the standard asec/pix this will be asec)..

`radav` The average radius of the ellipse (units of '`pixscale`', so if '`pixscale`' represents the standard asec/pix this will be asec)..

`axrat` The axial ratio of the ellipse as given by '`radlo`'/'`radhi`'.

`ang` The angle of the ellipse in the usual ProFit sense, see `profitMakeModel`.

`box` The boxiness of the ellipse in the usual ProFit sense, see `profitMakeModel`.

`xsd` The flux weighted standard deviation in x (always in units of pix).

`ysd` The flux weighted standard deviation in y (always in units of pix).

`covxy` The flux weighted covariance in xy (always in units of pix).

corxy	The flux weighted correlation in xy (always in units of pix).
flux	The flux contained in the segmented pixels associated with this ellipse.
N	The number of segmented pixels associated with this ellipse.
SB	The mean surface brightness of the pixels associated with this ellipse (if 'pixscale' has been set correctly then this column will represent mag/asec^2, otherwise it will be mag/pix^2).

Author(s)

Aaron Robotham

See Also

profoundGetEllipsesPlot, profoundGetEllipse, profoundDrawEllipse

Examples

```
## Not run:
# We need the ProFit library to show the profile: library(ProFit)
image = readFITS(system.file("extdata", 'KiDS/G278109fitim.fits',
package="ProFit"))$imDat
segim = readFITS(system.file("extdata", 'KiDS/G278109segim.fits',
package="ProFit"))$imDat
ellipses_nobox = profoundGetEllipses(image=image, segim=segim, levels=20, dobox=FALSE,
pixscale=0.2)
ellipses_box = profoundGetEllipses(image=image, segim=segim, levels=20, dobox=TRUE,
pixscale=0.2)

magplot(ellipses_box$ellipses$radhi[4:19], ellipses_nobox$ellipses$SB[4:19],
ylim=c(25,17), grid=TRUE, type='l')
points(ellipses_box$ellipses$radhi[4:19],ellipses_box$ellipses$SB[4:19])
#A rough bulge+disk surface brightness profile (mean axrat~0.6):
rlocs=seq(1,30,by=0.1)
bulge=profitRadialSersic(rlocs, mag=18.2, re=1.7, nser=3)
disk=profitRadialSersic(rlocs, mag=18, re=13, nser=0.7)
lines(rlocs, profoundFlux2SB(bulge, pixscale=0.2), col='red')
lines(rlocs, profoundFlux2SB(disk, pixscale=0.2), col='blue')
lines(rlocs, profoundFlux2SB(bulge+disk, pixscale=0.2), col='green')
#To get correct magnitudes you would need to modify the components by the axrat
#and pixel scale.

#We can do a better 1D fit with ease:
#Since the ellipses are divided by equi-flux we can minimise sum-square of the SB diff:
sumsq1D=function(par=c(17.6, log10(1.7), log10(3), 17.4, log10(13), log10(0.7)),
rad, SB, pixscale=1){
  bulge=profitRadialSersic(rad, mag=par[1], re=10^par[2], nser=10^par[3])
  disk=profitRadialSersic(rad, mag=par[4], re=10^par[5], nser=10^par[6])
  total=profoundFlux2SB(bulge+disk, pixscale=pixscale)
  return=sum((total-SB)^2)
}
```

```
lower=c(10,0,-0.5,10,0,-0.5)
upper=c(30,2,1,30,2,1)

fit1D=optim(sumsq1D, par=c(17.6, log10(1.7), log10(3), 17.4, log10(13), log10(0.7)),
rad=ellipses_box$ellipses$radhi[4:19], SB=ellipses_box$ellipses$SB[4:19], pixscale=0.2,
method='L-BFGS-B', lower=lower, upper=upper)$par

magplot(ellipses_box$ellipses$radhi[4:19], ellipses_nobox$ellipses$SB[4:19],
ylim=c(25,17), grid=TRUE, type='l')
points(ellipses_box$ellipses$radhi[4:19],ellipses_box$ellipses$SB[4:19])
#A simple bulge+disk surface brightness profile:
rlocs=seq(1,30,by=0.1)
bulge=profitRadialSersic(rlocs, mag=fit1D[1], re=10^fit1D[2], nser=10^fit1D[3])
disk=profitRadialSersic(rlocs, mag=fit1D[4], re=10^fit1D[5], nser=10^fit1D[6])
lines(rlocs, profoundFlux2SB(bulge, pixscale=0.2), col='red')
lines(rlocs, profoundFlux2SB(disk, pixscale=0.2), col='blue')
lines(rlocs, profoundFlux2SB(bulge+disk, pixscale=0.2), col='green')

## End(Not run)
```

---

profoundGetEllipsesPlot

Create diagnostic plot of estimated iso-photal ellipses

Description

Generates a useful plot merging a rapidly changing colour mapping with the estimated ellipses.

Usage

```
profoundGetEllipsesPlot(image = NULL, ellipses = NULL, segim = NULL, segID = 1,
segellipseID = "all", pixscale = 1, col = rep(rainbow(10, s = 0.5), 4), border = "auto",
lty = 'auto', lwd = 'auto', ...)
```

Arguments

`image`	Numeric matrix; required, the image we want to analyse.
`ellipses`	Data.frame; the ellipse information, but in practice the '`ellipse`' list output of `profoundGetEllipses`.
`segim`	Integer matrix; optional, the segmentation map of the image. This matrix *must* be the same dimensions as '`image`'.
`segID`	Integer scalar; optional, the desired '`segim`' segment to extract from the '`image`'.
`segellipseID`	Integer vector; the segellipseID to be plotted. The default of 'all' will display all ellipses.
`pixscale`	Numeric scalar; the pixel scale, where pixscale=asec/pix (e.g. 0.4 for SDSS). This should only be used if the radii columns in '`ellipses`' have already been scaled by the pixel scale.

col	The colour palette to be used for the background 'image'. The default is chosen to be high contrast, to make it easier to compare the computed ellipses with the underlying isophotes.
border	The colour of the ellipse border drawn by draw.ellipse. If 'auto' then a sensible default is chosen.
lty	The line type of the ellipse border drawn by draw.ellipse. If 'auto' then a sensible default is chosen ('lty'=1 within the 90% flux radius and 'lty'=2 outside).
lwd	The line width of the ellipse border drawn by draw.ellipse. If 'auto' then a sensible default is chosen ('lwd'=0.5 within the 50% flux radius, 'lwd'=1 above the 50% flux radius, except for the annuli at 50%/90% which is 'lwd'=2).
...	Further arguments to be passed to magimage.

Details

The default options should create useful diagnostics, but there are lots of potential plots that can be made with the outputs of profoundGetEllipses, including e.g. making plots of how various parameters behave with radius, which can give helpful insight to starting parameters for bulge and disk profiles. The user is encouraged to experiment.

Value

No value is returned, this function is run purely for the side effect of making a diagnostic plot.

Author(s)

Aaron Robotham

See Also

profoundGetEllipses, profoundGetEllipse, profoundDrawEllipse

Examples

```
## Not run:
# We need the ProFit library to show the profile: library(ProFit)
image = readFITS(system.file("extdata", 'KiDS/G266035fitim.fits', package="ProFit"))$imDat
segim = readFITS(system.file("extdata", 'KiDS/G266035segim.fits', package="ProFit"))$imDat
ellipses = profoundGetEllipses(image=image, segim=segim, segID=4, plot=FALSE)

#We can get a good overall idea of how good the ellipses are by running with defaults:
profoundGetEllipsesPlot(image=image, ellipses=ellipses$ellipses)

#We can check a specific ellipse too:
profoundGetEllipsesPlot(image=ellipses$segellipses==8, ellipses=ellipses$ellipses,
segellipseID=8, col=grey(0:1), border='red', lwd=2)

## End(Not run)
```

profoundIm *Image Transformations*

Description

Various image transformation functions that assist in exploring data. These all require the imager package to be installed.

Usage

```
profoundImBlur(image = NULL, sigma = 1, plot = FALSE, ...)
profoundImGrad(image = NULL, sigma = 1, plot = FALSE, ...)
profoundImDiff(image = NULL, sigma = 1, plot = FALSE, ...)
```

Arguments

image	Numeric matrix; required, the image we want to analyse.
sigma	Numeric scalar; standard deviation of the blur.
plot	Logical; should a magimage plot of the output be generated?
...	Further arguments to be passed to magimage. Only relevant is 'plot'=TRUE.

Value

Numeric matrix; a new image the same size as 'image', with the relevant transform applied.

For profoundImBlur the output is a smoothed version of the 'image'.

For profoundImGrad the output is the magnitude of the gradient of the smoothed version of the 'image'.

For profoundImDiff the output is the original 'image' minus the smoothed version of the 'image'.

Author(s)

Aaron Robotham

See Also

profoundMakeSegim, profoundMakeSegimExpand

Examples

```
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits',
package="ProFound"))$imDat
magimage(image)
profoundImBlur(image, plot=TRUE)
profoundImGrad(image, plot=TRUE)
profoundImDiff(image, plot=TRUE)
```

`profoundMag2Mu` *Magnitude to Surface Brightness Conversions*

Description

Functions to convert total magnitudes to surface brightness and vica-versa. These are provided to allow models to be either specified by total magnitude or mean surface brightness within Re. The latter is a useful way of specifying a disk model since surface brightness does not span a huge range.

Usage

```
profoundMag2Mu(mag = 15, re = 1, axrat = 1, pixscale = 1)
profoundMu2Mag(mu = 17, re = 1, axrat = 1, pixscale = 1)
```

Arguments

`mag`	Total magnitude of the 2D Sersic profile.
`mu`	Mean surface brightness within Re of the 2D Sersic profile.
`re`	Effective radii of the 2D Sersic profile.
`axrat`	Axial ratio of Sersic profile defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.
`pixscale`	The pixel scale, where pixscale=asec/pix (e.g. 0.4 for SDSS). If set to 1, then the surface brightness is interpreted in terms of pixels, otherwise it is interpreted in terms of arcseconds^2.

Value

profoundMag2Mu returns the mean surface brightness within Re of the 2D Sersic profile.

profoundMag2Mu returns total magnitude of the 2D Sersic profile.

Author(s)

Aaron Robotham

See Also

`profoundSegimStats`

Examples

```
profoundMag2Mu(mag=22, re=10, axrat=0.5)
profoundMu2Mag(mu=28, re=10, axrat=0.5)
```

`profoundMakeSegim` *Watershed Image Segmentation*

Description

A high level utility to achieve decent quality image segmentation. It uses a mixture of image smoothing and watershed segmentation propagation to identify distinct objects for use in, e.g., `profitSetupData` (where the 'segim' list item output of `profoundMakeSegim` would be passed to the 'segim' input of `profitSetupData`).

Usage

```
profoundMakeSegim(image = NULL, mask = NULL, objects = NULL, skycut = 1, pixcut = 3,
tolerance = 4, ext = 2, reltol = 0, cliptol = Inf, sigma = 1, smooth = TRUE, SBlim = NULL,
magzero = 0, gain = NULL, pixscale = 1, sky = NULL, skyRMS = NULL, header = NULL,
verbose = FALSE, plot = FALSE, stats = TRUE, rotstats = FALSE, boundstats = FALSE,
offset = 1, sortcol = "segID", decreasing = FALSE, watershed = 'EBImage', ...)
```

Arguments

`image`	Numeric matrix; required, the image we want to analyse. Note, image NAs are treated as masked pixels.
`mask`	Boolean matrix; optional, parts of the image to mask out (i.e. ignore), where 1 means mask out and 0 means use for analysis. If provided, this matrix *must* be the same dimensions as 'image'.
`objects`	Boolean matrix; optional, object mask where 1 is object and 0 is sky. If provided, this matrix *must* be the same dimensions as 'image'.
`skycut`	Numeric scalar; the lowest threshold to make on the 'image' in units of the sky RMS. Passed to `profoundMakeSegim`.
`pixcut`	Integer scalar; the number of pixels required to identify an object. Passed to `profoundMakeSegim`.
`tolerance`	Numeric scalar; the minimum height of the object in the units of sky RMS between its highest point (seed) and the point where it contacts another object (checked for every contact pixel). If the height is smaller than the tolerance, the object will be combined with one of its neighbours, which is the highest. The range 1-5 offers decent results usually. This is passed to 'tolerance' in 'EBImage', or 'abstol' in 'ProFound' (see 'watershed').
`ext`	Numeric scalar; radius of the neighbourhood in pixels for the detection of neighbouring objects. Higher value smooths out small objects.
`reltol`	Numeric scalar; only relevant for 'watershed'='ProFound'. A modifier to the 'abstol', modifying it by the ratio of the segment peak flux divided by the saddle point flux to the power 'reltol'. The default means the 'reltol' has no effect since this modifier becomes 1. A larger value of 'reltol' means segments are more aggressively merged together.

cliptol	Numeric scalar; only relevant for 'watershed'='ProFound'. If ('image'-'sky')/optionskyRMS is above this level where segments touch then they are always merged, regardless of other criteria. When thinking in terms of sky RMS, values between 20-100 are probably appropriate for merging very bright parts of stars back together in optical data.
sigma	Numeric scalar; standard deviation of the blur used when 'smooth'=TRUE.
smooth	Logical; should smoothing be done on the target 'image'? If present, this will use the imblur function from the imager package. Otherwise it will use the gblur function from the EBImage package with a warning. These functions are very similar in output, but not strictly identical.
SBlim	Numeric scalar; the mag/asec^2 surface brightness threshold to apply. This is always used in conjunction with 'skycut', so set 'skycut' to be very large (e.g. Inf) if you want a pure surface brightness threshold for the segmentation. 'magzero' and 'pixscale' must also be present for this to be used.
magzero	Numeric scalar; the magnitude zero point. What this implies depends on the magnitude system being used (e.g. AB or Vega). If provided along with 'pixscale' then the flux and surface brightness outputs will represent magnitudes and mag/asec^2.
gain	Numeric scalar; the gain (in photo-electrons per ADU). This is only used to compute object shot-noise component of the flux error (else this is set to 0).
pixscale	Numeric scalar; the pixel scale, where pixscale=asec/pix (e.g. 0.4 for SDSS). If set to 1 (default), then the output is in terms of pixels, otherwise it is in arcseconds. If provided along with 'magzero' then the flux and surface brightness outputs will represent magnitudes and mag/asec^2.
sky	User provided estimate of the absolute sky level. If this is not provided then it will be computed internally using profoundSkyEst. Can be a scalar (value uniformly applied to full 'sigma' map) or a matrix matching the dimensions of 'image' (allows values to vary per pixel). This will be subtracted off the 'image' internally, so only provide this if the sky does need to be subtracted!
skyRMS	User provided estimate of the RMS of the sky. If this is not provided then it will be computed internally using profoundSkyEst. Can be a scalar (value uniformly applied to full 'sigma' map) or a matrix matching the dimensions of 'image' (allows values to vary per pixel).
header	Full FITS header in table or vector format. If this is provided then the segmentations statistics table will gain 'RAcen' and 'Decen' coordinate outputs. Legal table format headers are provided by the read.fitshdr function or the 'hdr' list output of read.fits in the astro package; the 'hdr' output of readFITS in the FITSio package or the 'header' output of magcutoutWCS. Missing header keywords are printed out and other header option arguments are used in these cases. See magWCSxy2radec.
verbose	Logical; should verbose output be displayed to the user? Since big image can take a long time to run, you might want to monitor progress.
plot	Logical; should a diagnostic plot be generated? This is useful when you only have a small number of sources (roughly a few hundred). With more than this it can start to take a long time to make the plot!

stats	Logical; should statistics on the segmented objects be returned? If 'magzero' and 'pixscale' have been provided then some of the outputs are computed in terms of magnitude and mag/asec^2 rather than flux and flux/pix^2 (see Value).
rotstats	Logical; if TRUE then the 'asymm', 'flux_reflect' and 'mag_reflect' are computed, else they are set to NA. This is because they are very expensive to compute compared to other photometric properties.
boundstats	Logical; if TRUE then various pixel boundary statistics are computed ('Nedge', 'Nsky', 'Nobject', 'Nborder', 'edge_frac', 'edge_excess' and 'FlagBorder'). If FALSE these return NA instead (saving computation time).
offset	Integer scalar; the distance to offset when searching for nearby segments (used in profoundSegimStats).
sortcol	Character; name of the output column that the returned segmentation statistics data.frame should be sorted by (the default is segID, i.e. segment order). See below for column names and contents.
decreasing	Logical; if FALSE (default) the segmentation statistics data.frame will be sorted in increasing order, if TRUE the data.frame will be sorted in decreasing order.
watershed	Character; the funciton to use to achieve the watershed deblend. Allowed options are 'EBImage' for EBImage::watershed, and 'ProFound' for the new Rcpp implementation included with the ProFound package.
...	Further arguments to be passed to magimage. Only relevant is 'plot'=TRUE.

Details

To use this function you will need to have EBImage installed. Since this can be a bit cumbersome on some platforms (given its dependencies) this is only listed as a suggested package. You can have a go at installing it by running:

> source("http://bioconductor.org/biocLite.R")

> biocLite("EBImage")

Linux users might also need to install some non-standard graphics libraries (depending on your install). If you do not have them already, you should look to install **jpeg** and **tiff** libraries (these are apparently technically not entirely free, hence not coming by default on some strictly open source Linux variants).

The profoundMakeSegim function offers a high level internal to R interface for making quick segmentation maps. The defaults should work reasonably well on modern survey data (see Examples), but should the solution not be ideal try modifying these parameters (in order of impact priority): 'skycut', 'pixcut', 'tolerance', 'sigma', 'ext'.

Value

A list containing:

segim	Integer matrix; the segmentation map matched pixel by pixel to 'image'.
objects	Logical matrix; the object map matched pixel by pixel to 'image'. 1 means there is an object at this pixel, 0 means it is a sky pixel. Can be used as a mask in various other functions that require objects to be masked out.

sky — The estimated sky level of the ‘image’.

skyRMS — The estimated sky RMS of the ‘image’.

segstats — If ‘stats’=TRUE this is a data.frame (see below), otherwise NULL.

header — The header provided, if missing this is NULL.

SBlim — The surface brightness limit of detected objects (requires at least ‘magzero’ to be provided and ‘skycut’>0, else NULL).

call — The original function call.

If ‘stats’=TRUE then the function profoundSegimStats is called and the ‘segstats’ part of the returned list will contain a data.frame with columns (else NULL):

segID — Segmentation ID, which can be matched against values in ‘segim’

uniqueID — Unique ID, which is fairly static and based on the xmax and ymax position

xcen — Flux weighted x centre

ycen — Flux weighted y centre

xmax — x position of maximum flux

ymax — y position of maximum flux

RAcen — Flux weighted degrees Right Ascension centre (only present if a ‘header’ is provided)

Deccen — Flux weighted degrees Declination centre (only present if a ‘header’ is provided)

RAmax — Right Ascension of maximum flux (only present if a ‘header’ is provided)

Decmax — Declination of maximum flux (only present if a ‘header’ is provided)

sep — Radial offset between the cen and max definition of the centre (units of ‘pixscale’, so if ‘pixscale’ represents the standard asec/pix this will be asec)

flux — Total flux (calculated using ‘image’-‘sky’) in ADUs

mag — Total flux converted to mag using ‘magzero’

cenfrac — Fraction of flux in the brightest pixel

N50 — Number of brightest pixels containing 50% of the flux

N90 — Number of brightest pixels containing 90% of the flux

N100 — Total number of pixels in this segment, i.e. contains 100% of the flux

R50 — Approximate elliptical semi-major axis containing 50% of the flux (units of ‘pixscale’, so if ‘pixscale’ represents the standard asec/pix this will be asec)

R90 — Approximate elliptical semi-major axis containing 90% of the flux (units of ‘pixscale’, so if ‘pixscale’ represents the standard asec/pix this will be asec)

R100 — Approximate elliptical semi-major axis containing 100% of the flux (units of ‘pixscale’, so if ‘pixscale’ represents the standard asec/pix this will be asec)

SB_N50 — Mean surface brightness containing brightest 50% of the flux, calculated as ‘flux’*0.5/‘N50’ (if ‘pixscale’ has been set correctly then this column will represent mag/asec^2. Otherwise it will be mag/pix^2)

SB_N90
: Mean surface brightness containing brightest 90% of the flux, calculated as 'flux'*0.9/'N90' (if 'pixscale' has been set correctly then this column will represent mag/asec^2. Otherwise it will be mag/pix^2)

SB_N100
: Mean surface brightness containing all of the flux, calculated as 'flux'/'N100' (if 'pixscale' has been set correctly then this column will represent mag/asec^2. Otherwise it will be mag/pix^2)

xsd
: Weighted standard deviation in x (always in units of pix)

ysd
: Weighted standard deviation in y (always in units of pix)

covxy
: Weighted covariance in xy (always in units of pix)

corxy
: Weighted correlation in xy (always in units of pix)

con
: Concentration, 'R50'/'R90'

asymm
: 180 degree flux asymmetry (0-1, where 0 is perfect symmetry and 1 complete asymmetry)

flux_reflect
: Flux corrected for asymmetry by doubling the contribution of flux for asymmetric pixels (defined as no matching segment pixel found when the segment is rotated through 180 degrees)

mag_reflect
: 'flux_reflect' converted to mag using 'magzero'

semimaj
: Weighted standard deviation along the major axis, i.e. the semi-major first moment, so ~2 times this would be a typical major axis Kron radius (always in units of pix)

semimin
: Weighted standard deviation along the minor axis, i.e. the semi-minor first moment, so ~2 times this would be a typical minor axis Kron radius (always in units of pix)

axrat
: Axial ratio as given by min/maj

ang
: Orientation of the semi-major axis in degrees. This has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical)

signif
: Approximate singificance of the detection using the Chi-Square distribution

FPlim
: Approximate false-positive significance limit below which one such source might appear spuriously on an image this large

flux_err
: Estimated total error in the flux for the segment

mag_err
: Estimated total error in the magnitude for the segment

flux_err_sky
: Sky subtraction component of the flux error

flux_err_skyRMS
: Sky RMS component of the flux error

flux_err_shot
: Object shot-noise component of the flux error (only if 'gain' is provided)

sky_mean
: Mean flux of the sky over all segment pixels

sky_sum
: Total flux of the sky over all segment pixels

skyRMS_mean
: Mean value of the sky RMS over all segment pixels

Nedge
: Number of edge segment pixels that make up the outer edge of the segment

Nsky
: Number of edge segment pixels that are touching sky

Nobject
: Number of edge segment pixels that are touching another object segment

Nborder	Number of edge segment pixels that are touching the 'image' border
Nmask	Number of edge segment pixels that are touching a masked pixel (note NAs in 'image' are also treated as masked pixels)
edge_frac	Fraction of edge segment pixels that are touching the sky i.e. 'Nsky''Nedge', higher generally meaning more robust segmentation statistics
edge_excess	Ratio of the number of edge pixels to the expected number given the elliptical geometry measurements of the segment. If this is larger than 1 then it is a sign that the segment geometry is irregular, and is likely a flag for compromised photometry
flag_border	A binary flag telling the user which 'image' borders the segment touches. The bottom of the 'image' is flagged 1, left=2, top=4 and right=8. A summed combination of these flags indicate the segment is in a corner touching two borders: bottom-left=3, top-left=6, top-right=12, bottom-right=9.

Author(s)

Aaron Robotham

References

See ?EBImage::watershed

See Also

profoundMakeSegimExpand, profoundProFound, profoundSegimStats, profoundSegimPlot

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits',
package="ProFound"))$imDat
segim=profoundMakeSegim(image, plot=TRUE)

#Providing a mask entirely removes regions of the image for segmentation:
mask=matrix(0,dim(image)[1],dim(image)[2])
mask[1:80,]=1
profoundMakeSegim(image, mask=mask, plot=TRUE)

#Providing a previously created object map can sometimes help with detection (not here):
profoundMakeSegim(image, mask=mask, object=segim$objects, plot=TRUE)

## End(Not run)
```

profoundMakeSegimExpand *Segmentation Map Expansion and Dilation*

Description

A high level utility to achieve decent quality image segmentation based on the expansion of a pre-existing segmentation map. It uses smoothing and local flux weighted comparisons to grow the current segmentation map so as to better identify distinct objects for use in, e.g., `profitSetupData`.

Usage

```
profoundMakeSegimExpand(image = NULL, segim = NULL, mask = NULL, objects = NULL,
skycut = 1, SBlim = NULL, magzero = 0, gain = NULL, pixscale = 1, sigma = 1,
smooth = TRUE, expandsigma = 5, expand = "all", sky = NULL, skyRMS = NULL, header = NULL,
verbose = FALSE, plot = FALSE, stats = TRUE, rotstats = FALSE, boundstats = FALSE,
offset = 1, sortcol = "segID", decreasing = FALSE, ...)
profoundMakeSegimDilate(image = NULL, segim = NULL, mask = NULL, size = 9, shape = "disc",
expand = "all", magzero = 0, gain = NULL, pixscale = 1, sky = 0, skyRMS = 0,
header = NULL, verbose = FALSE, plot = FALSE, stats = TRUE, rotstats = FALSE,
boundstats = FALSE, offset = 1, sortcol = "segID", decreasing = FALSE, ...)
```

Arguments

`image`	Numeric matrix; required, the image we want to analyse. Note, image NAs are treated as masked pixels.
`segim`	Integer matrix; required, the segmentation map of the image. This matrix *must* be the same dimensions as 'image'.
`mask`	Boolean matrix; optional, parts of the image to mask out (i.e. ignore), where 1 means mask out and 0 means use for analysis. If provided, this matrix *must* be the same dimensions as 'image'.
`objects`	Boolean matrix; optional, object mask where 1 is object and 0 is sky. If provided, this matrix *must* be the same dimensions as 'image'.
`skycut`	Numeric scalar; the lowest threshold to make on the 'image' in units of the skyRMS. Since we are restricted to expanding out pre-existing segmentation regions we can usually afford to make this value lower than the equivalent in `profoundMakeSegim`.
`SBlim`	Numeric scalar; the magnitude/arcsec^2 surface brightness threshold to apply. This is always used in conjunction with 'skycut', so set 'skycut' to be very large (e.g. Inf) if you want a pure surface brightness threshold for the segmentation. 'magzero' and 'pixscale' must also be present for this to be used.
`magzero`	Numeric scalar; the magnitude zero point. What this implies depends on the magnitude system being used (e.g. AB or Vega). If provided along with 'pixscale' then the flux and surface brightness outputs will represent magnitudes and mag/asec^2.

gain
: Numeric scalar; the gain (in photo-electrons per ADU). This is only used to compute object shot-noise component of the flux error (else this is set to 0).

pixscale
: Numeric scalar; the pixel scale, where pixscale=asec/pix (e.g. 0.4 for SDSS). If set to 1 (default), then the output is in terms of pixels, otherwise it is in arcseconds. If provided along with 'magzero' then the flux and surface brightness outputs will represent magnitudes and mag/asec^2.

sigma
: Numeric scalar; standard deviation of the blur used when 'smooth'=TRUE.

smooth
: Logical; should smoothing be done on the target 'image'? If present, this will use the imblur function from the imager package. Otherwise it will use the gblur function from the EBImage package with a warning. These functions are very similar in output, but not strictly identical.

expandsigma
: Numeric scalar; standard deviation of the blur used when expanding out the 'segim'. Roughly speaking if 'skycut' is set to a low number (say -5) then the expansion will not be prevented by the local sky level and it will grow by the number of pixels specified by 'expandsigma'.

expand
: Integer vector; specifies which segmentation regions should be expanded by the segID integer reference. If left with the default 'expand'='all' then all segments will be expanded.

size
: Integer scalar; the size (e.g. width/diameter) of the dilation kernel in pixels. Should be an odd number else will be rounded up to the nearest odd number. See makeBrush.

shape
: Character scalar; the shape of the dilation kernel. See makeBrush.

sky
: User provided estimate of the absolute sky level. Can be a scalar or a matrix matching the dimensions of 'image' (allows values to vary per pixel). This will be subtracted off the 'image' internally, so only provide this if the sky does need to be subtracted!

skyRMS
: User provided estimate of the RMS of the sky. Can be a scalar or a matrix matching the dimensions of 'image' (allows values to vary per pixel).

header
: Full FITS header in table or vector format. If this is provided then the segmentations statistics table will gain 'RAcen' and 'Decen' coordinate outputs. Legal table format headers are provided by the read.fitshdr function or the 'hdr' list output of read.fits in the astro package; the 'hdr' output of readFITS in the FITSio package or the 'header' output of magcutoutWCS. Missing header keywords are printed out and other header option arguments are used in these cases. See magWCSxy2radec.

verbose
: Logical; should verbose output be displayed to the user? Since big image can take a long time to run, you might want to monitor progress.

plot
: Logical; should a diagnostic plot be generated? This is useful when you only have a small number of sources (roughly a few hundred). With more than this it can start to take a long time to make the plot!

stats
: Logical; should statistics on the segmented objects be returned?

rotstats
: Logical; if TRUE then the 'asymm', 'flux_reflect' and 'mag_reflect' are computed, else they are set to NA. This is because they are very expensive to compute compared to other photometric properties.

boundstats	Logical; if TRUE then various pixel boundary statistics are computed ('Nedge', 'Nsky', 'Nobject', 'Nborder', 'edge_frac', 'edge_excess' and 'FlagBorder'). If FALSE these return NA instead (saving computation time).
offset	Integer scalar; the distance to offset when searching for nearby segments (used in profoundSegimStats).
sortcol	Character; name of the output column that the returned segmentation statistics data.frame should be sorted by (the default is segID, i.e. segment order). See below for column names and contents.
decreasing	Logical; if FALSE (default) the segmentation statistics data.frame will be sorted in increasing order, if TRUE the data.frame will be sorted in decreasing order.
...	Further arguments to be passed to magimage. Only relevant is 'plot'=TRUE.

Details

The basic behaviour of profoundMakeSegimExpand and profoundMakeSegimDilate is to intelligently expand out image segments already identified by, e.g., profoundMakeSegim.

The profoundMakeSegimExpand defaults should work reasonably well on modern survey data (see Examples), but should the solution not be ideal try modifying these parameters (in order of impact priority): 'skycut', 'dim', 'expandsigma', 'sigma'.

profoundMakeSegimDilate is similar in nature to the pixel growing objmask routine in IRAF (see the 'ngrow' and 'agrow' description at http://stsdas.stsci.edu/cgi-bin/gethelp.cgi?objmasks). This similarity was discovered after implementation, but it is worth noting that the higher level curve of growth function profoundProFound is not trivially replicated by other astronomy tools.

The main difference between profoundMakeSegimExpand and profoundMakeSegimDilate is the former grows the expansion a bit more organically, whereas the latter always gives new pixels to the brighter object if in doubt. That said, profoundMakeSegimDilate often gives very similar solutions and runs about 10+ times faster, so might be the only option for larger images.

Value

A list containing:

segim	Integer matrix; the segmentation map matched pixel by pixel to 'image'.
objects	Logical matrix; the object map matched pixel by pixel to 'image'. 1 means there is an object at this pixel, 0 means it is a sky pixel. Can be used as a mask in various other functions that require objects to be masked out.
sky	The estimated sky level of the 'image'. profoundMakeSegimExpand only).
skyRMS	The estimated sky RMS of the 'image'. profoundMakeSegimExpand only).
segstats	If 'stats'=TRUE this is a data.frame (see below), otherwise NULL.
header	The header provided, if missing this is NULL.
SBlim	The surface brightness limit of detected objects. Requires at least 'magzero' to be provided and 'skycut'>0, else NULL. profoundMakeSegimExpand only.
call	The original function call.

If 'stats'=TRUE then the function profoundSegimStats is called and the 'segstats' part of the returned list will contain a data.frame with columns (else NULL):

segID	Segmentation ID, which can be matched against values in 'segim'
uniqueID	Unique ID, which is fairly static and based on the xmax and ymax position
xcen	Flux weighted x centre
ycen	Flux weighted y centre
xmax	x position of maximum flux
ymax	y position of maximum flux
RAcen	Flux weighted degrees Right Ascension centre (only present if a 'header' is provided)
Deccen	Flux weighted degrees Declination centre (only present if a 'header' is provided)
RAmax	Right Ascension of maximum flux (only present if a 'header' is provided)
Decmax	Declination of maximum flux (only present if a 'header' is provided)
sep	Radial offset between the cen and max definition of the centre (units of 'pixscale', so if 'pixscale' represents the standard asec/pix this will be asec)
flux	Total flux (calculated using 'image'-'sky') in ADUs
mag	Total flux converted to mag using 'magzero'
cenfrac	Fraction of flux in the brightest pixel
N50	Number of brightest pixels containing 50% of the flux
N90	Number of brightest pixels containing 90% of the flux
N100	Total number of pixels in this segment, i.e. contains 100% of the flux
R50	Approximate elliptical semi-major axis containing 50% of the flux (units of 'pixscale', so if 'pixscale' represents the standard asec/pix this will be asec)
R90	Approximate elliptical semi-major axis containing 90% of the flux (units of 'pixscale', so if 'pixscale' represents the standard asec/pix this will be asec)
R100	Approximate elliptical semi-major axis containing 100% of the flux (units of 'pixscale', so if 'pixscale' represents the standard asec/pix this will be asec)
SB_N50	Mean surface brightness containing brightest 50% of the flux, calculated as 'flux'*0.5/'N50' (if 'pixscale' has been set correctly then this column will represent mag/asec^2. Otherwise it will be mag/pix^2)
SB_N90	Mean surface brightness containing brightest 90% of the flux, calculated as 'flux'*0.9/'N90' (if 'pixscale' has been set correctly then this column will represent mag/asec^2. Otherwise it will be mag/pix^2)
SB_N100	Mean surface brightness containing all of the flux, calculated as 'flux'/'N100' (if 'pixscale' has been set correctly then this column will represent mag/asec^2. Otherwise it will be mag/pix^2)
xsd	Weighted standard deviation in x (always in units of pix)
ysd	Weighted standard deviation in y (always in units of pix)
covxy	Weighted covariance in xy (always in units of pix)

corxy
: Weighted correlation in xy (always in units of pix)

con
: Concentration, 'R50'/'R90'

asymm
: 180 degree flux asymmetry (0-1, where 0 is perfect symmetry and 1 complete asymmetry)

flux_reflect
: Flux corrected for asymmetry by doubling the contribution of flux for asymmetric pixels (defined as no matching segment pixel found when the segment is rotated through 180 degrees)

mag_reflect
: 'flux_reflect' converted to mag using 'magzero'

semimaj
: Weighted standard deviation along the major axis, i.e. the semi-major first moment, so ~2 times this would be a typical major axis Kron radius (always in units of pix)

semimin
: Weighted standard deviation along the minor axis, i.e. the semi-minor first moment, so ~2 times this would be a typical minor axis Kron radius (always in units of pix)

axrat
: Axial ratio as given by min/maj

ang
: Orientation of the semi-major axis in degrees. This has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical)

signif
: Approximate singificance of the detection using the Chi-Square distribution

FPlim
: Approximate false-positive significance limit below which one such source might appear spuriously on an image this large

flux_err
: Estimated total error in the flux for the segment

mag_err
: Estimated total error in the magnitude for the segment

flux_err_sky
: Sky subtraction component of the flux error

flux_err_skyRMS
: Sky RMS component of the flux error

flux_err_shot
: Object shot-noise component of the flux error (only if 'gain' is provided)

sky_mean
: Mean flux of the sky over all segment pixels

sky_sum
: Total flux of the sky over all segment pixels

skyRMS_mean
: Mean value of the sky RMS over all segment pixels

Nedge
: Number of edge segment pixels that make up the outer edge of the segment

Nsky
: Number of edge segment pixels that are touching sky

Nobject
: Number of edge segment pixels that are touching another object segment

Nborder
: Number of edge segment pixels that are touching the 'image' border

Nmask
: Number of edge segment pixels that are touching a masked pixel (note NAs in 'image' are also treated as masked pixels)

edge_frac
: Fraction of edge segment pixels that are touching the sky i.e. 'Nsky''Nedge', higher generally meaning more robust segmentation statistics

edge_excess
: Ratio of the number of edge pixels to the expected number given the elliptical geometry measurements of the segment. If this is larger than 1 then it is a sign that the segment geometry is irregular, and is likely a flag for compromised photometry

flag_border — A binary flag telling the user which 'image' borders the segment touches. The bottom of the 'image' is flagged 1, left=2, top=4 and right=8. A summed combination of these flags indicate the segment is in a corner touching two borders: bottom-left=3, top-left=6, top-right=12, bottom-right=9.

Author(s)

Aaron Robotham

See Also

profoundMakeSegim, profoundProFound, profoundSegimStats, profoundSegimPlot

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits',
package="ProFound"))$imDat
segim=profoundMakeSegim(image, plot=TRUE, skycut=2)
profoundMakeSegimExpand(image, segim$segim, plot=TRUE, skycut=1)
profoundMakeSegimDilate(image, segim$segim, plot=TRUE)

#Some other examples:

profoundMakeSegimExpand(image, segim$segim, plot=TRUE, skycut=0)
profoundMakeSegimExpand(image, segim$segim, plot=TRUE, skycut=-Inf, sigma=3)

profoundMakeSegimDilate(image, segim$segim, plot=TRUE, size = 15)
profoundMakeSegimDilate(image, segim$segim, plot=TRUE, size = 21)

#This expansion process is a *much* better idea then simply setting the original skycut
#to a low value like 1/0:
profoundMakeSegim(image, plot=TRUE, skycut = 1)
profoundMakeSegim(image, plot=TRUE, skycut = 0)

## End(Not run)
```

---

profoundMakeSegimPropagate

Propagate Identified Segments

Description

Propagates all identified segments across the full image, only ignoring masked regions. This serves to identify which segment every pixel is most likely to belong to using a number of image related criteria. Uses EBImage's propagate function to do the grunt work.

Usage

```
profoundMakeSegimPropagate(image = NULL, segim = NULL, objects = NULL, mask = NULL,
sky = 0, lambda = 1e-04, plot = FALSE, ...)
```

Arguments

image	Numeric matrix; required, the image we want to analyse. Note, image NAs are treated as masked pixels.
segim	Integer matrix; required, the segmentation map of the image. This matrix *must* be the same dimensions as 'image'.
objects	Boolean matrix; optional, object mask where 1 is object and 0 is sky. If provided, this matrix *must* be the same dimensions as 'image'.
mask	Boolean matrix; optional, parts of the image to mask out (i.e. ignore), where 1 means mask out and 0 means use for analysis. If provided, this matrix *must* be the same dimensions as 'image'.
sky	User provided estimate of the absolute sky level. Can be a scalar or a matrix matching the dimensions of 'image' (allows values to vary per pixel). This will be subtracted off the 'image' internally, so only provide this if the sky does need to be subtracted!
lambda	A numeric value. The regularization parameter used in the metric, determining the trade-off between the Euclidean distance in the image plane and the contribution of the gradient of x. See Details.
plot	Logical; should a diagnostic plot be generated? This is useful when you only have a small number of sources (roughly a few hundred). With more than this it can start to take a long time to make the plot!
...	Further arguments to be passed to magimage. Only relevant is 'plot'=TRUE.

Details

This function propgates out the identified segments into the rest of the 'image', only region identified in the 'mask' will not be assigned to a segment. To assign pixels a mixture of the Euclidian distance and the local gradient is used (as described below). The purpose of this routine is to identify all pixels in the image with their most likely segment (whether nominally object or sky pixel). The true sky pixels identified as belonging to a segment should also provide the best possible local estimate of the sky level.

For internal completeness, the below description is taken almost verbatim from the EBImage propagate function.

The method operates by computing a discretized approximation of the Voronoi regions for given seed points on a Riemann manifold with a metric controlled by local 'image' features.

Under this metric, the infinitesimal distance d between points v and v+dv is defined by:

d^2 = ((t(dv)*g)^2 + lambda*t(dv)*dv)/(lambda + 1) , where g is the gradient of 'image' x at point v.

'lambda' controls the weight of the Euclidean distance term. When 'lambda' tends to infinity, d tends to the Euclidean distance. When 'lambda' tends to 0, d tends to the intensity gradient of the 'image'.

The gradient is computed on a neighborhood of 3x3 pixels.

Segmentation of the Voronoi regions in the vicinity of flat areas (having a null gradient) with small values of 'lambda' can suffer from artifacts coming from the metric approximation.

Value

A list containing two images:

propim	The propagated segmentation map including the original segments identified.
propim_sky	The propagated segmentation map removing the original segments identified (these pixels are set to 0).

Author(s)

Aaron Robotham

See Also

profoundProFound propagate

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits', package="ProFound"))

profound=profoundProFound(image, skycut=1.5, magzero=30, verbose=TRUE, plot=TRUE)

tempprop=profoundMakeSegimPropagate(image$imDat, segim=profound$segim, plot=TRUE)

tempprop_stats=profoundSegimStats(image$imDat, segim=tempprop$propim_sky,
sky=profound$sky, skyRMS=profound$skyRMS)

magplot(profound$segstats$mag, tempprop_stats$flux/tempprop_stats$N100, grid=TRUE)

#You can stop the propogation using a mask:

mask=array(0, dim=dim(image$imDat))
mask[1:50,]=1

profoundMakeSegimPropagate(image$imDat, segim=profound$segim, plot=TRUE, mask=mask)

## End(Not run)
```

profoundMakeSigma *Make a Sigma Map*

Description

A utility function to construct a ProFit legal sigma map that can be input to profitSetupData.

Usage

```
profoundMakeSigma(image = NULL, objects = NULL, sky = 0, skyRMS = 0, readRMS = 0,
darkRMS = 0, skycut = 0, gain = 1, image_units = 'ADU', sky_units = 'ADU',
read_units = 'ADU', dark_units = 'ADU', output_units = 'ADU', plot = FALSE, ...)
```

Arguments

image	Numeric matrix; required, the image we want to analyse.
objects	Boolean matrix; optional, object mask where 1 is object and 0 is sky. Pixels set to 0 are interpreted as sky, and set to zero for calculating object shot-noise. If provided, this matrix *must* be the same dimensions as 'image'.
sky	Numeric; the absolute sky level. Consider using the sky output from profoundSkyEst or profoundMakeSkyGrid. Can be a scalar (value uniformly applied to full 'sigma' map) or a matrix matching the dimensions of 'image' (allows values to vary per pixel). This will be subtracted off the 'image' internally, so only provide this if the sky does need to be subtracted!
skyRMS	Numeric; the RMS of the sky. Consider using the skyRMS output from profoundSkyEst or profoundMakeSkyGrid. Can be a scalar (value uniformly applied to full 'sigma' map) or a matrix matching the dimensions of 'image' (allows values to vary per pixel).
readRMS	Numeric; the RMS of the read-noise. If you have estimated the sky RMS from the image directly this should not be necessary since it naturally captures this component. Can be a scalar (value uniformly applied to full 'sigma' map) or a matrix matching the dimensions of 'image' (allows values to vary per pixel).
darkRMS	Numeric; the RMS of the dark-current-noise. If you have estimated the sky RMS from the image directly this should not be necessary since it naturally captures this component. Can be a scalar (value uniformly applied to full 'sigma' map) or a matrix matching the dimensions of 'image' (allows values to vary per pixel).
skycut	How many multiples of 'skyRMS' above the 'sky' to start calculating shot-noise based on the 'gain' scaling of the 'image'. If you are missing an object mask You almost certainly do not want this to be below 0 (else you will reduce the level of the sigma map just due to fluctuations in the sky), and in practice this should probably be set in the range 1-3.
gain	Numeric; the gain (in photo-electrons per ADU). For a very rough estimate consider using the gain output from profoundGainEst. Can be a scalar (value uniformly applied to full 'sigma' map) or a matrix matching the dimensions of 'image' (allows values to vary per pixel).

image_units	Character; the units of the 'image'. Must either be 'ADU' for generic astronomical data units, or 'elec' for photo-electrons.
sky_units	Character; the units of 'sky' and 'skyRMS'. Must either be 'ADU' for generic astronomical data units (the same type and scaling as per 'image'), or 'elec' for photo-electrons.
read_units	Character; the units of 'read'. Must either be 'ADU' for generic astronomical data units (the same type and scaling as per 'image'), or 'elec' for photo-electrons.
dark_units	Character; the units of 'dark'. Must either be 'ADU' for generic astronomical data units (the same type and scaling as per 'image'), or 'elec' for photo-electrons.
output_units	Character; the units of the output sigma map. Must either be 'ADU' for generic astronomical data units (the same type and scaling as per 'image'), or 'elec' for photo-electrons.
plot	Logical; should a magimage plot of the output be generated?
...	Further arguments to be passed to magimage. Only relevant is 'plot'=TRUE.

Details

This is a simple utility function, but useful for beginners if they are unsure of how the error terms should be propagated (in short: in quadrature).

Value

Numeric matrix; a sigma map the same size as 'image'. This should be appropriate for feeding into profitSetupData.

Author(s)

Aaron Robotham

See Also

profoundSkyEst, profoundGainEst

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits',
package="ProFound"))
profound=profoundProFound(image)

sigma_est=profoundMakeSigma(image$imDat, objects=profound$objects, sky=profound$sky,
skyRMS=profound$skyRMS)

## End(Not run)
```

profoundMakeSkyMap *Calculate Sky Maps*

Description

The high level function computes the absolute sky and sky RMS level over an image at a scale defined locally by the 'box' parameter. This coarse map can then be used to compute sky/skyRMS values for the local sky anywhere on an image. This function uses profoundSkyEstLoc to calculate the sky statistics for the subset boxcar regions.

Usage

```
profoundMakeSkyMap(image = NULL, objects = NULL, mask = NULL, box = c(100,100),
grid = box, skytype = "median", skyRMStype = "quanlo", sigmasel = 1,
skypixmin = prod(box)/2, boxadd = box/2, boxiters = 0, doclip = TRUE, shiftloc = FALSE,
paddim = TRUE, cores = 1)
profoundMakeSkyGrid(image = NULL, objects = NULL, mask = NULL, box = c(100,100),
grid = box, type = 'bicubic', skytype = "median", skyRMStype = "quanlo", sigmasel = 1,
skypixmin = prod(box)/2, boxadd = box/2, boxiters = 0, doclip = TRUE, shiftloc = FALSE,
paddim = TRUE, cores = 1)
```

Arguments

image	Numeric matrix; required, the image we want to analyse.
objects	Boolean matrix; optional, object mask where 1 is object and 0 is sky. If provided, this matrix *must* be the same dimensions as 'image'.
mask	Boolean matrix; optional, parts of the image to mask out (i.e. ignore), where 1 means mask out and 0 means use for analysis. If provided, this matrix *must* be the same dimensions as 'image'.
box	Integer vector; the dimensions of the box car filter to estimate the sky with.
grid	Integer vector; the resolution of the background grid to estimate the sky with. By default this is set to be the same as the 'box'.
type	Character scalar; either "bilinear" for bilinear interpolation or "bicubic" for bicubic interpolation (default). The former creates sharper edges.
skytype	Character scalar; the type of sky level estimator used. Allowed options are 'median' (the default), 'mean' and 'mode' (see profoundSkyEstLoc for an explanation of what these estimators do). In all cases this is the estimator applied to unmasked and non-object pixels. If 'doclip'=TRUE then the pixels will be dynamically sigma clipped before the estimator is run.
skyRMStype	Character scalar; the type of sky level estimator used. Allowed options are 'quanlo' (the default), 'quanhi', 'quanboth', and 'sd' (see profoundSkyEstLoc for an explanation of what these estimators do). In all cases this is the estimator applied to unmasked and non-object pixels. If 'doclip'=TRUE then the pixels will be dynamically sigma clipped before the estimator is run.

`sigmasel` Numeric scalar; the quantile to use when trying to estimate the true standard-deviation of the sky distribution. If contamination is low then the default of 1 is about optimal in terms of S/N, but you might need to make the value lower when contamination is very high.

`skypixmin` Numeric scalar; the minimum number of sky pixels desired in our cutout. The default is that we need half the original number of pixels in the 'box' to be sky.

`boxadd` Integer vector; the dimensions to add to the 'box' to capture more pixels if 'skypixmin' has not been achieved.

`boxiters` Integer scalar; the number of 'box'+'boxadd' iterations to attempt in order to capture 'skypixmin' sky pixels. The default means the box will not be grown at all.

`doclip` Logical; should the unmasked non-object pixels used to estimate to local sky value be further sigma-clipped using magclip? Whether this is used or not is a product of the quality of the objects extraction. If all detectable objects really have been found and the dilated objects mask leaves only apparent sky pixels then an advanced user might be confident enough to set this to FALSE. If an doubt, leave as TRUE.

`shiftloc` Logical; should the cutout centre for the sky shift from 'loc' if the desired 'box' size extends beyond the edge of the image? (See magcutout for details).

`paddim` Logical; should the cutout be padded with image data until it meets the desired 'box' size (if 'shiftloc' is true) or padded with NAs for data outside the image boundary otherwise? (See magcutout for details).

`cores` Integer scalar; how many cores should be used to calculate sky properties of the image. Given the overhead for parallel computing, this should probably only be above 1 for larger images.

Details

The matrix generated will have many fewer pixels than the original 'image', so it will need to be interpolated back onto the full grid by some mechanism in order to have 1-1 values for the sky and sky RMS.

Value

`profoundMakeSkyMap` produces a list of two lists. The first (called sky) contains a list of x,y,z values for the absolute sky, and second (called skyRMS) contains a list of x,y,z values for the sky RMS. The grids returned are as coarse as the 'box' option provided.

`profoundMakeSkyGrid` produces a list of two lists. The first (called sky) is a matrix of values for the absolute sky. The second (called skyRMS) is a matrix of values for the absolute sky RMS. The image matrices returned are pixel matched to the input 'image' using the specified interpolation scheme.

Author(s)

Aaron Robotham

See Also

profoundSkyEst, profoundSkyEstLoc

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits',
package="ProFound"))$imDat
magimage(image)
skymap = profoundMakeSkyMap(image, box=c(89,89))
magimage(skymap$sky)
magimage(skymap$skyRMS)

# Now again, masking out the known objects (will not help too much in this case):

segim=profoundMakeSegim(image, skycut=1.5, plot=TRUE)
segim_ex=profoundMakeSegimExpand(image, segim$segim, skycut=-Inf, plot=TRUE)

skymap=profoundMakeSkyMap(image, objects=segim_ex$objects, box=c(89,89))
magimage(skymap$sky, magmap=FALSE)
magimage(skymap$skyRMS, magmap=FALSE)

# We can bilinear interpolate this onto the full image grid:

skybil = profoundMakeSkyGrid(image, objects=segim_ex$objects, box=c(89,89),
type='bilinear')
magimage(skybil$sky, magmap=FALSE)
magimage(skybil$skyRMS, magmap=FALSE)

# Or we can bicubic interpolate this onto the full image grid:

skybic = profoundMakeSkyGrid(image, objects=segim_ex$objects, box=c(89,89), type='bicubic')
magimage(skybic$sky, magmap=FALSE)
magimage(skybic$skyRMS, magmap=FALSE)

# The differences tend to be at the edges:

magimage(skybil$sky-skybic$sky, magmap=FALSE)
magimage(skybil$skyRMS-skybic$skyRMS, magmap=FALSE)

## End(Not run)
```

---

profoundMakeStack *Stack Images*

Description

Stacks multiple images based on their signal-to-noise.

Usage

```
profoundMakeStack(image_list = NULL, sky_list = NULL, skyRMS_list = NULL, magzero_in = 0,
magzero_out = 0)
```

Arguments

`image_list`	List; each list element is a numeric matrix representing the image to be stacked.
`sky_list`	List; each list element is a numeric matrix representing the sky to be subtracted.
`skyRMS_list`	List; each list element is a numeric matrix representing the sky-RMS to weight the stack with.
`magzero_in`	Numeric vector; the input mag-zero points. If length 1 then it is assumed all input frames have the same mag-zero point.
`magzero_out`	Numeric scalar; the output mag-zero point desired.

Details

The stack is actually done based on variance weighting. In pseudo code:

stack=0 stackRMS=0 for(i in 1:length(image_list)) stack=stack+(image_list[[i]]-sky_list[[i]])/(skyRMS_list[[i]]^2) sky_stack=sky_stack+(image_list[[i]]^2)

stack=stack*sky_stack/(length(skyRMS_list)^2)

The output is explictly sky subtracted (so the sky is now 0 everywhere by definition as far as `profoundProFound` is concerned). The stacked sky is not returned. However, it can be computed by running `profoundMakeStack` again, but passing the sky list originally passed to the '`sky_list`' argument to the '`image_list`' argument instead, and not providing any input to the '`sky_list`' argument (or setting this to 0).

Value

A list containing:

`image`	Numeric matrix; the variance-weighted sky-subtracted stacked image.
`skyRMS`	Numeric matrix/scalar; the sky RMS image/value of the final stacked image
`magzero`	The mag-zero point of the stacked image.

Author(s)

Aaron Robotham

See Also

`profoundProFound`

Examples

```
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits', package="ProFound"))

stack=profoundMakeStack(list(image$imDat, image$imDat, image$imDat),
skyRMS_list = list(8,8,3))

#The new signal-to-noise weighted sky should equal sqrt(1/(1/8^2+1/8^2+1/3^2)) = 2.65

stack$skyRMS
```

---

profoundMultiBand *Multi Band ProFound Photometry*

Description

Run multiband ProFound photometry either with loaded data, or images on a local disk.

Usage

```
profoundMultiBand(inputlist = NULL, dir = "", segim = NULL, mask = NULL, iters_det = 6,
iters_tot = 0, detectbands = "r", multibands = c("u", "g", "r", "i", "z"), magzero = 0,
gain = NULL, bandappend = multibands, totappend = "t", colappend = "c", grpappend = 'g',
dotot = TRUE, docol = TRUE, dogrp = TRUE, deblend = FALSE, groupstats = FALSE, ...)
```

Arguments

inputlist — A list of already loaded images. Typically of the type loaded in from FITS files by the astro package's read.fits function, or the FITSio package's readFITS function. If using the 'inputlist' parameter the length of the list must be the same length as 'multibands' (and the related parameters).

dir — If 'inputlist' is left as NULL then profoundMultiBand will instead try to load in FITS images from the directory specified by 'dir'. The images in the directory must have names like 'multibands'[i].fits etc (so with the defaults names like u.fits and g.fits would be okay). Since 'multibands' effectively specifies the file names much more complicated naming can be used and passed in, but it is also used by default for naming the catalogue column outputs, so shorter names/references are likely to be preferable there (i.e. mag_ut is simpler than mag_KiDS_VST_ut etc). This can be over-ridden by using 'bandappend'.

segim — Integer matrix; a specified segmentation map of the image. This matrix *must* be the same dimensions as the detection image/s if supplied. If this option is used then profoundMultiBand will not compute its initial segmentation map using profoundMakeSegim, which is then dilated. Instead it will use the one passed through 'segim' and dilate this according to the 'iters_det' argument (so set this to 0 if you want the 'segim' to be used as is).

mask
: Boolean matrix or integer scalar; optional, parts of the image to mask out (i.e. ignore). If a matrix is provided, this matrix *must* be the same dimensions as 'image' where 1 means mask out and 0 means use for analysis. if a scalar is provided it indicates the exact 'image' values that should be treated as masked (e.g. by setting masked pixels to 0 or -999). The latter achieves the same effect as setting masked 'image' pixels to NA, but allows for the fact not all programs can produce R legal NA values.

iters_det
: Integer scalar; the maximum number of curve of growth dilations that should be made to the detection image. This needs to be large enough to capture all the flux for sources of interest, but increasing this will increase the computation time for profoundProFound. If this is set to 0 then the undilated 'segim' image, whether provided or computed internally via profoundMakeSegim, will be used instead.

iters_tot
: Integer vector; the maximum number of curve of additional growth dilations that should be made above the dilated detection segmentation map for multi band total colour photometry. This is only relevant if 'dotot'=TRUE. This should not be set too high (and might even be 0, the default) since the detection image should generally be fairly deep. 'iters_tot' must either be length 1 (in which this value is used for all bands), or the same length and order as 'multibands'.

detectbands
: Character vector; the names of the detection bands that will be stacked using profoundMakeStack and then analysed with the provided settings with profoundProFound to make a reference segmentation map for further multi band photometry. These bands must be present in 'multibands'. Can be a scalar (i.e. a single band is used). If set to 'get' then it will use all legal FITS files in the target directory. If set to 'all' then it will use all 'multibands' inputs.

multibands
: Character vector; the names of the target multi band photometry images. If set to 'get' then it will use all legal FITS files in the target directory. If using the 'inputlist' parameter the length of the list must be the same length as 'multibands'. 'magzero' must either be length 1 (in which this value is used for all bands), or the same length and order as 'multibands'. If specified, 'gain' must either be length 1 (in which this value is used for all bands), or the same length and order as 'multibands'. If specified, 'catappend' must either the same length and order as 'multibands'.

magzero
: Numeric vector; the magnitude zero point of the images being used. 'magzero' must either be length 1 (in which this value is used for all bands), or the same length and order as 'multibands'. See also profoundProFound.

gain
: Numeric vector; the gain of the images being used. 'gain' must either be length 1 (in which this value is used for all bands), or the same length and order as 'multibands'. See also profoundProFound.

bandappend
: Character vector; characters to be appended per band in the output multi band photometry catalogues. The default will create columns with names like mag_ut (total) and mag_uc (colour).

totappend
: Character scalar; character to be appended in the output multi band total photometry catalogue (cat_tot). The default will create columns with names like mag_ut and R50_ut.

colappend	Character scalar; character to be appended in the output multi band colour photometry catalogue (cat_col). The default will create column with names like mag_uc and R50_uc.
grpappend	Character scalar; character to be appended in the grouped segment multi band total photometry catalogue (cat_tot). The default will create columns with names like mag_ug and R50_ug.
dotot	Logical; should dilated segment total photometry be computed for the bands specified in 'multibands'? This will return closer to total magnitudes in all target bands.
docol	Logical; should non-dilated segment colour photometry be computed for the bands specified in 'multibands'? This will return better colour magnitudes in all target bands (i.e. more accurate differences between bands) and will typically under-represent the total photometry.
dogrp	Logical; should group segment photometry be computed for the bands specified in 'multibands'? This might be useful for re-assembling large galaxies that are broken up at a later date. 'boundstats' must also be set to TRUE if 'dogrp'=TRUE is set.
deblend	Logical; should total segment flux be deblended using profoundFluxDeblend and these columns appended to the end of the output segstats? This only applies to the 'cat_tot' output.
groupstats	Logical; if TRUE then the IDs of grouped segments is calculated for the detection image via profoundSegimGroup and output to the returned object 'group'. By default this option is linked to 'boundstats', i.e. it is assumed if you want boundary statistics then you probably also want grouped object IDs returned.
...	Further arguments to be passed to all instances of profoundProFound. E.g. if the sky 'box' is set to a non-default value (default is 'box'=c(100,100)), this will be propogated to all of the multi band photometry runs of profoundProFound.

Details

This very high level function simplifies a sequence of function calls that we found users typically needed to make, but when scripted they were prone to mistakes and made multi band photometry scripts hard to maintain.

In the simplest sense this script runs profoundProFound on each detection band and uses this information to make a stacked image using profoundMakeStack. profoundProFound is then run on this stacked image to make a deep segmentation map. For good total photometry the segim object from this output is used, and allowed to further dilate to account for different observing conditions (i.e. PSFs). For good colour photometry the segim_orig object from this output is used. Only the profoundSegimStats output is kept for the target multi band images, so not all of the outputs from profoundProFound since this is usually unnecessary when operating in this mode, and creates a huge quantity of data.

Value

An object list of class 'profoundmulti' containing:

pro_detect	The full output of profoundProFound for the detection image (of class 'profound').
cat_tot	If 'dotot'=TRUE, the dilated total photometry for the target bands. Effectively the output of profoundSegimStats run on pro_detect$segim.
cat_col	If 'docol'=TRUE, the non-dilated colour photometry for the target bands. Effectively the output of profoundSegimStats run on pro_detect$segim_orig.
cat_grp	If 'dogrp'=TRUE, the group segment photometry for the target bands. Effectively the output of profoundSegimStats run on pro_detect$group$groupim.
detectbands	Character vector; the names of the detection bands used.
multibands	Character vector; the names of the target multi band photometry images used.
call	The original function call.
date	The date, more specifically the output of date.
time	The elapsed run time in seconds.

Author(s)

Aaron Robotham

References

Robotham A.S.G., et al., 2018, MNRAS, 476, 3137

See Also

profoundProFound

Examples

```
## Not run:
# Load images
GALEX_NUV=readFITS(system.file("extdata", 'GALEX_NUV.fits', package="magicaxis"))
VST_r=readFITS(system.file("extdata", 'VST_r.fits', package="magicaxis"))
VISTA_K=readFITS(system.file("extdata", 'VISTA_K.fits', package="magicaxis"))

# Warp to common WCS:
GALEX_NUV_VST=magwarp(GALEX_NUV, VST_r$hdr)
VISTA_K_VST=magwarp(VISTA_K, VST_r$hdr)

# Run profoundMultiBand on defaults:
multi=profoundMultiBand(inputlist=list(GALEX_NUV_VST, VST_r, VISTA_K_VST),
magzero=c(20.08,0,30), detectbands='r', multibands=c('NUV','r','K'))

# Notice the blue halo around the central sources:
plot(multi$pro_detect)

# Run profoundMultiBand with boxiters=2 (to avoid over-subtracting the sky):
multi=profoundMultiBand(inputlist=list(GALEX_NUV_VST, VST_r, VISTA_K_VST),
magzero=c(20.08,0,30), detectbands='r', multibands=c('NUV','r','K'), boxiters = 2)
```

```
# Looks better now:
plot(multi$pro_detect)

magplot(multi$cat_tot$mag_rt, multi$cat_col$mag_NUVc-multi$cat_col$mag_rc, ylim=c(-2,10))
points(multi$cat_tot$mag_rt, multi$cat_col$mag_rc-multi$cat_col$mag_Kc, col='red')

# Some options on passing segim:

multi2=profoundMultiBand(segim=multi$pro_detect$segim, inputlist=list(GALEX_NUV_VST,
VST_r, VISTA_K_VST), magzero=c(20.08,0,30), detectbands='r', multibands=c('NUV','r','K'),
iters_det = 0, boxiters=2)

multi3=profoundMultiBand(segim=multi$pro_detect$segim_orig, inputlist=list(GALEX_NUV_VST,
VST_r, VISTA_K_VST), magzero=c(20.08,0,30), detectbands='r', multibands=c('NUV','r','K'),
iters_det = 6, boxiters=2)

# multi and multi3 should create identical plots (since we are dilating the original
# segim_orig in the same manner), but multi2 will just be the final dilated segim without
# any dilations, hence the top-right is all green (segim=segim_orig). The final fluxes
# should be the same though for all 3 runs (left-middle, bottom-centre and bottom-right).

plot(multi$pro_detect)
plot(multi2$pro_detect)
plot(multi3$pro_detect)

## End(Not run)
```

---

profoundPixelCorrelation — *Pixel to pixel correlation statistics*

Description

Returns the x and y dimension pixel-to-pixel correlation (often called covariance) at various scales, optionally returning a diagnostic plot.

Usage

```
profoundPixelCorrelation(image = NULL, objects = NULL, mask = NULL, sky = 0, skyRMS = 1,
lag = c(1:9, 1:9 * 10, 1:9 * 100, 1:9 * 1000, 1:9 * 10000), fft = TRUE, plot = FALSE,
ylim=c(-1,1), log='x', grid=TRUE, ...)
profoundSkySplitFFT(image = NULL, objects = NULL, mask = NULL, sky = 0, skyRMS = 1,
skyscale = 100, profound = NULL)
```

Arguments

`image`	Numeric matrix; required, the image we want to analyse. Note, image NAs are treated as masked pixels.

`objects`	Boolean matrix; optional, object mask where 1 is object and 0 is sky. If provided, this matrix *must* be the same dimensions as image.
`mask`	Boolean matrix; optional, parts of the ‘`image`’ to mask out (i.e. ignore), where 1 means mask out and 0 means use for analysis. If provided, this matrix *must* be the same dimensions as ‘`image`’.
`sky`	Numeric; the absolute sky level. Can be a scalar or a matrix matching the dimensions of ‘`image`’ (allows values to vary per pixel).
`skyRMS`	Numeric; the RMS of the sky. Can be a scalar or a matrix matching the dimensions of ‘`image`’ (allows values to vary per pixel).
`lag`	Interger verctor; the pixel lags to measure pixel-to-pixel correlation over the x and y dimensions.
`fft`	Logical; if TRUE the 2D FFT is computed and the modulus image matrix is returned to ‘`fft`’ and the (‘`image`’-‘`sky`’)/‘`skyRMS`’ is return to ‘`image_sky`’, if FALSE the ‘`fft`’ and ‘`image_sky`’ objects are returned as NULL. ‘`object`’ and ‘`mask`’ pixels are used to identify pixels to replace as described below.
`plot`	Logical; should a x/y correlation diagnostic plot be generated?
`ylim`	Numeric vector; range of data to display (see `magplot` for details). Only relevant if ‘`plot`’=TRUE.
`log`	Character scalar; log axis arguments to be passed to plot. E.g. use ’x’, ’y’, ’xy’ or ’yx’ as appropriate (see `magplot` for details). Only relevant if ‘`plot`’=TRUE.
`grid`	Logical; indicates whether a background grid should be drawn onto the plotting area (see `magplot` for details). Only relevant if ‘`plot`’=TRUE.
`skyscale`	Numeric scalar; required, the pixel scale that the FFT should split the provided ‘`image_sky`’ at. This should be chosen so as to separate out true sky modes and possible sources still in the sky. Too small and real sources will be put into the ‘`sky_lo`’ image returned, so larger is usually safer.
`profound`	List; object of class ’profound’. If this is provided then missing input arguments are taking directly from this structure (see Examples). As an added convenience, you can assign the profound object directly to the ‘`image`’ input.
`...`	Further arguments to passe to `magplot`. Only relevant if ‘`plot`’=TRUE.

Details

`profoundPixelCorrelation`:

All statistics are computed on (‘`image`’-‘`sky`’)/‘`skyRMS`’. If ‘`fft`’=TRUE this matrix is return to ‘`image_sky`’.

The function is useful to assessing a number of image attributes. For one things it tells you whether all spatial variance has been detected and removed at small scales as objects (e.g. using `profoundProFound`), or at larger scales as sky fluctuations. Assuming the object detection and sky removal has worked well, the remaining pixel-to-pixel correlation likely represents instrument level covariance. In practice nearly all processes produce positive pixel correlation, but it is not impossible that negative correlation can be introduced during the reduction process, particularly when over-subtracting the sky around bright stars.

For calculating the raw pixel-to-pixel correlation (as returned by ‘`cortab`’) ‘`mask`’ and ‘`object`’ pixels are ignored, so correlation is only considered where both pixels are flagged as un-masked sky pixels. The 2D image FFT output ('fft') replaces masked or object pixels with Normally distributed noise after the input 'image' has had the 'sky' subtracted and divided by the 'skyRMS'. Note that this means the FFT generated is partly stochastic (it will differ a bit each time it is run), but in practice it will be quite persistant for large scales (the centre) and stochastic at small scales (around the edge of the FFT image).

The slightly weird units used for the k modes of the FFT (see the value section below) is convenient because it means we can correctly label the FFT image in integer pixels counting out from the centre. The way to interpret the k-modes is that if you have an image of size L=356x356 then you can find the pixel representing a particular scale by computing L/S, where S is the scale of interest in pixels. I.e. S=356 is the mode representing the full image length scale since L/S=1 and can be found 1 pixel from the centre, whilst S=178/89 represents the half/quarter image scale and can be found at pixels L/S=2 or 4 (respectively) from the centre. From this reasoning we have Nyqvist sampling at 356/2=178 pixels from the centre (i.e. the edges of the FFT image).

The relative standard-deviations returned in 'cortab' are calculated by taking the standard-deviation of the lagged pixel differences of ('image'-'sky')/'skyRMS' and dividing through by sqrt(2). This means for well behaved data they should be 1, and the dashed lines on the diagnostic plot should fall on 1.

`profoundSkySplitFFT`:

The FFT split output separates the provided image into hi k ('sky_hi') and low k ('sky_lo') modes. The idea is that 'sky_lo' might represent additional sky with complex structure (not captured by the bicubic/bilinear extimated sky) that still needs to be subtracted off the image, whilst 'sky_hi' might contain some as yet un-subtracted sources.

In principle `profoundSkySplitFFT` can be run with any image, but the separation into the low and high k modes is not easily interpretable in the presence of many real objects since they will dominate the power at all scales (trust me on this).

Value

`profoundPixelCorrelation`:

A list containing three objects:

- cortab: A data.frame containing:
  - lag: The pixel lag
  - corx: The correlation in the x-dimension
  - cory: The correlation in the y-dimension
  - corx_neg: The correlation of sub sky versus sky pixels in x
  - cory_neg: The correlation of sub sky sversus ky pixels in y
  - corx_pos: The correlation of excess sky versus sky pixels in x
  - cory_pos: The correlation of excess sky versus sky pixels in y
  - corx_diff: corx_pos - corx_neg
  - cory_diff: cory_pos - cory_neg
  - relsdx: The pixel lag implied relative standard-deviation in x
  - relsdy: The pixel lag implied relative standard-deviation in y

- fft: if ‘fft’=TRUE this object contains a list containing x, y, and z. If ‘fft’=FALSE it is NULL. x and y contain the k mode values of the 2D FFT in units of (2.pi)/(L.pix), where L is the original dimensions of the image being Fourier transformed in x and y respectively. z contains the power component of the 2D FFT image as a numeric matrix; the modulus of the 2D FFT of the ‘image’ with the same dimensions. We use the optical representation, where the DC (or k=0) mode is in the absolute centre. This means larger scale produce power in the central parts of the FFT image, and smaller scales produce power in the outer parts of the FFT image.
- image_sky: Numeric matrix; if ‘fft’=TRUE this object contains the (‘image’-‘sky’)/‘skyRMS’, if ‘fft’=FALSE it is NULL.

  profoundSkySplitFFT:

  A list containing three numeric matrices:
- skyThe new sky estimate, defined as the input ‘sky’+‘sky_lo’.
- sky_loThe low k modes extracted from the objects masked ‘image’-‘sky’.
- sky_hiThe high k modes extracted from the objects masked ‘image’-‘sky’.

Author(s)

Aaron Robotham

See Also

profoundProFound

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits', package="ProFound"))

profound=profoundProFound(image, skycut=1.5, magzero=30, verbose=TRUE, plot=TRUE)

corout_raw=profoundPixelCorrelation(image$imDat, plot=TRUE)
magimage(corout_raw$fft, xlab='kx (2pi/356pix)', ylab='ky (2pi/356pix)')
points(0, 0, cex=10, col='red')

# There is clearly some residual structure masking out the brighter parts of objects:

corout_objects=profoundPixelCorrelation(image$imDat, sky=profound$sky,
skyRMS=profound$skyRMS, objects=profound$objects, plot=TRUE)
magimage(corout_objects$fft, xlab='kx (2pi/356pix)', ylab='ky (2pi/356pix)')
points(0, 0, cex=10, col='red')

# Using the more aggressive objects_redo removed nearly all of this:

corout_objects_redo=profoundPixelCorrelation(image$imDat, sky=profound$sky,
skyRMS=profound$skyRMS, objects=profound$objects_redo, plot=TRUE)
magimage(corout_objects_redo$fft, xlab='kx (2pi/356pix)', ylab='ky (2pi/356pix)')
points(0, 0, cex=10, col='red')

# We can use the pixel correlation function, in particular the FFT output, to assess how
```

```
# much further we can afford to push the source extraction in our image.

profound=profoundProFound(image, skycut=2.0, magzero=30, verbose=TRUE, plot=TRUE)
corout_objects_redo=profoundPixelCorrelation(image$imDat, sky=profound$sky,
skyRMS=profound$skyRMS, objects=profound$objects_redo)
magimage(corout_objects_redo$image_sky)
profoundProFound(corout_objects_redo$fft$z, skycut=2, verbose=TRUE, plot=TRUE)

profound=profoundProFound(image, skycut=1.5, magzero=30, verbose=TRUE, plot=TRUE)
corout_objects_redo=profoundPixelCorrelation(image$imDat, sky=profound$sky,
skyRMS=profound$skyRMS, objects=profound$objects_redo)
magimage(corout_objects_redo$image_sky)
profoundProFound(corout_objects_redo$fft$z, skycut=2, verbose=TRUE, plot=TRUE)

profound=profoundProFound(image, skycut=1.0, magzero=30, verbose=TRUE, plot=TRUE)
corout_objects_redo=profoundPixelCorrelation(image$imDat, sky=profound$sky,
skyRMS=profound$skyRMS, objects=profound$objects_redo)
magimage(corout_objects_redo$image_sky)
profoundProFound(corout_objects_redo$fft$z, skycut=2, verbose=TRUE, plot=TRUE)

profound=profoundProFound(image, skycut=0.8, magzero=30, verbose=TRUE, plot=TRUE)
corout_objects_redo=profoundPixelCorrelation(image$imDat, sky=profound$sky,
skyRMS=profound$skyRMS, objects=profound$objects_redo)
magimage(corout_objects_redo$image_sky)
profoundProFound(corout_objects_redo$fft$z, skycut=2, verbose=TRUE, plot=TRUE)

profound=profoundProFound(image, skycut=0.6, magzero=30, verbose=TRUE, plot=TRUE)
corout_objects_redo=profoundPixelCorrelation(image$imDat, sky=profound$sky,
skyRMS=profound$skyRMS, objects=profound$objects_redo)
magimage(corout_objects_redo$image_sky)
profoundProFound(corout_objects_redo$fft$z, skycut=2, verbose=TRUE, plot=TRUE)

# By doing ProFoundsource detection on the FFT itself it tells us if there are significant
# sources of a certain common scale (usually small) still in the image to extract.
# The levels above suggest we cannot push much further than a skycut=1.0. Clearly using
# skycut=0.6 introduces a lot of fake sources.

# We can improve the sky using profoundSkySplitFFT

profound=profoundProFound(image, type="bicubic")
newsky=profoundSkySplitFFT(image$imDat, objects=profound$objects_redo, sky=profound$sky,
skyRMS=profound$skyRMS)

# For convenience, the above is the same as running:

newsky=profoundSkySplitFFT(profound=profound)

# For super added convenience you can also un:

newsky=profoundSkySplitFFT(profound)

# Old versus new sky:
```

```
magimage(profound$sky)
magimage(newsky$sky)

# Original image, old sky subtraction and new sky subtraction (pretty subtle!):

magimage(image$imDat)
magimage(image$imDat-profound$sky)
magimage(image$imDat-newsky$sky)

# Be warned, you need a reasonable estimate of the sky and objects before running this.
# If we run on the original image that even the high/low k modes look very odd:

magimage(profoundSkySplitFFT(image$imDat)$sky_lo)
magimage(profoundSkySplitFFT(image$imDat)$sky_hi)

## End(Not run)
```

`profoundSegimGroup` *Create Segmentation Groups*

Description

Given an input segmentation map, returns a map of groups of touching segments as well as the IDs of segments within each group.

Usage

```
profoundSegimGroup(segim = NULL)
```

Arguments

`segim`	Integer matrix; required, the segmentation map.

Details

To use this function you will need to have EBImage installed. Since this can be a bit cumbersome on some platforms (given its dependencies) this is only listed as a suggested package. You can have a go at installing it by running:

> source("http://bioconductor.org/biocLite.R")

> biocLite("EBImage")

Linux users might also need to install some non-standard graphics libraries (depending on your install). If you do not have them already, you should look to install **jpeg** and **tiff** libraries (these are apparently technically not entirely free, hence not coming by default on some strictly open source Linux variants).

`profoundSegimGroup` uses the `bwlabel` function from EBImage.

Value

A list containting the following structures:

- `groupim` An map of the unique groups identified in the input '`segim`', where the groupID is the same as the lowest valued segID in the group.
- `groupsegID` A data.frame of lists giving the segIDs of segments in each group.

The data.frame returned by '`groupsegID`' is a slightly unusal structure to see in R, but it allows for a compact manner of storing uneven vectors of grouped segments. E.g. you might have a massive group containing 30 other segments and many groups containing a single segment. Padding a normal matrix out to accommodate the larger figure would be quite inefficient. It contains the following:

- `groupID` Group ID, which can be matched against values in '`groupim`'
- `segID` An embedded list of segmentation IDs for segments in the group. I.e. each list element of '`segID`' is a vector (see Examples for clarity).
- `Ngroup` The total number of segments that are in the group.
- `Npix` The total number of pixels that are in the group.

Author(s)

Aaron Robotham

See Also

`profoundSegimNear`, ~~~

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits', package="ProFound"))
profound=profoundProFound(image, skycut=1.5, magzero=30, verbose=TRUE)

#Look for nearby (in this case touching) neighbours

group=profoundSegimGroup(profound$segim)

#Look at the first few rows (groups 1:5):

group$groupsegID[1:5,]

#To access the embedded vectors you have to use unlist:

unlist(group$groupsegID[1,2])

#We can check to see which segments are in group number 1:

profoundSegimPlot(image$imDat, profound$segim)
magimage(group$groupim==1, col=c(NA,'red'), add=TRUE)

## End(Not run)
```

profoundSegimInfo *Image Segmentation Statistics*

Description

Basic summary statistics for image segments, e.g. aperture parameters, fluxes and surface brightness estimates. These might provide useful first guesses to ProFit fitting parameters (particularly 'flux', 'axrat' and 'ang').

Usage

```
profoundSegimStats(image = NULL, segim = NULL, mask = NULL, sky = 0, skyRMS = 0,
magzero = 0, gain = NULL, pixscale = 1, header, sortcol = "segID", decreasing = FALSE,
rotstats = FALSE, boundstats = FALSE, offset = 1)
profoundSegimPlot(image = NULL, segim = NULL, mask = NULL, sky = 0, header = NULL,
col = rainbow(max(segim), end=2/3), profound = NULL, ...)
```

Arguments

image	Numeric matrix; required, the image we want to analyse. Note, image NAs are treated as masked pixels.
segim	Integer matrix; required, the segmentation map of the 'image'. This matrix *must* be the same dimensions as 'image'.
mask	Boolean matrix; optional, parts of the image to mask out (i.e. ignore), where 1 means mask out and 0 means use for analysis. If provided, this matrix *must* be the same dimensions as 'image'.
sky	User provided estimate of the absolute sky level. Can be a scalar or a matrix matching the dimensions of 'image' (allows values to vary per pixel). This will be subtracted off the 'image' internally, so only provide this if the sky does need to be subtracted!
skyRMS	User provided estimate of the RMS of the sky. Can be a scalar or a matrix matching the dimensions of 'image' (allows values to vary per pixel).
magzero	Numeric scalar; the magnitude zero point. What this implies depends on the magnitude system being used (e.g. AB or Vega). If provided along with 'pixscale' then the flux and surface brightness outputs will represent magnitudes and mag/asec^2.
gain	Numeric scalar; the gain (in photo-electrons per ADU). This is only used to compute object shot-noise component of the flux error (else this is set to 0).
pixscale	Numeric scalar; the pixel scale, where pixscale=asec/pix (e.g. 0.4 for SDSS). If set to 1 (default), then the output is in terms of pixels, otherwise it is in arcseconds. If provided along with 'magzero' then the flux and surface brightness outputs will represent magnitudes and mag/asec^2.
header	Full FITS header in table or vector format. If this is provided then the segmentations statistics table will gain 'RAcen' and 'Decen' coordinate outputs. Legal table format headers are provided by the read.fitshdr function or the 'hdr'

list output of `read.fits` in the astro package; the 'hdr' output of `readFITS` in the `FITSio` package or the 'header' output of `magcutoutWCS`. Missing header keywords are printed out and other header option arguments are used in these cases. See `magWCSxy2radec`.

`sortcol`	Character; name of the output column that the returned segmentation statistics data.frame should be sorted by (the default is segID, i.e. segment order). See below for column names and contents.
`decreasing`	Logical; if FALSE (default) the segmentation statistics data.frame will be sorted in increasing order, if TRUE the data.frame will be sorted in decreasing order.
`rotstats`	Logical; if TRUE then the '`asymm`', '`flux_reflect`' and '`mag_reflect`' are computed, else they are set to NA. This is because they are very expensive to compute compared to other photometric properties.
`boundstats`	Logical; if TRUE then various pixel boundary statistics are computed ('`Nedge`', '`Nsky`', '`Nobject`', '`Nborder`', '`edge_frac`', '`edge_excess`' and '`FlagBorder`'). If FALSE these return NA instead (saving computation time).
`offset`	Integer scalar; the distance to offset when searching for nearby segments.
`col`	Colour palette; the colours to map the segment IDs against. This is by default the magnitude using a rainbow palette, going from red for bright segments, via green, to blue for faint segments.
`profound`	List; object of class 'profound'. If this is provided then missing input arguments are taking directly from this structure. As an added convenience, you can assign the profound object directly to the '`image`' input.
`...`	Further arguments to be passed to `magimage`.

Details

`profoundSegimStats` provides summary statistics for the individual segments of the image, e.g. properties of the apertures, and the sum of the flux etc. This is used inside of `profoundMakeSegim` and `profoundMakeSegimExpand`, but it may be useful to use separately if manual modifications are made to the segmentation, or two segmentations (e.g. a hot and cold mode segmentation) need to be combined.

The interpretation of some of these outputs will depend a lot on the data being analysed, so it is for the user to decide on sensible next steps (e.g. using the outputs to select stars etc). One output of interest might be '`flux_reflect`'. This attempts to correct for missing flux where segments start colliding. This probably returns an upper limit to the flux since in some regions it can even be double counted if the two sources that have colliding segmentation maps are very close together and similar in brightness, so somewhere between '`flux`' and '`flux_reflect`' the truth probably lies. If you want a better estimate of the flux division then you should really be using the profiling routine of `ProFit`.

`profoundSegimPlot` is useful when you only have a small number of sources (roughly a few hundred). With more than this it can start to take a long time to make the plot! If you provide a header or a list containing the iamge and header to '`header`' then it will be plotted with the WCS overlaid using `magimageWCS`, otherwise it will use `magimage`.

Value

A data.frame with columns:

`segID`	Segmentation ID, which can be matched against values in ‘`segim`’
`uniqueID`	Unique ID, which is fairly static and based on the xmax and ymax position
`xcen`	Flux weighted x centre
`ycen`	Flux weighted y centre
`xmax`	x position of maximum flux
`ymax`	y position of maximum flux
`RAcen`	Flux weighted degrees Right Ascension centre (only present if a ‘`header`’ is provided)
`Deccen`	Flux weighted degrees Declination centre (only present if a ‘`header`’ is provided)
`RAmax`	Right Ascension of maximum flux (only present if a ‘`header`’ is provided)
`Decmax`	Declination of maximum flux (only present if a ‘`header`’ is provided)
`sep`	Radial offset between the cen and max definition of the centre (units of ‘`pixscale`’, so if ‘`pixscale`’ represents the standard asec/pix this will be asec)
`flux`	Total flux (calculated using ‘`image`’-‘`sky`’) in ADUs
`mag`	Total flux converted to mag using ‘`magzero`’
`cenfrac`	Fraction of flux in the brightest pixel
`N50`	Number of brightest pixels containing 50% of the flux
`N90`	Number of brightest pixels containing 90% of the flux
`N100`	Total number of pixels in this segment, i.e. contains 100% of the flux
`R50`	Approximate elliptical semi-major axis containing 50% of the flux (units of ‘`pixscale`’, so if ‘`pixscale`’ represents the standard asec/pix this will be asec)
`R90`	Approximate elliptical semi-major axis containing 90% of the flux (units of ‘`pixscale`’, so if ‘`pixscale`’ represents the standard asec/pix this will be asec)
`R100`	Approximate elliptical semi-major axis containing 100% of the flux (units of ‘`pixscale`’, so if ‘`pixscale`’ represents the standard asec/pix this will be asec)
`SB_N50`	Mean surface brightness containing brightest 50% of the flux, calculated as ‘`flux`’*0.5/‘`N50`’ (if ‘`pixscale`’ has been set correctly then this column will represent mag/asec^2. Otherwise it will be mag/pix^2)
`SB_N90`	Mean surface brightness containing brightest 90% of the flux, calculated as ‘`flux`’*0.9/‘`N90`’ (if ‘`pixscale`’ has been set correctly then this column will represent mag/asec^2. Otherwise it will be mag/pix^2)
`SB_N100`	Mean surface brightness containing all of the flux, calculated as ‘`flux`’/‘`N100`’ (if ‘`pixscale`’ has been set correctly then this column will represent mag/asec^2. Otherwise it will be mag/pix^2)
`xsd`	Weighted standard deviation in x (always in units of pix)
`ysd`	Weighted standard deviation in y (always in units of pix)

covxy	Weighted covariance in xy (always in units of pix)
corxy	Weighted correlation in xy (always in units of pix)
con	Concentration, 'R50'/'R90'
asymm	180 degree flux asymmetry (0-1, where 0 is perfect symmetry and 1 complete asymmetry)
flux_reflect	Flux corrected for asymmetry by doubling the contribution of flux for asymmetric pixels (defined as no matching segment pixel found when the segment is rotated through 180 degrees)
mag_reflect	'flux_reflect' converted to mag using 'magzero'
semimaj	Weighted standard deviation along the major axis, i.e. the semi-major first moment, so ~2 times this would be a typical major axis Kron radius (always in units of pix)
semimin	Weighted standard deviation along the minor axis, i.e. the semi-minor first moment, so ~2 times this would be a typical minor axis Kron radius (always in units of pix)
axrat	Axial ratio as given by min/maj
ang	Orientation of the semi-major axis in degrees. This has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical)
signif	Approximate singificance of the detection using the Chi-Square distribution
FPlim	Approximate false-positive significance limit below which one such source might appear spuriously on an image this large
flux_err	Estimated total error in the flux for the segment
mag_err	Estimated total error in the magnitude for the segment
flux_err_sky	Sky subtraction component of the flux error
flux_err_skyRMS	Sky RMS component of the flux error
flux_err_shot	Object shot-noise component of the flux error (only if 'gain' is provided)
sky_mean	Mean flux of the sky over all segment pixels
sky_sum	Total flux of the sky over all segment pixels
skyRMS_mean	Mean value of the sky RMS over all segment pixels
Nedge	Number of edge segment pixels that make up the outer edge of the segment
Nsky	Number of edge segment pixels that are touching sky
Nobject	Number of edge segment pixels that are touching another object segment
Nborder	Number of edge segment pixels that are touching the 'image' border
Nmask	Number of edge segment pixels that are touching a masked pixel (note NAs in 'image' are also treated as masked pixels)
edge_frac	Fraction of edge segment pixels that are touching the sky i.e. 'Nsky''Nedge', higher generally meaning more robust segmentation statistics
edge_excess	Ratio of the number of edge pixels to the expected number given the elliptical geometry measurements of the segment. If this is larger than 1 then it is a sign that the segment geometry is irregular, and is likely a flag for compromised photometry

flag_border A binary flag telling the user which 'image' borders the segment touches. The bottom of the 'image' is flagged 1, left=2, top=4 and right=8. A summed combination of these flags indicate the segment is in a corner touching two borders: bottom-left=3, top-left=6, top-right=12, bottom-right=9.

profoundSegimPlot is a simple function that overlays the image segments on the original 'image'. This can be very slow for large numbers (1,000s) of segments because it uses the base contour function to draw the segments individually.

Author(s)

Aaron Robotham

See Also

profoundProFound, profoundMakeSegim, profoundMakeSegimExpand

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits', package="ProFound"))
profound=profoundProFound(image, magzero=30, rotstats=TRUE)

print(profound$segstats)

#Note row 6 (the central galaxy) gains 0.05 mag of flux due to the missing flux when
#rotated through 180 degrees. The reflected value of 18.4 is closer to the full profile
#solution (~18.35) than the non-reflected flux (18.45).

profound$segim[35:55, 80:100]=max(profound$segim)+1
print(profoundSegimStats(image$imDat, segim=profound$segim, sky=profound$sky,
header=image$hdr))
profoundSegimPlot(image, profound$segim)

## End(Not run)
```

---

profoundSegimKeep *Merge Segmentation Map with Grouped Segmentation map*

Description

Allows users to safely merge a standard segim with a groupim, where you can specify segments to be newly merged together, or groups to be merged.

Usage

```
profoundSegimKeep(segim = NULL, groupim = NULL, groupID_merge = NULL, segID_merge = NULL,
clean = FALSE)
```

Arguments

- `segim` — Integer matrix; required, the segmentation map.
- `groupim` — Integer matrix; the grouped segmentation map. This matrix *must* be the same dimensions as 'segim' (if supplied).
- `groupID_merge` — Integer vector; the group IDs that the user wants to persist into the final segmentation map (removing all 'segim' segments that overlap with any of the specified group IDs).
- `segID_merge` — Integer list; each list element should specify collections of segments to be merged.
- `clean` — Logical; should segments partially overlapping with chosen groups be aggressively removed?

Details

The merged segments inherit the lowest segment value, e.g. list(c(1,2,4),c(5,6)) would merge together segments 1,2,4 and to be a new segment 1, and then 5,6 to be a new segment 5.

Value

Integer matrix; the merged segmentation map, where specified groups and segments have been merged.

Author(s)

Aaron Robotham

See Also

profoundSegimMerge

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits', package="ProFound"))

profound=profoundProFound(image, magzero=30, groupstats=TRUE, verbose=TRUE, plot=TRUE)

segim_new=profoundSegimKeep(profound$segim, profound$group$groupim, groupID_merge=1,
segID_merge=list(c(12, 26, 62), c(13, 24)))

profoundSegimPlot(image, segim=segim_new)

## End(Not run)
```

profoundSegimMerge *Merge Segmentation Maps*

Description

Takes two segmentation maps and merges them in a sensible manner, making sure segments representing the same object are not overlaid on each other.

Usage

```
profoundSegimMerge(image = NULL, segim_base = NULL, segim_add = NULL, mask = NULL,
sky = 0)
```

Arguments

image	Numeric matrix; required, the image we want to analyse. Note, image NAs are treated as masked pixels.
segim_base	Integer matrix; required, the base segmentation map of the 'image'. This matrix *must* be the same dimensions as 'image'.
segim_add	Integer matrix; required, the new segmentation map of the 'image' that is to be added. This matrix *must* be the same dimensions as 'image'.
mask	Boolean matrix; optional, parts of the image to mask out (i.e. ignore), where 1 means mask out and 0 means use for analysis. If provided, this matrix *must* be the same dimensions as 'image'.
sky	User provided estimate of the absolute sky level. Can be a scalar or a matrix matching the dimensions of 'image' (allows values to vary per pixel). This will be subtracted off the 'image' internally, so only provide this if the sky does need to be subtracted!

Details

The merger strategy is quite simple. Matching object segments are identified by the 'uniqueID' ID from an internal run of profoundSegimStats. Whichever segment contains more flux is determined to be the best map to use as the base segment. Unmatched segments in the 'segim_add' map are added back in after this initial merging process, so will end up on top and potentially appear as segment islands within larger segments (which is not possible using the standard segmentation process in profoundMakeSegim).

An obvious reason to use this function is in situations where bright stars are embedded deep within an extended source. The standard watershed segmentation used in profoundMakeSegim will tend to break a large portion of the extended source off to form the segmented region. By running profoundProFound in different modes it is possible to identify the bright peaks (see Examples below), and then use profoundSegimMerge to piece the segments back together appropriately.

Value

Integer matrix; the merged segmentation map matched pixel by pixel to 'image'.

Author(s)

Aaron Robotham

See Also

profoundMakeSegim, ~~~

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits',
package="ProFound"))$imDat
profound=profoundProFound(image, plot=TRUE)
profound_diff=profoundProFound(profoundImDiff(image, sigma=2), plot=TRUE)
tempmerge=profoundSegimMerge(image, profound$segim, profound_diff$segim)

#Notice the new embedded blue segment near the centre:

profoundSegimPlot(image, segim=tempmerge)

## End(Not run)
```

---

profoundSegimNear *Segment Neighbour IDs*

Description

Returns a data.frame of all nearby (default is touching) segments surrounding every segment in a provided segim.

Usage

```
profoundSegimNear(segim = NULL, offset = 1)
```

Arguments

segim	Integer matrix; a specified segmentation map of the image (required).
offset	Integer scalar; the distance to offset when searching for nearby segments.

Details

This function can be run by the user directly, but usually it is called from within a higher routine in the ProFound suite of objects detection functions.

Value

A data.frame of lists giving the segIDs of nearby segments for every segment. This is a slightly unusal structure to see in R, but it allows for a compact manner of storing uneven vectors of touching segmentss. E.g. you might have a massive segment touching 30 other segments and many segments touching none. Padding a normal matrix out to accommodate the larger figure would be quite inefficient.

`segID`	Segmentation ID, which can be matched against values in '`segim`'
`nearID`	An embedded list of segmentation IDs for nearby segments. I.e. each list element of '`nearID`' is a vector (see Examples for clarity).
`Nnear`	The total number of segments that are considered to be nearby.

Note

Due to the construction of the segmented curve-of-growth in ProFound you may have cases where the separation between segments is two or three pixels. Since these are very close to touching you might want to catch these close neighbours rather than strictly touching. By increasing '`offset`' to a larger number (2 or 3 in the cases above) you can flag these events.

Author(s)

Aaron Robotham

See Also

`profoundProFound`, `profoundMakeSegim`, `profoundMakeSegimDilate`, `profoundMakeSegimExpand`, `profoundSegimStats`, `profoundSegimPlot`

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits', package="ProFound"))
profound=profoundProFound(image, skycut=1.5, magzero=30, verbose=TRUE)

#Look for nearby (in this case touching) neighbours

near=profoundSegimNear(profound$segim)

#Look at the first few rows (segIDs 1:5):

near[1:5,]

#To access the embedded vectors you have to use unlist:

unlist(near[3,2])

#We can check to see which segments are touching segID number 3:

profoundSegimPlot(image$imDat, profound$segim)
magimage(profound$segim==3, col=c(NA,'red'), add=TRUE)
```

```
magimage(matrix(profound$segim %in% unlist(near[3,2]), dim(profound$segim)[1]),
col=c(NA,'blue'), add=TRUE)

## End(Not run)
```

---

profoundSegimWarp *Remap Segmentation Map via Warping*

Description

Remaps an input segmentation map WCS Tan Gnomonic projection system to a different target WCS. This uses magwarp with sensible settings, but magwarp can be used more directly if the other lower level options are required. This interface should cover most practical use cases though. Using profoundProFound with a remapped segmentation map is likely to be more sensible than remapping image flux since it will not produce flux interpolation errors.

Usage

```
profoundSegimWarp(segim_in = NULL, header_in = NULL, header_out = NULL)
```

Arguments

segim_in	Integer matrix; required, the segmentation map we want to remap. If 'segim_in' is a list as created by readFITS, read.fits of magcutoutWCS then the image part of the list is parsed to 'segim_in' and the correct header part is passed to 'header_in'.
header_in	Full FITS header in table or vector format. This should be the header WCS that matches 'segim_in'. Legal table format headers are provided by the read.fitshdr function or the 'hdr' list output of read.fits in the astro package; the 'hdr' output of readFITS in the FITSio package or the 'header' output of magcutoutWCS. If 'header_in' is provided then key words will be taken from here as a priority. Missing header keywords are printed out and other header option arguments are used in these cases.
header_out	Full FITS header in table or vector format. This is the target WCS projection that 'segim_in' will be mapped onto. Legal table format headers are provided by the read.fitshdr function or the 'hdr' list output of read.fits in the astro package; the 'hdr' output of readFITS in the FITSio package or the 'header' output of magcutoutWCS. If 'header_out' is provided then key words will be taken from here as a priority. Missing header keywords are printed out and other header option arguments are used in these cases.

Details

This function uses the 'interpolation'='nearest' and 'doscale'=FALSE in magwarp.

Value

Integer matrix; the remapped image using the target WCS.

Author(s)

Aaron Robotham

See Also

magwarp

Examples

```
## Not run:
VST_r=readFITS(system.file("extdata", 'VST_r.fits', package="magicaxis"))
GALEX_NUV=readFITS(system.file("extdata", 'GALEX_NUV.fits', package="magicaxis"))

profound_KiDS=profoundProFound(VST_r, sky=0, skycut=1, sigma=2, tolerance=8, plot=TRUE)

segimFUV=profoundSegimWarp(profound_KiDS$segim, profound_KiDS$header, GALEX_NUV$hdr)

profoundSegimPlot(GALEX_NUV, segim = segimFUV)

profound_GALEX=profoundProFound(GALEX_NUV, segim=segimFUV, plot=TRUE)

## End(Not run)
```

---

profoundSkyEst *Old Sky Estimator (Somewhat Defunct)*

Description

A high level utility to estimate the sky properties of a supplied 'image'. This is closely related to the equivalent routines available in the LAMBDAR R package.

Usage

```
profoundSkyEst(image = NULL, objects = NULL, mask = NULL, cutlo = cuthi/2,
cuthi = sqrt(sum((dim(image)/2)^2)), skycut = 'auto', clipiters = 5, radweight = 0,
plot = FALSE, ...)
```

Arguments

image	Numeric matrix; required, the image we want to analyse. The galaxy should be approximately central within this image since annuli weighting is done to avoid brighter central regions dominated by galaxy flux.
objects	Boolean matrix; optional, object mask where 1 is object and 0 is sky. If provided, this matrix *must* be the same dimensions as 'image'.
mask	Boolean matrix; optional, non galaxy parts of the image to mask out, where 1 means mask out and 0 means use for analysis. If provided, this matrix *must* be the same dimensions as 'image'.

cutlo	Numeric scalar; radius where the code will start to calculate the sky annuli around the central object. Should be large enough to avoid significant object flux, i.e. a few times the flux 90 radius. Default is half of 'cuthi'.
cuthi	Numeric scalar; radius where the code will stop calculating the sky annuli around the central object. Default is the corner edge of the 'image'.
skycut	Numeric scalar; clipping threshold to make on the 'image' in units of the skyRMS. The default scales the clipping to the number of pixels in the 'image', and will usually work reasonably.
clipiters	Numeric scalar; How many iterative clips of the sky will be made.
radweight	Numeric scalar; what radius power-law weighting should be used to bias the sky towards sky annuli nearer to the central object. 'radweight'>0 weight the sky value more towards larger radii and 'radweight'<0 weight the sky values towards the 'image' centre. The default of 0 means there is no radial weightings. This becomes clear when plotting the 'radrun' output (see Examples). Note this behaves differently to the similarly named option in LAMBDAR's sky.estimate.
plot	Logical; should a diagnostic plot be generated?
...	Further arguments to be passed to magplot. Only relevant is 'plot'=TRUE.

Details

This function is closely modelled on the sky.estimate function in the LAMBDAR package (the basic elements of which were written by ASGR). The defaults work well for data where the main objects (usually a galaxy) is centrally located in the 'image' since the 'cutlo' default will usually ignore contaminated central pixels. On top of this it does pretty aggressive object pixel rejection using the 'skycut' and 'clipiters' options.

The defaults should work reasonably well on modern survey data (see Examples), but should the solution not be ideal try modifying these parameters (in order of impact priority): 'skycut', 'cutlo', 'radweight', 'clipiters'.

It is interesting to note that a better estimate of the sky RMS can be made by using the output of profoundImDiff (see Examples).

Value

Returns a list with 5 elements:

sky	The value of the estimated sky.
skyerr	The estimated uncertainty in the sky level.
skyRMS	The RMS of the sky pixels.
Nnearsky	The number of sky annuli that have error bars encompassing the final sky.
radrun	The output of magrun for radius versus sky pixels values.

Author(s)

Aaron Robotham

See Also

profoundMakeSegim, profoundMakeSegimExpand

Examples

```
## Not run:
image = readFITS(system.file("extdata", 'KiDS/G266035fitim.fits',
package="ProFit"))$imDat
sky1 = profoundSkyEst(image, plot=TRUE)
image_sky = image-sky1$sky
sky2 = profoundSkyEst(profoundImDiff(image_sky), plot=TRUE)

#You can check whether you are contaminated by the central objects by plotting the radrun
#object in the list (it should be flat for a well behaved sky):
sky = profoundSkyEst(image, cutlo=0, plot=TRUE)
magplot(sky$radrun)
abline(h=sky$sky)

#The above shows heavy contamination by the central object without. We can either mask
#this out using the output of profoundSegImWatershed, set cutlo to be larger or weight
#the sky towards outer annuli.

profound=profoundProFound(image)
sky = profoundSkyEst(image, mask=profound$objects, cutlo=0, plot=TRUE)
magplot(sky$radrun)
abline(h=sky$sky)

#The above is better, but not great. A more aggressive mask helps:

sky = profoundSkyEst(image, mask=profound$objects_redo, cutlo=0, plot=TRUE)
magplot(sky$radrun)
abline(h=sky$sky)

#Or weighting the sky to outer radii

sky = profoundSkyEst(image, mask=profound$objects, cutlo=0, radweight=1, plot=TRUE)
magplot(sky$radrun)
abline(h=sky$sky)

#Finally we can leave the central cutlo mask turned on:

sky = profoundSkyEst(image, mask=profound$objects, plot=TRUE)
magplot(sky$radrun)
abline(h=sky$sky)

## End(Not run)
```

---

profoundSkyEstLoc *Calculate Sky in Subset of Pixels*

Description

Calculate the sky and sky RMS for a subset region of a larger image, as used in profoundMakeSkyMap.

Usage

```
profoundSkyEstLoc(image = NULL, objects = NULL, mask = NULL, loc = dim(image)/2,
box = c(100, 100), skytype = "median", skyRMStype = "quanlo", sigmasel = 1,
skypixmin = prod(box)/2, boxadd = box/2, boxiters = 0, doclip = TRUE, shiftloc = FALSE,
paddim = TRUE, plot = FALSE, ...)
```

Arguments

image	Numeric matrix; required, the image we want to analyse.
objects	Boolean matrix; optional, object mask where 1 is object and 0 is sky. If provided, this matrix *must* be the same dimensions as 'image'.
mask	Boolean matrix; optional, non galaxy parts of the image to mask out, where 1 means mask out and 0 means use for analysis. If provided, this matrix *must* be the same dimensions as 'image'.
loc	Integer vector; the [x,y] location where we want to estimate the sky and sky RMS.
box	Integer vector; the dimensions of the box car filter to estimate the sky with.
skytype	Character scalar; the type of sky level estimator used. Allowed options are 'median' (the default), 'mean' and 'mode' (see Details for an explanation of what these estimators do). In all cases this is the estimator applied to unmasked and non-object pixels. If 'doclip'=TRUE then the pixels will be dynamically sigma clipped before the estimator is run.
skyRMStype	Character scalar; the type of sky level estimator used. Allowed options are 'quanlo' (the default), 'quanhi', 'quanboth', and 'sd' (see Details for an explanation of what these estimators do). In all cases this is the estimator applied to unmasked and non-object pixels. If 'doclip'=TRUE then the pixels will be dynamically sigma clipped before the estimator is run.
sigmasel	Numeric scalar; the quantile to use when trying to estimate the true standard-deviation of the sky distribution. If contamination is low then the default of 1 is about optimal in terms of S/N, but you might need to make the value lower when contamination is very high.
skypixmin	Numeric scalar; the minimum number of sky pixels desired in our cutout. The default is that we need half the original number of pixels in the 'box' to be sky.
boxadd	Integer vector; the dimensions to add to the 'box' to capture more pixels if 'skypixmin' has not been achieved.
boxiters	Integer scalar; the number of 'box'+'boxadd' iterations to attempt in order to capture 'skypixmin' sky pixels. The default means the box will not be grown at all.
doclip	Logical; should the unmasked non-object pixels used to estimate to local sky value be further sigma-clipped using magclip? Whether this is used or not is a product of the quality of the objects extraction. If all detectable objects really

have been found and the dilated objects mask leaves only apparent sky pixels then an advanced user might be confident enough to set this to FALSE. If an doubt, leave as TRUE.

shiftloc — Logical; should the cutout center shift from 'loc' if the desired 'box' size extends beyond the edge of the image? (See magcutout for details).

paddim — Logical; should the cutout be padded with image data until it meets the desired 'box' size (if 'shiftloc' is true) or padded with NAs for data outside the image boundary otherwise? (See magcutout for details).

plot — Logical; should a diagnostic plot be generated?

... — Further arguments to be passed to magimage. Only relevant is 'plot'=TRUE.

Details

This is a somewhat handy standalone utility function if you have a large image and want to check the quality and stability of the local sky and sky RMS.

Regarding 'skytype', the meaning of the median and mean options re obvious enough. The mode is computed by running the data through density with the default options including automatuc selection of the appropriate smoothing band-width. The peak value of the smoothed density is then extracted, and the pixel value at this point is returned as the 'mode' sky estimator.

Regarding 'skyRMStype', if you know that your contamination only comes from positive flux sources (e.g., astronomical data when trying to select sky pixels) then you should probably use the lower side to determine Normal statistics (quanlo). Similarly if the contamination is on the low side then you should use the higher side to determine Normal statistics (quanhi). If you believe the selected sky pixels to be unbiased then 'quanboth' uses both sides and will give you a more accurate estimator of the sky RMS. The final option is to use the standard-deviation, with the caveat that this is calculated around the esstimated sky level (of type specified by 'skytype') and not necessarily simply the mean (as it would be typically). The most common choices for 'skyRMStype' will likely be 'quanlo' or 'sd'.

There are many questions to think about when choosing the best combination of sky estimators. Have all detectable sources been robustly extracted and masked? Is the remaining contamintion due to background undetected sources or wing flux from foreground stars? The most significant choice to be made is whether to choose the more robust 'median' or the potentially biased 'mean'. The former makes sense if you think there might be detectable sources still contributing to your nominal sky pixels, the latter makes sense if the positive flux of undetected sources is spread round the sky in an random but uniform manner. If you are very confident that your object mask represents all plausible sources then you might even want to set 'doclip'=FALSE. The defaults behave in quite a safe manner and have resistance to unmasked objects being included in the sky pixels. Using different options (particularly 'doclip'=FALSE and 'skytype') requires more advanced knowledge about the specific data being anlysed.

Value

A 2 component list containing:

val — A length two vector where the first element is the sky and the second is the skyRMS.

clip — The full vector of pixels selected as being sky pixels (can then be plotted with maghist etc.)

Author(s)

Aaron Robotham

See Also

profoundSkyEst, profoundMakeSkyMap, profoundMakeSkyGrid

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits',
package="ProFound"))$imDat
profoundSkyEstLoc(image, loc=c(20,20), box=c(40,40), plot=TRUE)$val
profoundSkyEstLoc(image, loc=c(40,20), box=c(40,40), plot=TRUE)$val
profoundSkyEstLoc(image, loc=c(60,20), box=c(40,40), plot=TRUE)$val

## End(Not run)
```

---

water_cpp *Rcpp Watershed Function*

Description

This is a standalone implementation of a watershed deblend, with some astronomy specific tweeks. E.g. it is possible to both adapt the extent of the saddlepoint search, and it can be modified by both an absolute and relative tolerance. Defaults behave much like EBImage's watershed function. In general it is a factor of a few faster than the EBImage implementation, especially for large images with lots of deblending required.

Usage

```
water_cpp(image = 0L, nx = 1L, ny = 1L, abstol = 1, reltol = 0, cliptol = 1e+06, ext = 1L,
skycut = 0, pixcut = 1L, verbose = FALSE, Ncheck = 1000000L)
water_cpp_old(image = 0L, nx = 1L, ny = 1L, abstol = 1, reltol = 0, cliptol = 1e+06,
ext = 1L, skycut = 0, pixcut = 1L, verbose = FALSE, Ncheck = 1000000L)
```

Arguments

image	Numeric matrix; required, the image we want to analyse. Note, image NAs are treated as masked pixels.
nx	Integer scalar; required, the dimension x of the supplied 'image', i.e. should be dim(image)[1].
ny	Integer scalar; required, the dimension y of the supplied 'image', i.e. should be dim(image)[2].

abstol	Numeric scalar; the minimum height of the object in the units of image intensity between its highest point (seed) and the point where it contacts another object (checked for every contact pixel). If the height is smaller than the tolerance, the object will be combined with its brightest neighbour. Tolerance should be chosen according to the range of 'image'. Default works well when the 'image' has been divided by the sky-RMS. A larger value of 'abstol' means segments are more aggressively merged together.
reltol	Numeric scalar; a modifier to the 'abstol', modifying it by the ratio of the segment peak flux divided by the saddle point flux to the power 'reltol'. The default means the 'reltol' has no effect since this modifier becomes 1. A larger value of 'reltol' means segments are more aggressively merged together.
cliptol	Numeric scalar; if 'image' is above this level where segments touch then they are always merged, regardless of other criteria. When thinking in terms of sky RMS, values between 20-100 are probably appropriate for merging very bright parts of stars back together in optical data.
ext	Integer scalar; square offset of the neighborhood in pixels for the detection of neighboring objects. Higher value smoothes out small objects.
skycut	Numeric scalar; background value under which pixels are not considered any-more for watersheding.
pixcut	Integer scalar; the minimum number of pixels allowed in a segment. Below this number segments are set to 0, i.e. the background. This means they are not considered real objects in profoundProFound.
verbose	Logical; should verbose output be displayed to the user? Since big image can take a long time to run, you might want to monitor progress.
Ncheck	Integer scalar; the pixel scanning interval to check for interupts and for printing out the verbose state.

Details

This was hand written from scratch by A Robotham, but in the end the approach is somewhat similar to `EBImage::watershed`. There do seem to be fairly large speed improvements for more sparse images though, since only pixels above the background 'skycut' are ever looked at. This knowledge of sparcity does not exist in `EBImage::watershed`.

`water_cpp` is the newer variant re-writeen by R Tobar based on the Rcpp implentation. The older Rcpp one is still available as `water_cpp_old`.

Value

Integer matrix; the segmentation map matched pixel by pixel to 'image'.

Author(s)

Aaron Robotham

References

Some aspects of Meyer's floodfill used, but not explicitly based on any published approach, so might be in detail similar by accident.

See Also

profoundMakeSegim, ?EBImage::watershed

Examples

```
## Not run:
image=readFITS(system.file("extdata", 'VIKING/mystery_VIKING_Z.fits',
package="ProFound"))$imDat

segim=water_cpp(im=image, nx=dim(image_smooth)[1], ny=dim(image_smooth)[2], skycut=10)
magimage(segim, col=c(0,rainbow(1e3)))

## End(Not run)
```

Index