Package 'ProFit'

July 23, 2019

Type Package

Title Fit Projected 2D Profiles to Galaxy Images

Version 1.3.2

Date 2019-07-15

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

Description Get data / Define model / ??? / Profit! 'ProFit' is a Bayesian galaxy fitting tool that uses a fast 'C++' image generation library and a flexible interface to a large number of likelihood samplers.

License LGPL-3

URL https://github.com/ICRAR/ProFit

BugReports https://github.com/ICRAR/ProFit/issues

Depends R (>= 3.0), FITSio, magicaxis (>= 2.0.3)

Imports cubature, RColorBrewer, LaplacesDemon, methods, celestial (>= 1.4.1), checkmate

Suggests fftw, knitr, rmarkdown, ProFound, sn

VignetteBuilder knitr

NeedsCompilation yes

Author Aaron Robotham [aut, cre] (<https://orcid.org/0000-0003-0429-3579>), Dan Taranu [aut] (<https://orcid.org/0000-0001-6268-1882>), Rodrigo Tobar [aut] (<https://orcid.org/0000-0002-1052-0611>)

Repository CRAN

Date/Publication 2019-07-23 04:20:02 UTC

R topics documented:

`ProFit-package` *Fit Projected 2D Profiles to Galaxy Images*

Description

Get data / Define model / ??? / Profit! 'ProFit' is a Bayesian galaxy fitting tool that uses a fast 'C++' image generation library and a flexible interface to a large number of likelihood samplers.

Details

Package:	ProFit
Type:	Package
Version:	1.3.2
Date:	2019-07-15
License:	LGPL-3
Depends:	R (>= 3.0), FITSio, magicaxis (>= 2.0.3)
Imports:	cubature, RColorBrewer, LaplacesDemon, methods, celestial (>= 1.4.1), checkmate
Suggests:	fftw, knitr, rmarkdown, ProFound, sn

Author(s)

NA

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

References

Robotham A.S.G., et al., 2017, MNRAS, 466, 1513

Examples

```
modellist = list(
  sersic = list(
    xcen   = c(180, 60),
    ycen   = c(90, 10),
    mag = c(15, 13),
    re  = c(14, 5),
    nser  = c(3, 10),
    ang  = c(46, 80),
    axrat  = c(0.4, 0.6),
    box = c(0.5,-0.5)
  ),
  pointsource = list(
    xcen   = c(34,10,150),
    ycen   = c(74,120,130),
    mag = c(10,13,16)
  ),
  sky = list(
    bg = 3e-12
  )
)

# Without a PSF provided only the extended sources are shown, with no convolution:

magimage(profitMakeModel(modellist=modellist, dim=c(200,200)))
```

```
# With a PSF provided the PSFs are displayed and the extended sources are convolved with
# the PSF:

magimage(profitMakeModel(modellist=modellist, psf=profitMakePointSource(), dim=c(200,200)))

############### Full L-BFGS-B fit example ##############

## Not run:

# Load ProFit example data

# There are 2 data source options: KiDS or SDSS (the galaxies are the same)

datasource='KiDS'

# Now we can extract out the example files we have available for fitting by checking the
# contents of the directory containing the example FITS files:

data('ExampleInit')
ExampleFiles=list.files(system.file("extdata",datasource,package="ProFit"))
ExampleIDs=unlist(strsplit(ExampleFiles[grep('fitim',ExampleFiles)],'fitim.fits'))
print(ExampleIDs)

# There are 10 example galaxies included. Here we run example 1:

useID=ExampleIDs[1]

image = readFITS(system.file("extdata", paste(datasource,'/',useID,'fitim.fits',sep=''),
package="ProFit"))$imDat
sigma = readFITS(system.file("extdata", paste(datasource,'/',useID,'sigma.fits',sep=''),
package="ProFit"))$imDat
segim = readFITS(system.file("extdata", paste(datasource,'/',useID,'segim.fits',sep=''),
package="ProFit"))$imDat
psf = readFITS(system.file("extdata", paste(datasource,'/',useID,'psfim.fits',sep=''),
package="ProFit"))$imDat

# Very rough model (not meant to look too good yet):

useIDnum=as.integer(strsplit(useID,'G')[[1]][2])
useloc=which(ExampleInit$CATAID==useIDnum)

# For our initial model we treat component 1 as the putitive bulge and componet 2 as
# the putitive disk. We are going to attempt a fit where the disk is forced to have
# nser=1 and the bulge has an axial ratio of 1.

modellist=list(
  sersic=list(
    xcen= c(dim(image)[1]/2, dim(image)[1]/2),
    ycen= c(dim(image)[2]/2, dim(image)[2]/2),
    mag= c(ExampleInit$sersic.mag1[useloc], ExampleInit$sersic.mag2[useloc]),
    re= c(ExampleInit$sersic.re1[useloc], ExampleInit$sersic.re2[useloc])*
      if(datasource=='KiDS'){1}else{0.2/0.339},
    nser= c(ExampleInit$sersic.nser1[useloc], 1),  #Disk is initially nser=1
```

```
    ang= c(ExampleInit$sersic.ang2[useloc], ExampleInit$sersic.ang2[useloc]),
    axrat= c(1, ExampleInit$sersic.axrat2[useloc]),  #Bulge is initially axrat=1
    box=c(0, 0)
  )
)

# The pure model (no PSF):
magimage(profitMakeModel(modellist,dim=dim(image)))

# The original image:
magimage(image)

# The convolved model (with PSF):
magimage(profitMakeModel(modellist,dim=dim(image),psf=psf))

# What should we be fitting:

tofit=list(
  sersic=list(
    xcen= c(TRUE,NA), #We fit for xcen and tie the two togther
    ycen= c(TRUE,NA), #We fit for ycen and tie the two togther
    mag= c(TRUE,TRUE), #Fit for both
    re= c(TRUE,TRUE), #Fit for both
    nser= c(TRUE,FALSE), #Fit for bulge
    ang= c(FALSE,TRUE), #Fit for disk
    axrat= c(FALSE,TRUE), #Fit for disk
    box= c(FALSE,FALSE) #Fit for neither
  )
)

# What parameters should be fitted in log space:

tolog=list(
  sersic=list(
    xcen= c(FALSE,FALSE),
    ycen= c(FALSE,FALSE),
    mag= c(FALSE,FALSE),
    re= c(TRUE,TRUE), #re is best fit in log space
    nser= c(TRUE,TRUE), #nser is best fit in log space
    ang= c(FALSE,FALSE),
    axrat= c(TRUE,TRUE), #axrat is best fit in log space
    box= c(FALSE,FALSE)
  )
)

# The hard interval limits to use when fitting. This is not strictly required, but without
# this we cannot ensure the sampler does not enter unallowed values like negative sizes,
# Sersic indices and axial ratios etc:

intervals=list(
  sersic=list(
    xcen=list(lim=c(0,300),lim=c(0,300)),
    ycen=list(lim=c(0,300),lim=c(0,300)),
```

```
    mag=list(lim=c(10,30),lim=c(10,30)),
    re=list(lim=c(1,100),lim=c(1,100)),
    nser=list(lim=c(0.5,20),lim=c(0.5,20)),
    ang=list(lim=c(-180,360),lim=c(-180,360)),
    axrat=list(lim=c(0.1,1),lim=c(0.1,1)),
    box=list(lim=c(-1,1),lim=c(-1,1))
  )
)

# Setup the minimal data structure we need for optimisation. See vignettes for
# more complex examples using priors, and constraints:

Data=profitSetupData(image=image, sigma=sigma, segim=segim,psf=psf,
modellist=modellist, tofit=tofit, tolog=tolog, intervals=intervals, magzero=0,
algo.func='optim', verbose=TRUE)

# This produces a fairly complex R object, but with all the bits we need for fitting,
# e.g. (notice the tolog parameteres are now logged):

Data$init

# These are the parameters we wish to fit for, and we take the initial guesses from the
# model list we provided before.

# We can test how things currently look (we get an output because we set verbose=TRUE
# earlier):

profitLikeModel(parm=Data$init, Data=Data, makeplots=TRUE)

# Let us try optim BFGS:

optimfit=optim(Data$init, profitLikeModel, method='BFGS', Data=Data,
control=list(fnscale=-1))

# The best optim BFGS fit is given by:

optimfit$par

# Check it out:

profitLikeModel(optimfit$par,Data,makeplots=TRUE,whichcomponents=list(sersic=1))
profitLikeModel(optimfit$par,Data,makeplots=TRUE,whichcomponents=list(sersic=2))
profitLikeModel(optimfit$par,Data,makeplots=TRUE,whichcomponents=list(sersic='all'))

modeloptim=profitRemakeModellist(optimfit$par,Data$modellist,Data$tofit,Data$tolog)$modellis
profitEllipsePlot(Data,modeloptim,pixscale=0.2,FWHM=0.5,SBlim=26)

## End(Not run)
```

`ExampleInit` *Initial 2D Sersic Model Image Parameters*

Description

Rough initial 2 component 2D Sersic model parameters for the provided model images.

Usage

```
data("ExampleInit")
```

Format

A data frame with 39 observations on the following 10 variables.

`CATAID` GAMA CATAID reference

`sersic.xcen1` x centres of the 2D Sersic profiles

`sersic.ycen1` y centres of the 2D Sersic profiles

`sersic.mag1` Total magnitudes of the 2D Sersic bulge profiles

`sersic.mag2` Total magnitudes of the 2D Sersic disk profiles

`sersic.re1` Effective radii of the 2D Sersic bulge profiles

`sersic.re2` Effective radii of the 2D Sersic disk profiles

`sersic.nser1` Sersic indices of the 2D Sersic bulge profiles

`sersic.ang2` Orientation of the major axis of the disk profile in degrees

`sersic.axrat2` Axial ratios of Sersic disk profiles defined as minor-axis/major-axis

Details

These rough initial guesses of the galaxy models were derived from single Sersic fits taken from Lange et al 2015.

References

Lange R., et al, 2015, MNRAS, 447, 2603

Examples

```
data(ExampleInit)
ExampleInit[1:5,]
```

`profitAddMats` *Add together image matrices*

Description

A simple function to add together two matrices. The base matrix must be equal in size or larger than the matrix being added, and some pixels of the added matrix must fall inside the base matrix.

Usage

```
profitAddMats(matbase, matadd, addloc = c(1, 1), plot = FALSE, ...)
```

Arguments

`matbase`	The base matrix to be added onto (the output will be the same size as the base matrix).
`matadd`	The matrix to be added (this cannot be larger than the base matrix).
`addloc`	The reference ID of the corner pixel to use when adding '`matadd`'. This will be the position at which matadd[1,1] is added, i.e. the default c(1,1) means the two matrices are lined up on the bottom-left pixel when plotted as an image. This can be negative or larger than dim(matbase), which means only a subset of the '`matadd`' matrix is added to '`matbase`'.
`plot`	Logical; should a `magimage` plot of the output be generated?
`...`	Further arguments to be passed to `magimage`. Only relevant is '`plot`'=TRUE.

Details

In practice this function is a convenient low level routine that us used by `profitMakePointSource`. It is unlikely the user will use it directly.

By ProFit convention the bottom-left part of the bottom-left pixel when plotting the image matrix is c(0,0) and the top-right part of the bottom-left pixel is c(1,1), i.e. the mid-point of pixels are half integer values in x and y.

To confuse things a bit, when R plots an image of a matrix it is transposed and re-ordered vertically to how it appears if you print the matrix directly to screen, i.e. compare print(matrix(1:4,2,2)) and image(matrix(1:4,2,2)). The lowest value (1) is top-left when printed but bottom-left when displayed using image (the red pixel). Both are "correct": the issue is whether you consider the first element of a matrix to be the Cartesian x position (movement in x) or a row element (movement in y). Matrices in maths are always written top-left first where the first argument refers to row number, but images by convention are accessed in a Cartesian sense. Hence [3,4] in a maths matrix means 3 down and 4 right from the top-left, but 3 right and 4 up from the bottom-left in an image.

Value

Matrix; a matrix the same size as matbase, with a region of it added to by the values in matadd.

Author(s)

Aaron Robotham

See Also

`profitMakePointSource`

Examples

```
model = list(
sersic = list(
xcen   = c(180, 60),
ycen   = c(90, 10),
mag = c(15, 13),
re  = c(14, 5),
nser  = c(3, 10),
ang  = c(46, 80),
axrat  = c(0.4, 0.6),
box = c(0.5,-0.5)
)
)

# We can add a PSF to an image of two Sersic profiles:

magimage(profitAddMats(matbase=profitMakeModel(model, dim=c(200,200))$z,
matadd=profitMakePointSource(), addloc=c(50,150)))
```

---

profitAvailableConvolvers

Returns supported convolver types

Description

Simple utility to query the supported convolver types.

Usage

```
profitAvailableConvolvers()
```

Value

The output is a vector of strings with all the supported convolver type names. These values can be passed to `profitMakeConvolver` to create different types of convolvers.

Depending on how ProFit was compiled, it will support more or less underlying convolvers.

Author(s)

Rodrigo Tobar

See Also

profitMakeConvolver, profitHasOpenCL, profitHasFFTW,

Examples

```
profitAvailableConvolvers()
```

---

profitAvailableIntegrators

Returns supported profile integration methods

Description

Simple utility to query the supported profile integration methods.

Usage

```
profitAvailableIntegrators()
```

Value

The output is a vector of strings with all the supported integration method names. Currently, these are "brute" and "opencl", for consistency with profitAvailableConvolvers. These values are not yet passed directly to profitMakeModel, but indirectly via OpenCL environment variables.

Author(s)

Dan Taranu

See Also

profitMakeConvolver, profitHasOpenCL, profitHasFFTW,

Examples

```
profitAvailableConvolvers()
```

profitBenchmark *Benchmark profile integration and image convolution using libprofit.*

Description

This function will benchmark integration of surface brightness profiles and/or convolution of an image with a kernel (usually a point spread function) using libprofit. It returns a data frame with available integration/convolution methods, the most efficient method(s), and (optionally) more detailed results including convolution accuracy. It is called by `profitSetupData` by default and the convolver is used in `profitMakeModel`.

Usage

```
profitBenchmark(image, methods=NULL, psf=NULL,
  modellist=NULL, finesample=1L, calcregion=NULL, nbench=1,
  benchconvolution=is.matrix(psf),
  precisions=c("double"), omp_threads=1,
  openclenvs = profitGetOpenCLEnvs(make.envs = TRUE),
  reference = "brute", reusepsffft = TRUE, fft_effort=0,
  returnimages = FALSE)
```

Arguments

`image`	A matrix containing the image to benchmark convolution for. It should already be padded by half of the PSF width on either side to ensure that the convolved model can be cropped to the same size as the data. If no '`image`' is supplied, the user must supply '`imagedim`'.
`methods`	List of strings specifying which methods to test. Methods must be amongst those returned by `profitAvailableConvolvers`.
`psf`	A matrix containing the PSF image to convolve the model image with. If no PSF is supplied, the user must supply '`psfdim`'.
`modellist`	A valid ProFit modellist for profile integration; see `profitMakeModel`.
`finesample`	Integer; the factor to oversample the image by. The default of one does no oversampling.
`calcregion`	A logical matrix specifying regions of the image to avoid computing convolution for. Can make brute force convolution more efficient if it is not sparse.
`nbench`	Integer; the number of times to benchmark each method. Repeated convolutions can vary in running time for all kinds of reasons, so '`nbench`' = 10 is recommended unless using brute force convolution with very large images and/or kernels.
`benchconvolution`	Logical; whether to benchmark convolution. Requires a PSF.
`precisions`	Character; the numerical precision(s) to benchmark. Must be one of "single" or "double".
`omp_threads`	Integer; the number of OpenMP threads to use for integration/convolution.

`openclenvs` A data.frame with information on available OpenCL environments, such as that returned from `profitGetOpenCLEnvs`.

`reference` String; the method to use as the reference result for comparing the accuracy of all other methods. This comparison is not done if reference is not contain in '`methods`'.

`reusepsffft` Logical specifying whether to re-do the PSF FFT every iteration, which would be necessary if one is fitting the PSF.

`fft_effort` The effort level to compute the FFTW plan. FFTW plans can take a very long time to set up, so consider carefully before increasing beyond 0 - particularly if your padded image only has a few large prime factors.

`returnimages` Logical; whether to return the convolved image for every method.

Details

The function is mainly used to determine the most efficient method for convoling the '`image`' with the '`psf`'. In situations where the '`psf`' has much smaller dimensions than '`image`' this will pretty much always be Brute force convolution, but when the '`psf`' becomes comparable in size to the '`image`' then FFTW is usually faster. In the provided example all three are similar speed. Benchmarks are more difficult to predict when using multiple cores and/or devices.

Value

List containing:

result The benchmarking results in a data.frame; see `profitGetOpenCLEnvs` for more information on the format.

images List of resulting images for each method.

Notes

TBD.

Author(s)

Dan Taranu & Aaron Robotham

See Also

`profitAvailableConvolvers`, `profitMakeModel`, `profitSetupData`

Examples

```
## Not run:
model = list(
sersic = list(
xcen   = c(180, 60),
ycen   = c(90, 10),
mag = c(15, 13),
re  = c(14, 5),
```

```
  nser  = c(3, 10),
  ang  = c(46, 80),
  axrat  = c(0.4, 0.6),
  box = c(0.5,-0.5)
  )
  )

  psffwhm=3

  # Use OpenCL if available
  # Makes a list of available OpenCL environments optionally with double precision if all
  # devices support it

  openclenvs = profitGetOpenCLEnvs(make.envs=TRUE)
  nbench=1L

  # Try up to 5L if you're adventurous and don't mind waiting up to a minute for
  # single-threaded brute

  for(finesample in c(1L:3L))
  {
    model.image=profitMakeModel(model=model, dim=rep(200,2), finesample=finesample, returnfine
    psf=profitMakeGaussianPSF(fwhm=3*finesample,dim=rep(25*finesample + 1 - mod(finesample,2),

    # Benchmark model integration:
    bench=profitBenchmark(model.image, modellist=model, nbench=nbench, openclenvs=openclenvs,
      methods=profitAvailableIntegrators())

    #Print relevant results
    print(profitBenchmarkResultStripPointers(bench$result)[
      c("name","env_name","version","dev_name",paste0("tinms.mean_",c("single","double")))])

    # Benchmark convolution:
    bench=profitBenchmark(model.image, psf=psf, nbench=nbench, openclenvs=openclenvs,
      methods=profitAvailableConvolvers())

    #Print relevant results
    print(profitBenchmarkResultStripPointers(bench$result)[
      c("name","env_name","version","dev_name",paste0("tinms.mean_",c("single","double")))])

    # The old benchmarking method, for reference
    temp=profitBenchmarkConv(model.image, psf = psf, nbench=nbench)
  }

  ## End(Not run)
```

`profitBenchmarkConv` *Benchmark convolution of an image with a point spread function (PSF).*

Description

This function will benchmark convolution of an image with a point spread function (PSF), returning results as well as a list of data stored by `profitSetupData` for optimising calls to `profitConvolvePSF`.

Usage

```
profitBenchmarkConv(image=NULL, psf=NULL, calcregion=NULL, nbench=10,
  methods = c("Bruteconv","FFTconv","FFTWconv"), imagedim=NULL, psfdim=NULL,
  refftpsf=FALSE, fftwplan=NULL,  maxfftwplaneffort=0)
```

Arguments

`image`	A matrix containing the image to benchmark convolution for. It should already be padded by half of the PSF width on either side to ensure that the convolved model can be cropped to the same size as the data. If no '`image`' is supplied, the user must supply '`imagedim`'.
`psf`	A matrix containing the PSF image to convolve the model image with. If no PSF is supplied, the user must supply '`psfdim.`'
`calcregion`	A logical matrix specifying regions of the image to avoid computing convolution for. See `profitBruteConv` and `profitConvolvePSF` for more details.
`nbench`	Integer; the number of times to benchmark each method. Repeated convolutions can vary in running time for all kinds of reasons, so '`nbench`' >= 10 is recommended.
`methods`	List of strings specifying which methods to test. Valid methods are Bruteconv", "FFTconv", and "FFTWconv". FFTconv is rarely fastest.
`imagedim`	Vector of dimensions of the image to create, if '`image`' is not provided.
`psfdim`	Vector of dimensions of the PSF to create, if '`psf`' is not provided.
`refftpsf`	Logical specifying whether to re-do the PSF FFT every iteration, which would be necessary if one is fitting the PSF.
`fftwplan`	A pre-computed plan for FFTW to decompose the FFT, as returned by "fftwplan" (can this be linked?). It must have been computed for a transform of an image with the same dimensions as the product of all image and PSF dimensions.
`maxfftwplaneffort`	The maximum effort level to compute the FFTW plan. FFTW plans can take a very long time to set up, so consider carefully before increasing beyond 0 - particularly if your padded image only has a few large prime factors.

Details

This function does two important things. Firstly it determines which of three different convolution options will work fastest given the provided combination of '`iamge`' and '`psf`'. In situations where the '`psf`' has much smaller dimensions than '`image`' this will pretty much always be Brute force convolution, but when the '`psf`' becomes comparable in size to the '`image`' then one of the two FFT routines will often be faster. In the provided example all three are similar speed.

The second important output of this function is preparing all the structures needed for FFT convolution if using `profitConvolvePSF` and selecting wither FFT or FFTW. The Examples show a clear example of how you use this output '`fft`' list in practice.

Value

List; complex structure containing:

result A character string summarizing the benchmark results.

times A vector of average time in ms for each method.

best A list containing:

name The name of the fastest method.

time The average time in ms for the fastest method.

method A character string containing the name of the best method (one of Bruteconv, FFTconv, FFTWconv), which defaults to best[['name']]. '`method`' can be directly parsed into `profitConvolvePSF` '`options`'.

fft A list of useful items for FFT. '`fft`' can be directly parsed into `profitConvolvePSF` '`options`'., including:

fftwplan The FFTW plan.

paddim The dimensions of the zero-padded image, usually twice the input image dimensions and necessary to avoid periodicity artefacts.

padimagex The x coordinates to place the original image in; by default the bottom-left corner of the padded image.

padimagey The y coordinates to place the original image in; by default the bottom-left corner of the padded image.

cropx The x coordinates of the convolved image within the padded output image; usually in the centre.

cropy The y coordinates of the convolved image within the padded output image; usually in the centre.

fft A list of useful items relating to the PSF, including:

r The R FFT of the PSF.

w The FFTW of the PSF. Should be nearly identical to r.

x The x coordinates to place the PSF in; by default the centre of the bottom-left quadrant of the padded image.

y The y coordinates to place the PSF in; by default the centre of the bottom-left quadrant of the padded image.

Notes

`profitBruteConv` is usually the fastest method, except for very large image/PSF combinations. Similarly, FFTW is almost always faster than R's built-in FFT.

Author(s)

Dan Taranu & Aaron Robotham

See Also

profitBruteConv, profitConvolvePSF, profitMakeModel, profitSetupData

Examples

```
## Not run:
model = list(
sersic = list(
xcen   = c(180, 60),
ycen   = c(90, 10),
mag = c(15, 13),
re  = c(14, 5),
nser  = c(3, 10),
ang  = c(46, 80),
axrat  = c(0.4, 0.6),
box = c(0.5,-0.5)
)
)

model.image=profitMakeModel(model=model, dim=c(200,200))$z

psf=profitMakeGaussianPSF()

#Do some benchmarking:

temp=profitBenchmarkConv(model.image, psf=psf, nbench=1)

#Check the best:

temp$best

#And we can use all three:

magimage(profitConvolvePSF(model.image, psf, options=list(method='Bruteconv')))
magimage(profitConvolvePSF(model.image, psf, options=list(method='FFTconv', fft=temp$fft)))
magimage(profitConvolvePSF(model.image, psf, options=list(method='FFTWconv', fft=temp$fft)))

#Some benchmarking for different size PSFs:

profitBenchmarkConv(imagedim=c(200,200), psfdim=c(11,11), nbench=1)
profitBenchmarkConv(imagedim=c(200,200), psfdim=c(21,21), nbench=1)
profitBenchmarkConv(imagedim=c(200,200), psfdim=c(31,31), nbench=1)

#Note they are all very similar in speed when psfdim=21. The time for FFT and FFTW
#pretty much scales with the number of pixels in the image (regardless of PSF).

#Because of how they scale, there are some rough rules-of-thumb you can use:

#Brute force is usually faster when psfdim<=21:

profitBenchmarkConv(imagedim=c(200,200), psfdim=c(15,15), nbench=1)
```

```
#FFT is usually faster when imagedim<400 & psfdim>21 & psfdim<100:

profitBenchmarkConv(imagedim=c(200,200), psfdim=c(51,51), nbench=1)

#FFTW is usually faster when imagedim>400 & psfdim>21

profitBenchmarkConv(imagedim=c(400,400), psfdim=c(25,25), nbench=1)

## End(Not run)
```

`profitBenchmarkResultBest` *Return best integration/convolution method from a profitBenchmark result.*

Description

This function will return the best method from a benchmark result returned by `profitBenchmark`.

Usage

```
profitBenchmarkResultBest(result, precision="double")
```

Arguments

`result`	A benchmarking result returned from `profitBenchmark`.
`precision`	The desired floating-point precision; either "single" or "double" (the default).

Value

List; complex structure containing:

convolver Pointer to the best profitConvolver; see `profitMakeConvolver`.

dev_name The name of the best device.

name The name of the best method and/or OpenCL environment.

openclenv Pointer to the best OpenCL environment; see `profitOpenCLEnv`.

precision The floating point precision (from '`precision`').

time The time per operation for the best method in ms.

usecalcregion Logical; whether the optimal method uses the calcregion matrix or not; see `profitSetupData`.

Author(s)

Dan Taranu

See Also

`profitBenchmark`, `profitSetupData`

Examples

```
## Not run:
img = profitMakeGaussianPSF()
bench=profitBenchmark(img, psf=img, nbench=1L, methods=profitAvailableConvolvers())
print(profitBenchmarkResultStripPointers(bench$result)[
  c("name","env_name","version","dev_name",paste0("tinms.mean_",c("single","double")))])
best = profitBenchmarkResultBest(bench$result)
print(paste('Name:',best$name,'time:',best$time))

## End(Not run)
```

---

profitBenchmarkResultStripPointers

Return a copy of a data.frame with pointers converted to strings for easy printing

---

Description

This function will take a data.frame with external pointers (like a result from `profitBenchmark`) and convert the pointers to strings so that the can be printed without errors.

Usage

```
profitBenchmarkResultStripPointers(dataframe, colnames=as.vector(
  outer(c("env","convolver"),c("single","double"),paste,sep="_")))
```

Arguments

`dataframe`	A data.frame with external pointers, such as a benchmarking result returned from `profitBenchmark`.
`colnames`	Character or nunmeric vector of the names or indices of columns to convert.

Value

The same data.frame given in '`dataframe`' with external pointers converted to strings.

Author(s)

Dan Taranu

See Also

`profitBenchmark`, `profitGetOpenCLEnvs`

Examples

```
## Not run:
  openclenvs = profitGetOpenCLEnvs(make.envs=TRUE)
  print(profitBenchmarkResultStripPointers(openclenvs))
  img = profitMakeGaussianPSF()
  bench=profitBenchmark(img, psf=img, nbench=1L, methods=profitAvailableConvolvers())
  print(profitBenchmarkResultStripPointers(bench$result)[
    c("name","env_name","version","dev_name",paste0("tinms.mean_",c("single","double")))])

## End(Not run)
```

`profitBrokenExp` *Broken-Exponential Profile Specific Functions*

Description

Useful functions related to the broken-exponential profile. `profitCubaSersic` computes the exact 2D pixel integrals for a given broken-exponential model image. This is very slow compared to `profitMakeModel`, but it is useful for checking model creation tuning (i.e. the degree to which speed can be increased without overly harming accuracy). Tests with this function were used to tune `profitMakeModel`. `profitRadialSersic` computes the 1D radial flux intensity of the broken-exponential profile along the major axis of the profile.

Usage

```
profitCubaBrokenExp(xcen = dim[1]/2, ycen = dim[2]/2, mag = 15, h1 = 1, h2 = h1, rb
a = 1, ang = 0, axrat = 1, box = 0, dim = c(25, 25), rel.tol = 0.001, abs.tol = 1e-
plot = FALSE, ...)
profitRadialBrokenExp(r = 1, mag = 15, h1 = 1, h2 = h1, rb = h1, a = 1, ang = 0,
axrat = 1, box = 0)
```

Arguments

`xcen`	Scalar; x centre of the 2D broken-exponential profile (can be fractional pixel positions).
`ycen`	Scalar; y centre of the 2D broken-exponential profile (can be fractional pixel positions).
`r`	Vector; the radius along the major axis at which to evaluate the flux intensity.
`mag`	Scalar; total magnitude of the 2D broken-exponential profile. Converted to flux using flux=10^(-0.4*(mag-magzero)).
`h1`	Scalar; scale length of the inner broken-exponential profile.
`h2`	Scalar; scale length of the outer broken-exponential profile.
`rb`	Scalar; break (or truncation) radius of the broken-exponential profile.
`a`	Scalar; strength of transition from inner core to outer broken-exponential. Larger +ve means sharper.

ang	Scalar; the orientation of the major axis of the Sersic profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang.
axrat	Scalar; axial ratio of the Sersic profile defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.
box	Scalar; the boxiness of the Sersic profile that traces contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When box=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).
dim	Scalar; the dimensions of the image to be generated. Typically this should be c(Nx,Ny). If length 1 then the value will be replicated for both dimensions.
rel.tol	Scalar; the requested relative accuracy. Default, 0.001.
abs.tol	Scalar; the requested absolute accuracy. The algorithm stops when either the relative or the absolute accuracies are met. Default, near 1e-10.
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 function uses the Cuba package to make an accurate (but expensive) cubature integral. This function was written to test the accuracy of ProFit Core-Sersic models generated by profitMakeModel.

By ProFit convention the bottom-left part of the bottom-left pixel when plotting the image matrix is c(0,0) and the top-right part of the bottom-left pixel is c(1,1), i.e. the mid-point of pixels are half integer values in x and y.

To confuse things a bit, when R plots an image of a matrix it is transposed and re-ordered vertically to how it appears if you print the matrix directly to screen, i.e. compare print(matrix(1:4,2,2)) and image(matrix(1:4,2,2)). The lowest value (1) is top-left when printed but bottom-left when displayed using image (the red pixel). Both are "correct": the issue is whether you consider the first element of a matrix to be the Cartesian x position (movement in x) or a row element (movement in y). Matrices in maths are always written top-left first where the first argument refers to row number, but images by convention are accessed in a Cartesian sense. Hence [3,4] in a maths matrix means 3 down and 4 right from the top-left, but 3 right and 4 up from the bottom-left in an image.

Value

profitCubaBrokenExp: Matrix; contains the flux values of the specified model image. Dimensions 'dim'.

profitRadialBrokenExp: Vector; same length as input 'r', specifying the flux intensity of the profile along the major axis.

Author(s)

Aaron Robotham

References

Erwin, P., Pohlen, M., & Beckman, J. E. 2008, AJ, 135, 20

See Also

`profitMakeModel`, `profitSersic`, `profitMoffat`, `profitFerrer`, `profitKing`

Examples

```
## Not run:
magimage(profitCubaBrokenExp(axrat=0.7, ang=30, h1=6, h2=3, rb=4, a=1))

## End(Not run)
```

`profitBruteConv` *Low level brute force image convolution*

Description

A low level direct C++ implementation of brute force convolution that takes a user supplied image and point spread function (PSF) as inputs. In most situations users should make convolutions using the higher level `profitConvolvePSF` provided.

Usage

```
profitBruteConv(image, psf, calcregion=matrix(1,1,1), docalcregion=FALSE,
 plot = FALSE, ...)
```

Arguments

`image`	The image matrix to be convolved by the point spread function (PSF).
`psf`	The point spread function (PSF) image matrix that ProFit will use to convolve the image (should have odd size in both dimensions to prevent the image becoming offset.
`calcregion`	Matrix; logical image matrix the same size as the input '`image`' matrix. If '`docalcregion`'=TRUE, then pixels in '`calcregion`' that are TRUE (or 1) will have the convolution calculated, pixels with FALSE (or 0) values will be set to 0. This is included to increase computation speed in situations where only a small region of the full image contains the galaxy of interest for fitting. In this case pixels a long way from the segmentation region for the galaxy will not need to be convolved in order to calculate the correct likelihood within the segmentation.
`docalcregion`	Logical; should the '`calcregion`' logical matrix be used to define a subset of pixels to be convolved.
`plot`	Logical; should a `magimage` plot of the output be generated?
`...`	Further arguments to be passed to `magimage`. Only relevant is '`plot`'=TRUE.

Details

In the regime where one image is significantly (a factor of a few) smaller than the other image, this tends to be faster than FFT based convolution due to the lack of image padding and other overheads. PSF images tend to be only dozens of pixels and images 100s, so brute force convolution is the default convolution algorithm in ProFit. For this low level function the PSF supplied must have odd size in both dimensions or the image will become offset. To alleviate this issue a higher level function `profitConvolvePSF` is provided, that will re-interpolate the image to force the required odd sizes.

Value

Matrix; image matrix the same size as the input '`image`' matrix.

Author(s)

Aaron Robotham & Dan Taranu

See Also

`profitConvolvePSF`, `profitMakePointSource`, `profitMakeModel`

Examples

```
model = list(
sersic = list(
xcen   = c(180, 60),
ycen   = c(90, 10),
mag = c(15, 13),
re  = c(14, 5),
nser  = c(3, 10),
ang  = c(46, 80),
axrat  = c(0.4, 0.6),
box = c(0.5,-0.5)
)
)

model.image=profitMakeModel(model=model, dim=c(200,200))$z

# Without convolution:

magimage(model.image)

# With convolution:

magimage(profitBruteConv(image=model.image, psf=profitMakePointSource()))
```

`profitCheckIsPositiveInteger` *Check if a value is a positive integer*

Description

A simple convenience function to check if a value is a positive integer, which several arguments to functions need to be.

Usage

```
profitCheckIsPositiveInteger(x)
```

Arguments

`x`	Hopefully a positive integer.

Value

Returns nothing. Stops process if condition is not met.

Author(s)

Dan Taranu

Examples

```
## Not run:
profitCheckIsPositiveInteger(3L)
profitCheckIsPositiveInteger(3.1)

## End(Not run)
```

`profitClearCache` *Clears the internal cache used by libprofit/ProFit*

Description

Usually unknown to the user, ProFit (via libprofit) caches a few objects to speed up loading times in future executions. Rarely users might need to clear this cache, specially if the package is failing to load because of some cache problem.

In particular, FFTW wisdom and OpenCL compiled kernels are cached by libprofit.

Usage

```
profitClearCache()
```

Author(s)

Rodrigo Tobar

Examples

```
profitClearCache()
```

---

profitConvolve *Performs a convolution using the give convolver object*

Description

Given a convover and two images (the source and the kernel), this method performs a convolution of the two images and returns the result. Convolution is performed using the give convolver, which must be created using `profitMakeConvolver`.

Usage

```
profitConvolve(convolver, image, kernel, mask = NULL)
```

Arguments

`convolver`	The convolver object that will perform the convolution. It must be created using `profitMakeConvolver`.
`image`	The source image to convolve.
`kernel`	The kernel user to convolve the image with.
`mask`	Logical; If not `NULL`, it has the same size of the source image, and indicates the pixels of the resulting image that should be part of the output (if `TRUE`), or left as `0` (if `FALSE`).

Details

...

Value

The output is the result of the convolution of the image and the kernel.

Author(s)

Rodrigo Tobar

See Also

`profitBruteConv`, `profitMakeConvolver`, `profitHasFFTW`

Examples

```
# Initial images
image = matrix(1, 100, 100)
psf = matrix(1:10000, 100, 100)

# Check for FFTW support and create a convolver
type = "brute"
if (profitHasFFTW()) {
type = "fft"
}
convolver = profitMakeConvolver(type, c(100, 100), psf, fft_effort=0,
                                 omp_threads=2)

# Perform convolution
image = profitConvolve(convolver, image, psf)
```

---

profitConvolvePSF *High level image convolution*

Description

A high level interface to fast convolution that takes a user supplied image and point spread function (PSF) as inputs. This routine calls lower level functions like `profitBruteConv` and also implements FFT-based convolution using either R's built-in FFT or the 'fftw' interface to the FFTW library (the latter is usually significantly faster).

Usage

```
profitConvolvePSF(image, psf, calcregion, docalcregion=FALSE,
  options=list(method="Bruteconv"), sky = 0, plot = FALSE, ...)
```

Arguments

`image`	The image matrix to be convolved by the point spread function (PSF).
`psf`	The point spread function (PSF) image matrix that ProFit will use to convolve the image. This can have odd sizes in each dimension. If the dimension has an even size then the function will internally interpolate it onto an odd sized grid 1 element larger. The PSF will be automatically rescaled so it sums to 1 before convolution to ensure flux conservation in the model.
`options`	Additional options for model convolution parsed as a list. option$method inputs allowed are Bruteconv (brute force convolution), FFTconv (FFT convolution using the R `fft` function) and FFTWconv (FFT using the FFTW library). If using FFTconv or FFTWconv you will also need to supply a fft list. In practice this is one of the list outputs of `profitBenchmarkConv` (see Examples).
`calcregion`	Logical matrix; logical image matrix the same size as the input '`image`' matrix. If '`docalcregion`'=TRUE, then pixels in '`calcregion`' that are TRUE (or 1) will have the convolution calculated, pixels with FALSE (or 0) values will be

	set to 0. This is included to increase computation speed in situations where only a small region of the full image contains the galaxy of interest for fitting. In this case pixels a long way from the segmentation region for the galaxy will not need to be convolved in order to calculate the correct likelihood within the segmentation.
docalcregion	Logical; should the 'calcregion' logical matrix be used to define a subset of pixels to be convolved.
sky	Numeric scalar; the sky level of the image. This is important to ensure the convolution works well at the edges, since the padded regions outside the image bounds will be effectively set to the 'sky' value. If this is much higher of lower than the true sky then you may see artefacts.
plot	Logical; should a magimage plot of the output be generated?
...	Further arguments to be passed to magimage. Only relevant is 'plot'=TRUE.

Details

In the regime where one image is significantly (a factor of a few) smaller than the other image, this tends to be faster than FFT based convolution due to the lack of image padding and other overheads. PSF images tend to be only dozens of pixels and images 100s, so brute force convolution is the standard approach used in ProFit. This function offers a convenient high level interface to link{profitBruteConv}, which can only accept odd size dimensions for the PSF.

Value

Matrix; convolved image matrix the same size as the input 'image' matrix.

Author(s)

Aaron Robotham & Dan Taranu

See Also

profitBruteConv, profitMakePointSource, profitBenchmarkConv

Examples

```
model = list(
sersic = list(
xcen   = c(180, 60),
ycen   = c(90, 10),
mag = c(15, 13),
re  = c(14, 5),
nser  = c(3, 10),
ang  = c(46, 80),
axrat  = c(0.4, 0.6),
box = c(0.5,-0.5)
)
)

model.image=profitMakeModel(model=model, dim=c(200,200))$z
```

```
psf=profitMakeGaussianPSF()

#Do some benchmarking:

temp=profitBenchmarkConv(model.image, psf=psf)

#Check the best:

temp$best

#And we can use all three:

magimage(profitConvolvePSF(model.image, psf, options=list(method='Bruteconv')))
magimage(profitConvolvePSF(model.image, psf, options=list(method='FFTconv', fft=temp$fft)))
magimage(profitConvolvePSF(model.image, psf, options=list(method='FFTWconv', fft=temp$fft)))
```

`profitCoreSersic` *Core-Sersic Profile Specific Functions*

Description

Useful functions related to the Core-Sersic profile. `profitCubaCoreSersic` computes the exact 2D pixel integrals for a given Core-Sersic model image. This is very slow compared to `profitMakeModel`, but it is useful for checking model creation tuning (i.e. the degree to which speed can be increased without overly harming accuracy). Tests with this function were used to tune `profitMakeModel`. `profitRadialCoreSersic` computes the 1D radial flux intensity of the Core-Sersic profile along the major axis of the profile.

Usage

```
profitCubaCoreSersic(xcen = dim[1]/2, ycen = dim[2]/2, mag = 15, re = 1, rb = 1,
nser = 4,a = 1, b = 1, ang = 0, axrat = 1, box = 0, dim = c(25, 25), rel.tol = 0.00
abs.tol = 1e-10, plot = FALSE, ...)
profitRadialCoreSersic(r = 1, mag = 15, re = 1, rb = 1, nser = 4, a = 1, b = 1, ang
axrat = 1, box = 0)
```

Arguments

`xcen`	Scalar; x centre of the 2D Core-Sersic profile (can be fractional pixel positions).
`ycen`	Scalar; y centre of the 2D Core-Sersic profile (can be fractional pixel positions).
`r`	Vector; the radius along the major axis at which to evaluate the flux intensity.
`mag`	Scalar; total magnitude of the 2D Core-Sersic profile. Converted to flux using flux=10^(-0.4*(mag-magzero)).
`re`	Scalar; effective radius of the Sersic component of the Core-Sersic profile.
`rb`	Scalar; transition radius of the Core-Sersic profile (from inner power-law to outer Sersic).

nser	Scalar; Sersic index of the Core-Sersic profile.
a	Scalar; strength of transition from inner core to outer Sersic. Larger +ve means sharper.
b	Scalar; the inner power-law of the Core-Sersic profile. Less than 1 is an increasingly flat core.
ang	Scalar; the orientation of the major axis of the Sersic profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang.
axrat	Scalar; axial ratio of the Sersic profile defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.
box	Scalar; the boxiness of the Sersic profile that traces contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When box=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).
dim	Scalar; the dimensions of the image to be generated. Typically this should be c(Nx,Ny). If length 1 then the value will be replicated for both dimensions.
rel.tol	Scalar; the requested relative accuracy. Default, 0.001.
abs.tol	Scalar; the requested absolute accuracy. The algorithm stops when either the relative or the absolute accuracies are met. Default, near 1e-10.
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 function uses the Cuba package to make an accurate (but expensive) cubature integral. This function was written to test the accuracy of ProFit Core-Sersic models generated by profitMakeModel.

By ProFit convention the bottom-left part of the bottom-left pixel when plotting the image matrix is c(0,0) and the top-right part of the bottom-left pixel is c(1,1), i.e. the mid-point of pixels are half integer values in x and y.

To confuse things a bit, when R plots an image of a matrix it is transposed and re-ordered vertically to how it appears if you print the matrix directly to screen, i.e. compare print(matrix(1:4,2,2)) and image(matrix(1:4,2,2)). The lowest value (1) is top-left when printed but bottom-left when displayed using image (the red pixel). Both are "correct": the issue is whether you consider the first element of a matrix to be the Cartesian x position (movement in x) or a row element (movement in y). Matrices in maths are always written top-left first where the first argument refers to row number, but images by convention are accessed in a Cartesian sense. Hence [3,4] in a maths matrix means 3 down and 4 right from the top-left, but 3 right and 4 up from the bottom-left in an image.

Value

profitCubaCoreSersic: Matrix; contains the flux values of the specified model image. Dimensions 'dim'.

profitRadialCoreSersic: Vector; same length as input 'r', specifying the flux intensity of the profile along the major axis.

Author(s)

Aaron Robotham

References

Graham A. W., Erwin P., Trujillo I., Asensio Ramos A., 2003, AJ, 125, 2951

See Also

`profitMakeModel`, `profitSersic`, `profitMoffat`, `profitFerrer`, `profitKing`

Examples

```
## Not run:
magimage(profitCubaCoreSersic(axrat=0.7, ang=30))

## End(Not run)
```

---

profitDataBenchmark

Setup ProFit Data Benchmarks

Description

This is a utility function to setup benchmarks for an object of class profit.data previously set up by `profitSetupData`. This is called internally by profitSetupData. but may be called again in order to re-do benchmarking.

Usage

```
profitDataBenchmark(modellist, calcregion, imgdim,
  finesample=1L, psf=NULL, fitpsf=FALSE, omp_threads=NULL, openclenv=NULL,
  openclenv_int=openclenv, openclenv_conv=openclenv,
  nbenchmark=0L, nbenchint=nbenchmark, nbenchconv=nbenchmark,
  benchintmethods=c("brute"), benchconvmethods = c("brute","fftw"),
  benchprecisions="double", benchconvprecisions=benchprecisions,
  benchintprecisions=benchprecisions,
  benchopenclenvs = profitGetOpenCLEnvs(make.envs = TRUE),
  printbenchmark=FALSE, printbenchint=printbenchmark, printbenchconv=printbenchmark
```

Arguments

`modellist`	List; required, the initial model list that describes the analytic model to be created. Can contain an analytical PSF model as well. See Details.
`calcregion`	Logical matrix; optional, specifying the parts of the image to be used for fitting. If provided this matrix *must* be the same dimensions as '`imgdim`'. Can be integer 1/0 or boolean TRUE/FALSE type logic.

`imgdim` Numeric; the dimensions of the image to create using the '`modellist`'. These dimensions will be padded by the dimensions of the '`psf`' (if any).

`finesample` An integer factor to determine how much finer of a grid the model image and PSF should be evaluated on. Because the PSF is discretized, convolution introduces additional discretization of the model, diminishing the accuracy of the convolved model. If this parameter is set to an integer greater than one, the model and PSF (but see '`psffinesampled`') will be upsampled prior to convolution, and then downsampled after convolution. The fine sampling factor must be an integer to avoid non-integral re-binning artefacts when downsampling. Large finesample factors will significantly increase convolution time and accuracy, while moderately increasing model generation time and accuracy, so it is recommended to set '`nbenchmark`' to at least a few when using this option.

`psf` Matrix; optional. An empirical point spread function (PSF) image matrix that ProFit will use to convolve the image, as an alternative to defining an analytical PSF in '`modellist`'. During any convolution ProFit will force the sum of the pixels to equal 1 to ensure flux conservation during convolution of the model image.

`fitpsf` Logical; will the profit.data object be used to fit a PSF at the same time as extended sources? If so, the FFT convolution and benchmarking thereof will not reuse the FFT of the PSF (see `profitMakeConvolver`).

`omp_threads` An integer indicating the number of threads to use to evaluate radial profiles. If not given only one thread is used. '`openclenv`' has precedence over this option, so if both are given then OpenCL evaluation takes place.

`openclenv` If NULL (default) then the CPU is used to compute the profile. If '`openclenv`' is a legal pointer to a graphics card of class externalptr then that card will be used to make a GPU based model. This object can be obtained from the `profitGetOpenCLEnvs` function with the make.envs option set to TRUE. If '`openclenv`'='get' then the OpenCL environment is obtained from running `profitOpenCLEnv` with default values (which are usually reasonable).

`openclenv_int` The OpenCL environment to use for integrating profiles. Defaults to the value specified in '`openclenv`'.

`openclenv_conv` The OpenCL environment to use for PSF convolution. Defaults to the value specified in '`openclenv`'.

`nbenchmark` Integer; the number of times to benchmark the speed of the available convolution and integration methods. The results of this benchmarking are saved, along with the optimal method.

`nbenchint` Integer; the number of times to benchmark the speed of the available profile integration methods. The results of this benchmarking are saved, along with the optimal benchmarking method. Defaults to the value specified in '`nbenchmark`'.

`nbenchconv` Integer; the number of times to benchmark the speed of the available convolution methods. The results of this benchmarking are saved, along with the optimal method and any additional data required for efficient convolution (such as the FFT of the PSF, if it is not variable). Defaults to the value specified in '`nbenchmark`'.

benchintmethods
: List of strings specifying which profile integration methods to benchmark. See `profitBenchmark` for details.

benchconvmethods
: List of strings specifying which convolution methods to benchmark. See `profitBenchmark` for details.

benchprecisions
: List of floating point precisions to benchmark. Available options are "single" and "double". Defaults to "double", which should be used unless you are certain that single-precision roundoff errors are not important.

benchintprecisions
: List of floating point precisions to benchmark profile integration with. Available options are "single" and "double". Defaults to '`benchprecisions`'.

benchconvprecisions
: List of floating point precisions to benchmark convolution with. Available options are "single" and "double". Defaults to '`benchprecisions`'.

benchopenclenvs
: List of OpenCL environments to benchmark. Defaults to all available environments. The optimal environment will then be used for '`openclenvint`' and '`openclenvconv`', overriding any values set there.

printbenchmark
: Logical; flag to output a summary of benchmarking results. Default false.

printbenchint
: Logical; flag to output a summary of profile integration benchmarking results. Defaults to '`printbenchmark`'.

printbenchconv
: Logical; flag to output a summary of convolution benchmarking results. Defaults to '`printbenchmark`'.

Details

Besides being called by `profitSetupData` when benchmarking is requested, users may want to call this function to re-do benchmarks for an existing profit.data object, either when loading a saved profit.data from disk or simply to change any of the benchmark arguments.

Many of the arguments to this function are shared with `profitSetupData` for obvious reasons; the documentation for these arguments are reproduced here for convenience.

Author(s)

Dan Taranu

See Also

`profitSetupData`, `profitDataSetOptionsFromBenchmarks`, `profitBenchmark`, `profitMakeConvolver`

Examples

```
## Not run:
# Load ProFit example data

# There are 2 data source options: KiDS or SDSS (the galaxies are the same)

datasource='KiDS'

# Now we can extract out the example files we have available for fitting by checking the
# contents of the directory containing the example FITS files:

data('ExampleInit')
ExampleFiles=list.files(system.file("extdata",datasource,package="ProFit"))
ExampleIDs=unlist(strsplit(ExampleFiles[grep('fitim',ExampleFiles)],'fitim.fits'))
print(ExampleIDs)

# There are 10 example galaxies included. Here we run example 1:

useID=ExampleIDs[1]

box = c(160,160)
image = magcutout(readFITS(system.file("extdata", paste(datasource,'/',useID,'fitim.fits',se
package="ProFit"))$imDat, box = box)$image
sigma = magcutout(readFITS(system.file("extdata", paste(datasource,'/',useID,'sigma.fits',se
package="ProFit"))$imDat, box = box)$image
segim = magcutout(readFITS(system.file("extdata", paste(datasource,'/',useID,'segim.fits',se
package="ProFit"))$imDat, box = box)$image
psf = magcutout(readFITS(system.file("extdata", paste(datasource,'/',useID,'psfim.fits',sep=
package="ProFit"))$imDat, box = c(21,21))$image

# Very rough model (not meant to look too good yet):

useIDnum=as.integer(strsplit(useID,'G')[[1]][2])
useloc=which(ExampleInit$CATAID==useIDnum)

# For our initial model we treat component 1 as the putitive bulge and componet 2 as
# the putitive disk. We are going to attempt a fit where the disk is forced to have
# nser=1 and the bulge has an axial ratio of 1.

modellist=list(
  sersic=list(
    xcen= c(dim(image)[1]/2, dim(image)[1]/2),
    ycen= c(dim(image)[2]/2, dim(image)[2]/2),
    mag= c(ExampleInit$sersic.mag1[useloc], ExampleInit$sersic.mag2[useloc]),
    re= c(ExampleInit$sersic.re1[useloc], ExampleInit$sersic.re2[useloc])*
      if(datasource=='KiDS'){1}else{0.2/0.339},
    nser= c(ExampleInit$sersic.nser1[useloc], 1),  #Disk is initially nser=1
    ang= c(ExampleInit$sersic.ang2[useloc], ExampleInit$sersic.ang2[useloc]),
    axrat= c(1, ExampleInit$sersic.axrat2[useloc]),  #Bulge is initially axrat=1
    box=c(0, 0)
  )
)
```

```
# The pure model (no PSF):
magimage(profitMakeModel(modellist,dim=dim(image)))

# The original image:
magimage(image)

# The convolved model (with PSF):
magimage(profitMakeModel(modellist,dim=dim(image),psf=psf))

# What should we be fitting:

tofit=list(
  sersic=list(
    xcen= c(TRUE,NA), #We fit for xcen and tie the two together
    ycen= c(TRUE,NA), #We fit for ycen and tie the two together
    mag= c(TRUE,TRUE), #Fit for both
    re= c(TRUE,TRUE), #Fit for both
    nser= c(TRUE,FALSE), #Fit for bulge
    ang= c(FALSE,TRUE), #Fit for disk
    axrat= c(FALSE,TRUE), #Fit for disk
    box= c(FALSE,FALSE) #Fit for neither
  )
)

# What parameters should be fitted in log space:

tolog=list(
  sersic=list(
    xcen= c(FALSE,FALSE),
    ycen= c(FALSE,FALSE),
    mag= c(FALSE,FALSE),
    re= c(TRUE,TRUE), #re is best fit in log space
    nser= c(TRUE,TRUE), #nser is best fit in log space
    ang= c(FALSE,FALSE),
    axrat= c(TRUE,TRUE), #axrat is best fit in log space
    box= c(FALSE,FALSE)
  )
)

# Setup the profit.data

openclenvs = data.frame()

Data=profitSetupData(image=image, sigma=sigma, segim=segim, psf=psf,
  modellist=modellist, tofit=tofit, tolog=tolog, magzero=0, algo.func='optim', verbose=TRUE,
  nbenchmark = 1L, benchconvmethods = "brute",
  benchintmethods = "brute", benchopenclenvs = openclenvs,
  finesample=4L, printbenchmark = TRUE)

system.time(profitLikeModel(parm=Data$init, Data=Data))

benchmarks = profitDataBenchmark(modellist = Data$modellist, calcregion = Data$calcregion,
```

```
  imgdim = dim(Data$image), finesample = Data$finesample, psf = Data$psf, fitpsf = Data$fitp
  nbenchmark = 1L, benchconvmethods = profitAvailableConvolvers(),
  benchintmethods = profitAvailableIntegrators(), benchopenclenvs = openclenvs,
  printbenchmark = TRUE)

Data = profitDataSetOptionsFromBenchmarks(Data, benchmarks)

system.time(profitLikeModel(parm=Data$init, Data=Data))

## End(Not run)
```

`profitDataSetOptionsFromBenchmarks`
Setup ProFit Data Options from Benchmarks

Description

This is a utility function to set integration and convolution options for a profit.data previously set up by `profitSetupData` based on the results from a profit.benchmark generated by `profitDataBenchmark`.

Usage

```
profitDataSetOptionsFromBenchmarks(Data, benchmarks)
```

Arguments

`Data`	List; required, a object (list) of class profit.data as generated by `profitSetupData`.
`benchmarks`	List; required, a object (list) of class profit.data.benchmark as generated by `profitDataBenchmark`.

Details

Besides being called by `profitSetupData` when benchmarking is requested, users may want to call this function to re-do benchmarks for an existing profit.data object, either when loading a saved profit.data from disk or simply to change any of the benchmark arguments. This function does not perform benchmarking.

Value

List of class profit.data, with integration and convolution results set to the best-performing methods from '`benchmarks`'; all other options are unchanged from the original '`Data`'.

Author(s)

Dan Taranu

See Also

`profitSetupData`, `profitDataBenchmark`, `profitBenchmark`

Examples

```
## Not run:
# Load ProFit example data

# There are 2 data source options: KiDS or SDSS (the galaxies are the same)

datasource='KiDS'

# Now we can extract out the example files we have available for fitting by checking the
# contents of the directory containing the example FITS files:

data('ExampleInit')
ExampleFiles=list.files(system.file("extdata",datasource,package="ProFit"))
ExampleIDs=unlist(strsplit(ExampleFiles[grep('fitim',ExampleFiles)],'fitim.fits'))
print(ExampleIDs)

# There are 10 example galaxies included. Here we run example 1:

useID=ExampleIDs[1]

box = c(160,160)
image = magcutout(readFITS(system.file("extdata", paste(datasource,'/',useID,'fitim.fits',se
package="ProFit"))$imDat, box = box)$image
sigma = magcutout(readFITS(system.file("extdata", paste(datasource,'/',useID,'sigma.fits',se
package="ProFit"))$imDat, box = box)$image
segim = magcutout(readFITS(system.file("extdata", paste(datasource,'/',useID,'segim.fits',se
package="ProFit"))$imDat, box = box)$image
psf = magcutout(readFITS(system.file("extdata", paste(datasource,'/',useID,'psfim.fits',sep=
package="ProFit"))$imDat, box = c(21,21))$image

# Very rough model (not meant to look too good yet):

useIDnum=as.integer(strsplit(useID,'G')[[1]][2])
useloc=which(ExampleInit$CATAID==useIDnum)

# For our initial model we treat component 1 as the putitive bulge and componet 2 as
# the putitive disk. We are going to attempt a fit where the disk is forced to have
# nser=1 and the bulge has an axial ratio of 1.

modellist=list(
  sersic=list(
    xcen= c(dim(image)[1]/2, dim(image)[1]/2),
    ycen= c(dim(image)[2]/2, dim(image)[2]/2),
    mag= c(ExampleInit$sersic.mag1[useloc], ExampleInit$sersic.mag2[useloc]),
    re= c(ExampleInit$sersic.re1[useloc], ExampleInit$sersic.re2[useloc])*
      if(datasource=='KiDS'){1}else{0.2/0.339},
    nser= c(ExampleInit$sersic.nser1[useloc], 1),  #Disk is initially nser=1
    ang= c(ExampleInit$sersic.ang2[useloc], ExampleInit$sersic.ang2[useloc]),
    axrat= c(1, ExampleInit$sersic.axrat2[useloc]),  #Bulge is initially axrat=1
    box=c(0, 0)
  )
)
```

```
# The pure model (no PSF):
magimage(profitMakeModel(modellist,dim=dim(image)))

# The original image:
magimage(image)

# The convolved model (with PSF):
magimage(profitMakeModel(modellist,dim=dim(image),psf=psf))

# What should we be fitting:

tofit=list(
  sersic=list(
    xcen= c(TRUE,NA), #We fit for xcen and tie the two together
    ycen= c(TRUE,NA), #We fit for ycen and tie the two together
    mag= c(TRUE,TRUE), #Fit for both
    re= c(TRUE,TRUE), #Fit for both
    nser= c(TRUE,FALSE), #Fit for bulge
    ang= c(FALSE,TRUE), #Fit for disk
    axrat= c(FALSE,TRUE), #Fit for disk
    box= c(FALSE,FALSE) #Fit for neither
  )
)

# What parameters should be fitted in log space:

tolog=list(
  sersic=list(
    xcen= c(FALSE,FALSE),
    ycen= c(FALSE,FALSE),
    mag= c(FALSE,FALSE),
    re= c(TRUE,TRUE), #re is best fit in log space
    nser= c(TRUE,TRUE), #nser is best fit in log space
    ang= c(FALSE,FALSE),
    axrat= c(TRUE,TRUE), #axrat is best fit in log space
    box= c(FALSE,FALSE)
  )
)

# Setup the profit.data

openclenvs = data.frame()

Data=profitSetupData(image=image, sigma=sigma, segim=segim, psf=psf,
  modellist=modellist, tofit=tofit, tolog=tolog, magzero=0, algo.func='optim', verbose=TRUE,
  nbenchmark = 1L, benchconvmethods = "brute",
  benchintmethods = "brute", benchopenclenvs = openclenvs,
  finesample=4L, printbenchmark = TRUE)

system.time(profitLikeModel(parm=Data$init, Data=Data))

benchmarks = profitDataBenchmark(modellist = Data$modellist, calcregion = Data$calcregion,
```

```
  imgdim = dim(Data$image), finesample = Data$finesample, psf = Data$psf, fitpsf = Data$fitp
  nbenchmark = 1L, benchconvmethods = profitAvailableConvolvers(),
  benchintmethods = profitAvailableIntegrators(), benchopenclenvs = openclenvs,
  printbenchmark = TRUE)

Data = profitDataSetOptionsFromBenchmarks(Data, benchmarks)

system.time(profitLikeModel(parm=Data$init, Data=Data))

## End(Not run)
```

`profitDeprojectImageEllipse`
Deproject an image along an ellipse's minor axis

Description

A utility function to deproject an image with a projected circular source such as a thin disk

Usage

```
profitDeprojectImageEllipse(image, xcen, ycen, axrat, ang, upsample=5L)
```

Arguments

`image`	List or numeric; required. An image matrix or a list of image matrices, each of which will be deprojected. Every image must have the same dimensions as the first image, or have both dimensions be '`upsample`' times larger (e.g. for a finely sampled model).
`xcen`	Numeric; required. The x-coordinate in pixels of the ellipse centre in the images.
`ycen`	Numeric; required. The y-coordinate in pixels of the ellipse centre in the images.
`axrat`	Numeric; required. The axis ratio of the ellipse.
`ang`	Numeric; required. The position angle of the ellipse in degrees, following `profitMakeModel` convention of up=0.
`upsample`	Integer; optional. The factor by which to upsample each image. Must be positive.

Details

This function deprojects images, assuming that the object forms an ellipse in the image plane because it is a projection of a thin disk. Each provided image is oversampled and then resampled by stretching by 1/axrat along the ellipse minor axis. The value in each oversampled subpixel is assigned to whichever new pixel the centre of the subpixel happens to fall in, so discreteness artefacts will appear (especially for small values of '`upsample`'). It can be used on images, masks and/or binary segmentation maps, and is useful for visual inspection of disk galaxy features like spiral arms.

Value

List; deprojected versions of all of the images provided in the original list ('image').

Author(s)

Dan Taranu

See Also

profitMakeModel

Examples

```
## Not run:
disk = profitMakeModel(modellist=list(sersic=list(xcen=50,ycen=50,mag=15,re=5,nser=1,
axrat=0.5,ang=125,box=0)))
magimage(log10(disk$z), zlim=c(-15,-7.5),magmap=FALSE)

deproj = profitDeprojectImageEllipse(disk$z, xcen=50, ycen=50, axrat=0.5, ang=125,
upsample = 9L)
magimage(log10(deproj$img), zlim=c(-15,-7.5),magmap=FALSE)

## End(Not run)
```

---

profitEllipse *Measure Isophotal Flux for Pseudo-Ellipses*

Description

In the world of galaxy fitting, projected 1D flux intensity (or surface brightness) plots are popular. This function implements the low level functionality of deprojecting image pixels given a set of geometrical parameters. In simple terms this means ellipses are expanded back up to circles. We use the term pseudo-ellipse since we can also account for boxiness distortion (if desired).

Usage

```
profitEllipse(x, y, flux, xcen = 0, ycen = 0, ang = 0, axrat = 1, box = 0)
```

Arguments

x	Either a vector of image pixel midpoints (given in the usual ProFit standard, where pixel mid-points are half-integer), or an image matrix, which is then used for 'x', 'y' and 'flux'.
y	Vector; image pixel midpoints (given in the usual ProFit standard, where pixel mid-points are half-integer). Not required if an image matrix is being parsed to 'x'.
flux	Vector; image pixel fluxes. Not required if an image matrix is being parsed to 'x'.

xcen	Scalar; x centre of the 2D Sersic profile (can be fractional pixel positions).
ycen	Scalar; y centre of the 2D Sersic profile (can be fractional pixel positions).
ang	Scalar; the orientation of the major axis of the Sersic profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang.
axrat	Scalar; axial ratio of the Sersic profile defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.
box	Scalar; the boxiness of the Sersic profile that traces contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When box=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).

Details

This function mostly exists for usage by the higher level `profitEllipsePlot` function. However, the outputs might well of interest if users want to create their own bespoke 1D plotting outputs.

Value

A two column matrix containing columns rad (the effective projected major axis radius for the pixel) and flux (the un-corrected flux for this pixel).

Note

Projecting data back to 1D profiles is a knowingly imperfect process, but it can be useful when exploring the data and for producing explanatory plots. In most use cases the data error and systematics will probably dominate the process, but be cautious in overly relying on these outputs- the 2D fit residuals from `profitMakePlots` contains more information. Where the 1D and 2D plots appear to be in conflict re the quality of the fit, the latter should always take precedence.

Author(s)

Aaron Robotham

References

As noted above, the pixel flux is not corrected for distortion. This is rarely an issue, but for highly elliptical galaxies where the gradient changes radically through the pixel along the minor versus major axis the isophotal contour will be biased. In practice this bias will be the same as for the model, so pixel to pixel comparisons are still fairly valid. If this level of inconsistency annoys you, then you almost certainly should not be attempting to make deprojected 1D plots of PSF convolved bulge-disk systems. This is because once a pure elliptical disk has been convolved with a PSF it is not strictly possible to project this to circular annuli (i.e. the pseudo major-axis output we desire) using elliptical isophotes. C'est la vie.

By ProFit convention the bottom-left part of the bottom-left pixel when plotting the image matrix is c(0,0) and the top-right part of the bottom-left pixel is c(1,1), i.e. the mid-point of pixels are half integer values in x and y.

To confuse things a bit, when R plots an image of a matrix it is transposed and re-ordered vertically to how it appears if you print the matrix directly to screen, i.e. compare print(matrix(1:4,2,2)) and image(matrix(1:4,2,2)). The lowest value (1) is top-left when printed but bottom-left when displayed using image (the red pixel). Both are "correct": the issue is whether you consider the first element of a matrix to be the Cartesian x position (movement in x) or a row element (movement in y). Matrices in maths are always written top-left first where the first argument refers to row number, but images by convention are accessed in a Cartesian sense. Hence [3,4] in a maths matrix means 3 down and 4 right from the top-left, but 3 right and 4 up from the bottom-left in an image.

See Also

`profitEllipsePlot`, `profitRemakeModellist`

Examples

```
#The rough best fit model for G266033 (KiDS)

model=list(
  sersic=list(
    xcen = c(65.60642, 65.60642),
    ycen = c(78.6091, 78.6091),
    mag = c(18.49816, 16.97754),
    re = c(1.69112, 14.75940),
    nser = c(1.053961, 1),
    ang = c(39.53360, 35.02479),
    axrat = c(1, 0.5990869),
    box = c(0,0)
  )
)

data('ExampleInit')
image = readFITS(system.file("extdata", 'KiDS/G266033fitim.fits',package="ProFit"))$imDat

temp=profitEllipse(x=image, xcen=model$sersic$xcen[2], ycen=model$sersic$ycen[2], ang=
  model$sersic$ang[2], axrat=model$sersic$axrat[2], box=model$sersic$box[2])

#A rough deprojected ellipse plot:

magplot(temp, type='l', log='y', xlim=c(0,50), ylim=c(1e-12,2e-9), xlab='Pixels',
 ylab='Pixel Flux')

#Notice in the example that the core looks noisier, this is because we are deprojecting
#the bulge using the disk parameters. We can do the same using the bulge:

temp2=profitEllipse(x=image, xcen=model$sersic$xcen[1], ycen=model$sersic$ycen[1], ang=
  model$sersic$ang[1], axrat=model$sersic$axrat[1], box=model$sersic$box[1])
```

```
magplot(temp2, type='l', log='y', xlim=c(0,50), ylim=c(1e-12,2e-9), xlab='Pixels',
 ylab='Pixel Flux')

#The inner region (<5 pixels) is better deprojected using the bulge parameters.

#There is no simple way to meaningfully deproject such systems, sincew there will always
#be regions that are an even mix of bulge and disk flux, and these require different
#deprojection parameters. This is even true with IRAF Ellipse style ring fitting. The
#collapsing of data in this manner is an inherently lossy process!
```

`profitEllipsePlot` *Plot Isophotal Surface Brightness for Pseudo-Ellipses*

Description

In the world of galaxy fitting, projected 1D flux intensity (or surface brightness) plots are popular. This function implements the high level functionality of deprojecting image pixels given a set of geometrical parameters and making a 1D surface brightness plot. In simple terms this means ellipses are expanded back up to circles. We use the term pseudo-ellipse since we can also account for boxiness distortion (if desired).

Usage

```
profitEllipsePlot(Data, modellist, bulgeloc = 1, diskloc = 2, pixscale = 1, FWHM =
SBlim = 26, df = 100, raw = FALSE)
```

Arguments

`Data`	Data of class profit.data. This must be generated by the `profitSetupData` function.
`modellist`	Model list (see `profitMakeModel` for details). To aid interpreting the results of fits the user can access `profitRemakeModellist`, where an initial model is adjusted to include new parameter values and is remade into a new model.
`bulgeloc`	Location ID of bulge component in the Sersic list provided in '`modellist`'.
`diskloc`	Location ID of disk component in the Sersic list provided in '`modellist`'.
`pixscale`	The pixel scale, where pixscale=asec/pix (e.g. 0.4 for SDSS). If set to 1, then the plot is output in terms of pixels, otherwise it is rescaled to be in arcseconds.
`FWHM`	The full width half max of the PSF in units of arc seconds. A vertical line is drawn at half this number (since we are plotting radius). The fits inside of the region is inherently hard because it is well within the PSF convolution kernel.
`SBlim`	5 sigma surface brightness limit of the data. The data will plot to '`SBlim`'+1. This should be provided in terms of how the '`pixscale`' is defined, e.g. it should either be per asec^2 (when '`pixscale`'!=1) or per pix^2 (when '`pixscale`'=1).
`df`	Degrees of freedom to use for spline fitting. Lower if the lines look too wavy.

raw — Logical; if FALSE (the default) then a smooth spline is used to represent the data and model 1D profiles. This smooths out deprojection noise caused by the PSF often being non-smooth. If TRUE then the raw pixel surface brightness values are shown. These will show much more scatter, but the trends ought to be very similar. If the raw and smooth 1D plots differ significantly then the 'df' flag probably needs to be changed to improve the smoothing. Notice that when the raw pixel values are plotted the shaded error polygon is very hard to see (it is usually subdominant compared to the pixel scatter created during deprojection that has both the Normal pixel error and the PSF induced deprojection error).

Details

A word of warning: 1D surface brightness plots hide many sins, and interpreting is dark and non-rigorous art. The manner of the lossy information compression the data and model undergo is such that you should not draw strong conclusions about features in the residuals.

The pixel flux is not corrected for distortion. This is rarely an issue, but for highly elliptical galaxies where the gradient changes radically through the pixel along the minor versus major axis the isophotal contour will be biased. In practice this bias will be the same as for the model, so pixel to pixel comparisons are still fairly valid. If this level of inconsistency annoys you, then you almost certainly should not be attempting to make deprojected 1D plots of PSF convolved bulge-disk systems. This is because once a pure elliptical disk has been convolved with a PSF it is not strictly possible to project this to circular annuli (i.e. the pseudo major-axis output we desire) using elliptical isophotes. C'est la vie.

Value

Run for the side effect of making a nice surface brightness and surface brightness residual plot. Most of the plot features are automatic.

For reference, a scaled version of the PSF profile is plotted in purple.

Vertical dashed lines are drawn at the HWHM of the PSF and at the point where the model profile crosses the 'SBlim' value provided. Also, a horizontal dashed line is drawn at the 'SBlim' value provided.

Note

Projecting data back to 1D profiles is a knowingly imperfect process, but it can be useful when exploring the data and for producing explanatory plots. In most use cases the data error and systematics will probably dominate the process, but be cautious in overly relying on these outputs- the 2D fit residuals from profitMakePlots contains more information. Where the 1D and 2D plots appear to be in conflict re the quality of the fit, the latter should always take precedence.

Author(s)

Aaron Robotham

See Also

profitEllipse, profitRemakeModellist

Examples

```
# Here we use galaxy G266033:

image = readFITS(system.file("extdata", 'KiDS/G266033fitim.fits',package="ProFit"))$imDat
sigma = readFITS(system.file("extdata", 'KiDS/G266033sigma.fits',package="ProFit"))$imDat
segim = readFITS(system.file("extdata", 'KiDS/G266033segim.fits',package="ProFit"))$imDat
psf = readFITS(system.file("extdata", 'KiDS/G266033psfim.fits',package="ProFit"))$imDat

#The rough best-fit model for G266033 (KiDS)

modellist=list(
  sersic=list(
    xcen = c(65.60642, 65.60642),
    ycen = c(78.6091, 78.6091),
    mag = c(18.49816, 16.97754),
    re = c(1.69112, 14.75940),
    nser = c(1.053961, 1),
    ang = c(39.53360, 35.02479),
    axrat = c(1, 0.5990869),
    box = c(0,0)
  )
)

# What should we be fitting:

tofit=list(
  sersic=list(
    xcen= c(TRUE,NA), #We fit for xcen and tie the two togther
    ycen= c(TRUE,NA), #We fit for ycen and tie the two togther
    mag= c(TRUE,TRUE), #Fit for both
    re= c(TRUE,TRUE), #Fit for both
    nser= c(TRUE,FALSE), #Fit for bulge
    ang= c(FALSE,TRUE), #Fit for disk
    axrat= c(FALSE,TRUE), #Fit for disk
    box= c(FALSE,FALSE) #Fit for neither
  )
)

# What parameters should be fitted in log space:

tolog=list(
  sersic=list(
    xcen= c(FALSE,FALSE),
    ycen= c(FALSE,FALSE),
    mag= c(FALSE,FALSE),
    re= c(TRUE,TRUE), #re is best fit in log space
    nser= c(TRUE,TRUE), #nser is best fit in log space
    ang= c(FALSE,FALSE),
    axrat= c(TRUE,TRUE), #axrat is best fit in log space
    box= c(FALSE,FALSE)
  )
)
```

```
# The priors. If the parameters are to be sampled in log space (above) then the priors
# will refer to dex not linear standard deviations. Priors should be specified in their
# unlogged state- the logging is done internally.

sigmas=c(2,2,2,2,5,5,1,1,1,1,30,30,0.3,0.3,0.3,0.3)

priors=list(
  sersic=list(
    xcen=list(function(x){dnorm(x,0,sigmas[1],log=TRUE)},function(x){dnorm(x,0,sigmas[2],
    log=TRUE)}), # should have tight constraints on x and y
    ycen=list(function(x){dnorm(x,0,sigmas[3],log=TRUE)},function(x){dnorm(x,0,sigmas[4],
    log=TRUE)}), # should have tight constraints on x and y
    mag=list(function(x){dnorm(x,0,sigmas[5],log=TRUE)},function(x){dnorm(x,0,sigmas[6],
    log=TRUE)}), # 5 mag SD
    re=list(function(x){dnorm(x,0,sigmas[7],log=TRUE)},function(x){dnorm(x,0,sigmas[8],
    log=TRUE)}), # i.e. 1 dex in re is the SD
    nser=list(function(x){dnorm(x,0,sigmas[9],log=TRUE)},function(x){dnorm(x,0,sigmas[10],
    log=TRUE)}), # i.e. 1 dex in nser is the SD
    ang=list(function(x){dnorm(x,0,sigmas[11],log=TRUE)},function(x){dnorm(x,0,sigmas[12],
    log=TRUE)}), # very broad 30 deg ang SD
    axrat=list(function(x){dnorm(x,0,sigmas[13],log=TRUE)},function(x){dnorm(x,0,sigmas[14],
    log=TRUE)}), # i.e. 1 dex in axrat is the SD
    box=list(function(x){dnorm(x,0,sigmas[15],log=TRUE)},function(x){dnorm(x,0,sigmas[16],
    log=TRUE)})
  )
)

#the hard intervals should also be specified in log space if relevant:

lowers=c(0,0,0,0,10,10,0,0,-1,-1,-180,-180,-1,-1,-1,-1)
uppers=c(1e3,1e3,1e3,1e3,30,30,2,2,1.3,1.3,360,360,0,0,1,1)

intervals=list(
  sersic=list(
    xcen=list(function(x){interval(x,lowers[1],uppers[1],reflect=FALSE)},
    function(x){interval(x,lowers[2],uppers[2],reflect=FALSE)}),
    ycen=list(function(x){interval(x,lowers[3],uppers[3],reflect=FALSE)},
    function(x){interval(x,lowers[4],uppers[4],reflect=FALSE)}),
    mag=list(function(x){interval(x,lowers[5],uppers[5],reflect=FALSE)},
    function(x){interval(x,lowers[6],uppers[6],reflect=FALSE)}),
    re=list(function(x){interval(x,lowers[7],uppers[7],reflect=FALSE)},
    function(x){interval(x,lowers[8],uppers[8],reflect=FALSE)}),
    nser=list(function(x){interval(x,lowers[9],uppers[9],reflect=FALSE)},
    function(x){interval(x,lowers[10],uppers[10],reflect=FALSE)}),
    ang=list(function(x){interval(x,lowers[11],uppers[11],reflect=FALSE)},
    function(x){interval(x,lowers[12],uppers[12],reflect=FALSE)}),
    axrat=list(function(x){interval(x,lowers[13],uppers[13],reflect=FALSE)},
    function(x){interval(x,lowers[14],uppers[14],reflect=FALSE)}),
    box=list(function(x){interval(x,lowers[15],uppers[15],reflect=FALSE)},
    function(x){interval(x,lowers[16],uppers[16],reflect=FALSE)})
  )
)
```

```
# Setup the data structure we need for optimisation:

Data=profitSetupData(image=image, sigma=sigma, segim=segim, psf=psf,
modellist=modellist, tofit=tofit, tolog=tolog, priors=priors, intervals=intervals,
magzero=0, algo.func='optim', verbose=TRUE)

modelimage=profitMakeModel(Data$mode,dim=Data$imagedim)
profitMakePlots(Data$image, modelimage$z, Data$region, Data$sigma)

#The pixel scale of VST/KiDS is 0.2 asec/pix and the SBlim=26 mag/asec^2 in r-band.

profitEllipsePlot(Data, Data$modellist, pixscale=0.2, SBlim=26)

#So to get things into pixel space and pixel surface brightness:

profitEllipsePlot(Data, Data$modellist, pixscale=1, SBlim=26-5*log10(0.2))
```

`profitFerrer` *Ferrer Profile Specific Functions*

Description

This function computes the exact 2D pixel integrals for a given Ferrer model image. This is very slow compared to `profitMakeModel`, but it is useful for checking model creation tuning (i.e. the degree to which speed can be increased without overly harming accuracy). Tests with this function were used to tune `profitMakeModel`. `profitRadialFerrer` computes the 1D radial flux intensity of the Ferrer profile along the major axis of the profile.

Usage

```
profitCubaFerrer(xcen = dim[1]/2, ycen = dim[2]/2, mag = 15, rout = 3, a = 1, b = 1
ang = 0, axrat = 1, box = 0, dim = c(25, 25), rel.tol = 0.001, abs.tol = 1e-10,
plot = FALSE, ...)
profitRadialFerrer(r = 1, mag = 15, rout = 3, a = 1, b = 1, ang = 0, axrat = 1, box
```

Arguments

`xcen`	Scalar; x centre of the 2D Sersic profile (can be fractional pixel positions).
`ycen`	Scalar; y centre of the 2D Sersic profile (can be fractional pixel positions).
`r`	Vector; the radius along the major axis at which to evaluate the flux intensity.
`mag`	Scalar; total magnitude of the 2D Ferrer profile. Converted to flux using flux=10^(-0.4*(mag-magzero)).
`rout`	Scalar; the outer limit of the Ferrer profile. Beyond this radius the profile is evaluated as zero.
`a`	Scalar; the global profile power-law slope. 0 would mean a flat top, and +ve increases in intensity towards the centre.

b	Scalar; the strength of the profile truncation as it approaches 'rout'. Must be less than 2. 'b'=2 is a soft truncation, and 'b'<2 (including -ve) is increasingly sharp.
ang	Scalar; the orientation of the major axis of the Sersic profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang.
axrat	Scalar; axial ratio of the Sersic profile defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.
box	Scalar; the boxiness of the Sersic profile that traces contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When box=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).
dim	Vector; the dimensions of the image to be generated. Typically this should be c(Nx,Ny). If length 1 then the value will be replicated for both dimensions.
rel.tol	Scalar; the requested relative accuracy. Default, 0.001.
abs.tol	Scalar; the requested absolute accuracy. The algorithm stops when either the relative or the absolute accuracies are met. Default, near 1e-10.
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 function uses the Cuba package to make an accurate (but expensive) cubature integral. This function was written to test the accuracy of Ferrer models generated by profitMakeModel.

By ProFit convention the bottom-left part of the bottom-left pixel when plotting the image matrix is c(0,0) and the top-right part of the bottom-left pixel is c(1,1), i.e. the mid-point of pixels are half integer values in x and y.

To confuse things a bit, when R plots an image of a matrix it is transposed and re-ordered vertically to how it appears if you print the matrix directly to screen, i.e. compare print(matrix(1:4,2,2)) and image(matrix(1:4,2,2)). The lowest value (1) is top-left when printed but bottom-left when displayed using image (the red pixel). Both are "correct": the issue is whether you consider the first element of a matrix to be the Cartesian x position (movement in x) or a row element (movement in y). Matrices in maths are always written top-left first where the first argument refers to row number, but images by convention are accessed in a Cartesian sense. Hence [3,4] in a maths matrix means 3 down and 4 right from the top-left, but 3 right and 4 up from the bottom-left in an image.

Value

profitCubaFerrer: Matrix; contains the flux values of the specified model image. Dimensions 'dim'.

profitRadialFerrer: Vector; same length as input 'r', specifying the flux intensity of the profile along the major axis.

Author(s)

Aaron Robotham

References

Laurikainen E., Salo H., & Buta R., 2005, 362, 1319

See Also

`profitMakeModel`, `profitSersic`, `profitMoffat`, `profitCoreSersic`, `profitKing`

Examples

```
## Not run:
magimage(profitCubaFerrer(axrat=0.7, ang=30))

## End(Not run)
```

`profitFlux2Mag` *Convert between fluxes and magnitudes.*

Description

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

Usage

```
profitFlux2Mag(flux = 1, magzero = 0)
profitMag2Flux(mag = 0, magzero = 0)
profitFlux2SB(flux = 1, magzero = 0, pixscale = 1)
profitSB2Flux(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

`profitFlux2Mag`
: Returns the magnitude, where '`mag`' = -2.5 * log10('`flux`') + '`magzero`')

`profitMag2Flux`
: Returns the flux, where '`flux`' = 10^(-0.4 * ('`mag`' - '`magzero`'))

HERE!!!

Author(s)

Aaron Robotham

Examples

```
profitFlux2Mag(1e5, 30)
profitMag2Flux(17.5, 30)
```

`profitGetOpenCLEnvs` *Get available OpenCL environments*

Description

This function returns a data.frame with information on available OpenCL environments, which can be used to integrate profiles and/or convolve images with CPUs and GPUs and passed on to `profitBenchmark`.

Usage

```
profitGetOpenCLEnvs(name = "opencl", make.envs = FALSE)
```

Arguments

`name`	String; the name to give all of the environments. The name can be passed as the method to functions like `profitMakeConvolver`.
`make.envs`	Logical; whether to actually initialize all of the environments or simply list them.

Value

The output is a data.frame with information on every device for each available environment.

Note, if the sub-list returned by `profitOpenCLEnvInfo` has NULL devices then that openCL device will be skipped when compiling this data.frame.

Author(s)

Dan Taranu

See Also

profitBenchmark, profitMakeConvolver, profitOpenCLEnv

Examples

```
envs = profitGetOpenCLEnvs(make.envs=FALSE)
print(envs)
str(envs)
```

---

profitHasOpen *Check for presence of OpenMP, OpenCL and FFTW*

Description

Simple utilities that check whether package has compile-time OpenMP, OpenCL or FFTW support.

Usage

```
profitHasOpenMP()
profitHasOpenCL()
profitHasFFTW()
```

Value

Logical; states whether package has been installed with OpenMP, OpenCL or FFTW support, respectively.

Author(s)

Rodrigo Tobar & Aaron Robotham

See Also

profitOpenCLEnv, profitOpenCLEnvInfo

Examples

```
profitHasOpenMP()
profitHasOpenCL()
profitHasFFTW()
```

`profitInterp2d` *2D image interpolation*

Description

A low level routine to interpolate a 2D image matrix at an arbitrary x/y pixel location. This function is unlikely to be used by the user, but it used internally to ensure that point sources defined by empirical point spread functions (PSFs) are accurately generated on an image.

Usage

```
profitInterp2d(x, y, image)
```

Arguments

`x`	The x position at which to make the interpolation with respect to the x centre of '`image`'.
`y`	The x position at which to make the interpolation with respect to the x centre of '`image`'.
`image`	The image matrix to be used for the interpolation.

Details

In practice this is a low level routine unlikely to be used by the user. `profitMakePointSource` should be used to generate point sources and PSFs.

For this function (and really, it is for user ease when interpolating a PSF) [0,0] is always the R image centre of the input '`image`'. This means it would be at the usual [1.5,2] position of a 3x4 image matrix.

Value

Matrix; a three column matrix where column 1 is the requested x interpolation locations, column 2 is the requested y interpolation locations and column 3 is the interpolated values.

Author(s)

Aaron Robotham

See Also

`profitConvolvePSF`, `profitMakePointSource`

Examples

```
PSFeven=profitMakePointSource(image = matrix(0,24,24))
magimage(PSFeven)
xrange=floor(-dim(PSFeven)[1]/2):ceiling(dim(PSFeven)[1]/2)
yrange=floor(-dim(PSFeven)[2]/2):ceiling(dim(PSFeven)[2]/2)
regrid=expand.grid(xrange,yrange)
PSFodd=matrix(profitInterp2d(x=regrid[,1], y=regrid[,2], image=PSFeven)[,3],
length(xrange),length(yrange))
magimage(PSFodd)
```

---

profitKing *King Profile Specific Functions*

Description

This function computes the exact 2D pixel integrals for a given King model image. This is very slow compared to `profitMakeModel`, but it is useful for checking model creation tuning (i.e. the degree to which speed can be increased without overly harming accuracy). Tests with this function were used to tune `profitMakeModel`. `profitRadialKing` computes the 1D radial flux intensity of the King profile along the major axis of the profile.

Usage

```
profitCubaKing(xcen = dim[1]/2, ycen = dim[2]/2, mag = 15, rc = 1, rt = 3, a = 2,
ang = 0, axrat = 1, box = 0, dim = c(25, 25), rel.tol = 0.001, abs.tol = 1e-10,
plot = FALSE, ...)
profitRadialKing(r=1, mag=15, rc=1, rt=3, a=2, ang=0, axrat=1, box=0)
```

Arguments

`xcen`	Scalar; x centre of the 2D Sersic profile (can be fractional pixel positions).
`ycen`	Scalar; y centre of the 2D Sersic profile (can be fractional pixel positions).
`r`	Vector; the radius along the major axis at which to evaluate the flux intensity.
`mag`	Scalar; total magnitude of the 2D King profile. Converted to flux using flux=10^(-0.4*(mag-magzero)).
`rc`	Scalar; the core radius of the King profile.
`rt`	Scalar; the truncation radius of the King profile. Beyond this radius the profile is evaluated as zero.
`a`	Scalar, the power-law of the King profile.
`ang`	Scalar; the orientation of the major axis of the Sersic profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang.
`axrat`	Scalar; axial ratio of the Sersic profile defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.

box	Scalar; the boxiness of the Sersic profile that traces contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When box=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).
dim	Vector; the dimensions of the image to be generated. Typically this should be c(Nx,Ny). If length 1 then the value will be replicated for both dimensions.
rel.tol	Scalar; the requested relative accuracy. Default, 0.001.
abs.tol	Scalar; the requested absolute accuracy. The algorithm stops when either the relative or the absolute accuracies are met. Default, near 1e-10.
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 function uses the Cuba package to make an accurate (but expensive) cubature integral. This function was written to test the accuracy of Ferrer models generated by profitMakeModel.

By ProFit convention the bottom-left part of the bottom-left pixel when plotting the image matrix is c(0,0) and the top-right part of the bottom-left pixel is c(1,1), i.e. the mid-point of pixels are half integer values in x and y.

To confuse things a bit, when R plots an image of a matrix it is transposed and re-ordered vertically to how it appears if you print the matrix directly to screen, i.e. compare print(matrix(1:4,2,2)) and image(matrix(1:4,2,2)). The lowest value (1) is top-left when printed but bottom-left when displayed using image (the red pixel). Both are "correct": the issue is whether you consider the first element of a matrix to be the Cartesian x position (movement in x) or a row element (movement in y). Matrices in maths are always written top-left first where the first argument refers to row number, but images by convention are accessed in a Cartesian sense. Hence [3,4] in a maths matrix means 3 down and 4 right from the top-left, but 3 right and 4 up from the bottom-left in an image.

Value

profitCubaKing: Matrix; contains the flux values of the specified model image. Dimensions 'dim'.

profitRadialKing: Vector; same length as input 'r', specifying the flux intensity of the profile along the major axis.

Author(s)

Aaron Robotham

References

King I., AJ, 1962, 71, 64

See Also

profitMakeModel, profitCubaSersic, profitCubaMoffat, profitCubaCoreSersic, profitCubaFerrer

Examples

```
## Not run:
magimage(profitCubaKing(axrat=0.7, ang=30))

## End(Not run)
```

---

profitLikeModel *Calculate the log likelihood of a model given the input data*

Description

This is the work-horse log-likelihood that we can use to assess the current fit. This function becomes the input for generic R fitting codes like optim (or any that user wants to use).

Usage

```
profitLikeModel(parm, Data, makeplots = FALSE,
whichcomponents=list(sersic="all",moffat="all",ferrer="all",pointsource="all"),
rough = FALSE, cmap = rev(colorRampPalette(brewer.pal(9,"RdYlBu"))(100)),
errcmap = cmap, plotchisq = FALSE, maxsigma = 5,
model=NULL, image=Data$image, sigma=Data$sigma, region=Data$region,
like.func=Data$like.func, algo.func=Data$algo.func, verbose=Data$verbose)
```

Arguments

parm	A vector of values for the parameters being fit. These must be in the expected order for the provided model. See profitSetupData for details.
Data	Data of class profit.data. This must be generated by the profitSetupData function.
makeplots	Logical; should an image be made showing the Data, model, and residuals; see profitMakePlots for details.
whichcomponents	A list specifying which component of each profile type should be used to create the model image. This is useful if you want to visualise the appearance of e.g. Sersic components 1 and 2 separately. The default entry list=(profilename1="all",...) will show the total model with all components added. If a given profile has no entry in the list, the default is "all", i.e. one must explicitly exclude components rather than including them, and an empty list will exclude nothing; the default value just lists available profile names explicitly.

rough	Logical; should an approximate model image be created. If TRUE only one evaluation of the Sersic model is made at the centre of each pixel. If FALSE then accurate upsampling is used to create more precise pixel values. It is often useful to use rough=TRUE when you are a long way from a viable solution and you are searching for a reasonable global minimum. Once near the global minimum then rough should be set to FALSE and more precise evaluations of the fit should be made. Rough fits are often pretty good and similar to the much more expensive accurate fits, except for very steep profiles.
cmap	The colour map to use for images if 'makeplots' is TRUE; see profitMakePlots for details.
errcmap	The colour map to use for chi-square residual images if 'makeplots' is TRUE; see profitMakePlots for details.
plotchisq	Logical flag to determine if the function should plot a map and a histogram of chi squared = (((image-'model')/'error')[/'region'])^2.
maxsigma	The maximum range of sigma deviations displayed. Only relevant if 'makeplots'=TRUE.
model	Matrix, optional; a model image. This will compute the likelihood for the supplied model image instead of generating a model from 'parm' and 'Data', although the prior (if any) will still be computed using 'parm'.
image	Matrix; the image data to compare to. Defaults to 'Data'$image but can be overridden.
sigma	Matrix; the error map. Defaults to 'Data'$sigma but can be overridden.
region	Matrix; a map of good pixels to evaluate likelihoods for. Defaults to 'Data'$region but can be overridden.
like.func	Characters; the name of the likelihood function. Defaults to 'Data'$like.func but can be overridden.
algo.func	Characters; the name of the optimization function used (if any). Defaults to 'Data'$algo.func but can be overridden. This changes the return values of the function.
verbose	Logical; verbosity setting for output from other ProFit functions. Defaults to 'Data'$verbose but can be overridden.

Details

While this function is designed to produce the required outputs for different optimisation schemes (optim, LaplaceApproximation, LaplacesDemon and CMA have been used successfully) the side effect of producing the model image is quite useful for prototyping.

Value

Option dependent output, either a Scalar or a List.

profitLikeModel uses the value of Data$algo.func to determine the type of output generated (see profitSetupData for details). If this flag is set to either "optim" or "CMA" then it will output the log-likelihood as a single scalar value. If set to "LA" or "LD" then a more complex list structure as expected by LaplaceApproximation and LaplacesDemon (see details for these functions). In practice the simple log-likelihood scalar output as given by setting to "optim" or "CMA" is useful for a large number of maximisation algorithms available within R. If an empty string is given, the function will simply return the model and PSF image.

Author(s)

Aaron Robotham & Dan Taranu

See Also

profitSetupData, profitMakePlots, LaplaceApproximation, LaplacesDemon

Examples

```
# Load ProFit example data

# There are 2 data source options: KiDS or SDSS (the galaxies are the same)

datasource='KiDS'

# Now we can extract out the example files we have available for fitting by checking the
# contents of the directory containing the example FITS files:

data('ExampleInit')
ExampleFiles=list.files(system.file("extdata",datasource,package="ProFit"))
ExampleIDs=unlist(strsplit(ExampleFiles[grep('fitim',ExampleFiles)],'fitim.fits'))
print(ExampleIDs)

# There are 10 example galaxies included. Here we run example 1:

useID=ExampleIDs[1]

image = readFITS(system.file("extdata", paste(datasource,'/',useID,'fitim.fits',sep=''),
package="ProFit"))$imDat
sigma = readFITS(system.file("extdata", paste(datasource,'/',useID,'sigma.fits',sep=''),
package="ProFit"))$imDat
segim = readFITS(system.file("extdata", paste(datasource,'/',useID,'segim.fits',sep=''),
package="ProFit"))$imDat
psf = readFITS(system.file("extdata", paste(datasource,'/',useID,'psfim.fits',sep=''),
package="ProFit"))$imDat

# Very rough model (not meant to look too good yet):

useIDnum=as.integer(strsplit(useID,'G')[[1]][2])
useloc=which(ExampleInit$CATAID==useIDnum)

# For our initial model we treat component 1 as the putitive bulge and componet 2 as
# the putitive disk. We are going to attempt a fit where the disk is forced to have
# nser=1 and the bulge has an axial ratio of 1.

modellist=list(
  sersic=list(
    xcen= c(dim(image)[1]/2, dim(image)[1]/2),
    ycen= c(dim(image)[2]/2, dim(image)[2]/2),
    mag= c(ExampleInit$sersic.mag1[useloc], ExampleInit$sersic.mag2[useloc]),
    re= c(ExampleInit$sersic.re1[useloc], ExampleInit$sersic.re2[useloc])*
      if(datasource=='KiDS'){1}else{0.2/0.339},
```

```
    nser= c(ExampleInit$sersic.nser1[useloc], 1),  #Disk is initially nser=1
    ang= c(ExampleInit$sersic.ang2[useloc], ExampleInit$sersic.ang2[useloc]),
    axrat= c(1, ExampleInit$sersic.axrat2[useloc]),  #Bulge is initially axrat=1
    box=c(0, 0)
  )
)

# The pure model (no PSF):
magimage(profitMakeModel(modellist,dim=dim(image)))

# The original image:
magimage(image)

# The convolved model (with PSF):
magimage(profitMakeModel(modellist,dim=dim(image),psf=psf))

# What should we be fitting:

tofit=list(
  sersic=list(
    xcen= c(TRUE,NA), #We fit for xcen and tie the two togther
    ycen= c(TRUE,NA), #We fit for ycen and tie the two togther
    mag= c(TRUE,TRUE), #Fit for both
    re= c(TRUE,TRUE), #Fit for both
    nser= c(TRUE,FALSE), #Fit for bulge
    ang= c(FALSE,TRUE), #Fit for disk
    axrat= c(FALSE,TRUE), #Fit for disk
    box= c(FALSE,FALSE) #Fit for neither
  )
)

# What parameters should be fitted in log space:

tolog=list(
  sersic=list(
    xcen= c(FALSE,FALSE),
    ycen= c(FALSE,FALSE),
    mag= c(FALSE,FALSE),
    re= c(TRUE,TRUE), #re is best fit in log space
    nser= c(TRUE,TRUE), #nser is best fit in log space
    ang= c(FALSE,FALSE),
    axrat= c(TRUE,TRUE), #axrat is best fit in log space
    box= c(FALSE,FALSE)
  )
)

# Setup the minimal data structure we need for likelihood.

Data=profitSetupData(image=image, sigma=sigma, segim=segim, psf=psf,
modellist=modellist, tofit=tofit, tolog=tolog, magzero=0, algo.func='optim', verbose=TRUE)

# Finally, calculate the likelihood and make a plot:
```

```
profitLikeModel(parm=Data$init, Data=Data, makeplots=TRUE)
```

`profitMag2Mu` *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

```
profitMag2Mu(mag = 15, re = 1, axrat = 1, pixscale = 1)
profitMu2Mag(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

profitMag2Mu returns the mean surface brightness within Re of the 2D Sersic profile.

profitMag2Mu returns total magnitude of the 2D Sersic profile.

Author(s)

Aaron Robotham

See Also

`profitMakeModel`

Examples

```
profitMag2Mu(mag=22, re=10, axrat=0.5)
profitMu2Mag(mu=28, re=10, axrat=0.5)
```

`profitMakeConvolver` *Creates a Convolver object.*

Description

Creates a Convolver object that can be used to perform convolution of images. Convolution can be carried out direcly via `profitConvolve` or when creating Model images with `profitMakeModel`.

This function allows users to create specific convolvers instead of letting `profitMakeModel` create a default one.

Usage

```
profitMakeConvolver(type, image_dimensions, psf,
 reuse_psf_fft = TRUE, fft_effort = 0, omp_threads = NULL, openclenv = NULL)
```

Arguments

`type`	The type of convolver to create. It should be one of the strings returned by `profitAvailableConvolvers`
`image_dimensions`	Dimensions of the images that will be convolved by this convolver.
`psf`	The point spread function (PSF) image matrix that will be used by this Convolver.
`reuse_psf_fft`	Logical; whether the FFT-ed version of the PSF used by the Convolver should be re-used across executions of the convolution. This is useful if the convolver will be re-used to convolve different images (of the same size) with the same PSF. Used only if '`type`' is `"fft"` and ProFit has FFTW support.
`fft_effort`	Amount of effort to spend creating the FFTW plans used by this Convolver. Accepted values range from 0 to 3, and map to the ESTIMATE, MEASURE, PATIENT and EXHAUSTIVE FFTW efforts, respectively. Used only if '`type`' is `"fft"` and ProFit has FFTW support.
`omp_threads`	Specifies the number of OpenMP threads to use to execute the underlying FFTW plans. Used only if '`type`' is `"fft"` and ProFit has FFTW support, OpenMP support, and the underlying FFTW library has OpenMP support.
`openclenv`	A valid pointer to an OpenCL environment (obtained from the `profitOpenCLEnv`). Used only if '`type`' is `"opencl"` or `"opencl-local"` and ProFit has OpenCL support.

Details

A convolver object can be used to perform one or more image convolutions. Depending on the convolver's requested configuration, it could be expensive to create them. Users should thus try to keep a hold on these objects.

Value

The output is an external pointer of class 'externalptr' to be passed to `profitMakeModel` via its convopt list option, or to be used to convolve images directly via `profitConvolve`

Author(s)

Rodrigo Tobar

See Also

`profitAvailableConvolvers`, `profitConvolve`, `profitBruteConv`, `profitMakePointSource`, `profitBenchmarkConv`, `profitHasFFTW` `profitOpenCLEnv`

Examples

```
## Not run:
psf = profitMakeGaussianPSF(dim=c(100,100))

has_openCL=profitHasOpenCL()
has_fft = profitHasFFTW()
has_openMP=profitHasOpenMP()

convolver_brute = profitMakeConvolver("brute", c(400, 400), psf)

if(has_openCL){
  convolver_bruteCL = profitMakeConvolver("opencl", c(400, 400), psf,
  openclenv=profitOpenCLEnv())
}

if(has_fft){
convolver_fft = profitMakeConvolver("fft", c(400, 400), psf, fft_effort=1,
omp_threads=1)
}

if(has_fft & has_openMP){
convolver_fftMP = profitMakeConvolver("fft", c(400, 400), psf, fft_effort=1,
omp_threads=4)
}

model = list(
  sersic = list(
    xcen   = c(80, 210),
    ycen   = c(190, 50),
    mag = c(15, 13),
    re  = c(14, 5),
    nser  = c(3, 10),
    ang  = c(46, 80),
    axrat  = c(0.4, 0.6),
    box = c(0.5,-0.5)
  )
)
```

```
system.time(for(i in 1:10){image_brute=profitMakeModel(model=model, dim=c(300,300), psf=psf,
  convopt=list(convolver=convolver_brute))$z})

if(has_openCL){
system.time(for(i in 1:10){image_bruteCL=profitMakeModel(model=model, dim=c(300,300), psf=ps
  convopt=list(convolver=convolver_bruteCL))$z})
}

if(has_fft){
system.time(for(i in 1:10){image_fft=profitMakeModel(model=model, dim=c(300,300), psf=psf,
  convopt=list(convolver=convolver_fft))$z})
}

if(has_fft & has_openMP){
system.time(for(i in 1:10){image_fftMP=profitMakeModel(model=model, dim=c(300,300), psf=psf,
  convopt=list(convolver=convolver_fftMP))$z})
}

magimage(image_brute)

if(has_openCL){
  magimage(image_bruteCL)
  magimage(image_brute-image_bruteCL)
}

if(has_fft){
  magimage(image_fft)
  magimage(image_brute-image_fft)
}

if(has_fft & has_openMP){
  magimage(image_fftMP)
  magimage(image_brute-image_fftMP)
}

## End(Not run)
```

`profitMakeGaussianPSF` *Make a 2D Gaussian PSF (point source profile or point spread function)*

Description

Creates an analytic 2D Gaussian PSF with a given full-width at half-maximum.

Usage

```
profitMakeGaussianPSF(fwhm = 3, dim = c(25,25), trim = 1 - pi/4, plot = FALSE, ...)
```

Arguments

fwhm	Numeric scalar; the full width half max (FWHM) of the desired PSF. This is internally converted to a Gaussian standard deviation (sigma) using sigma=FWHM/(2*sqrt(2*log(2)))~FWH
dim	Integer vector; the dimensions of the image to be generated. Typically this should be c(Nx,Ny). If length 1 then the value will be replicated for both dimensions.
trim	Numeric scalar; fraction of pixels to keep. This is done by using quantile to find the pixel value and setting pixels below this to zero. This is done to obtain a more circular kernel (often handy), where the defaults will approximately fill a square image with a circle of diameter 'dim'.
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 function to create a Gaussian PSF for prototyping image convolution/fits in cases where PSF has not been estimated. In general this should *not* be used for final fitting, since it is rare to have an exact, circular Gaussian profile PSFs in real astronomical images. Better options would be a double winged Gaussian, a Moffat (which is similar to a 2D Student-T distribution with no correlation), or an empirical PSF.

Value

Numeric matrix; the 2D image of the specified PSF with dimensions c(npix,npix).

Author(s)

Aaron Robotham & Dan Taranu

See Also

profitMakePointSource, profitConvolvePSF

Examples

```
#Various FWHM:

magimage(profitMakeGaussianPSF(fwhm=1), stretch='lin')
magimage(profitMakeGaussianPSF(fwhm=3), stretch='lin')
magimage(profitMakeGaussianPSF(fwhm=5), stretch='lin')
```

`profitMakeModel` *High-Level 2D Galaxy and Point Source Image Creation*

Description

Create an astronomical image containing model galaxies or point sources, with or without convolution with the PSF. This is achieved by providing a model list ('`modellist`') that contains the main parameters that define the model.

Usage

```
profitMakeModel(modellist, magzero = 0, psf=NULL, dim = c(100, 100),
whichcomponents=list(sersic = "all", moffat = "all", ferrer = "all", ferrers = "all
coresersic = "all", king = "all", brokenexp = "all", pointsource = "all"), rough =
acc = 0.1, finesample=1L, returnfine=FALSE, returncrop=TRUE, calcregion,
docalcregion=FALSE, adjust_calcregion = TRUE, magmu=FALSE, remax, rescaleflux = FAI
convopt=NULL, psfdim = c(25, 25), openclenv = NULL,
omp_threads = NULL, plot = FALSE, ...)
```

Arguments

`modellist`	The model list that describes the analytic model to be created. See Details.
`magzero`	The magnitude zero point, where values become scaled by the standard scale=10^(-0.4*(mag-magzero)).
`psf`	The PSF matrix to use for the model. This will both be used to convolve the radial profile models and to model point sources (i.e. stars). If this is left as NULL and a psf model is included in the '`modellist`', then this analytic PSF will be used instead.
`dim`	The desired dimensions of the 2D image matrix. This should be a two element vector which specifies c(width,height) in the plotted image. This becomes c(rows,columns) in the matrix itself (see Details below).
`whichcomponents`	A list specifying which component of each profile type should be used to create the model image. This is useful if you want to visualise the appearance of e.g. Sersic components 1 and 2 separately. The default entry list=(profilename1="all",...) will show the total model with all components added. If a given profile has no entry in the list, the default is "all", i.e. one must explicitly exclude components rather than including them, and an empty list will exclude nothing; the default value just lists available profile names explicitly.
`rough`	Logical; should an approximate model image be created. If TRUE only one evaluation of the Sersic model is made at the centre of each pixel. If FALSE then accurate upsampling is used to create more precise pixel values. It is often useful to use rough=TRUE when you are a long way from a viable solution and you are searching for a reasonable global minimum. Once near the global minimum then rough should be set to FALSE and more precise evaluations of the fit should be made. Rough fits are often pretty good and similar to the much more expensive accurate fits, except for very steep profiles.

`acc` Desired minimum per pixel accuracy within the upscaling region defined by 'RESWITCH'. 'ACC' specifies the allowed fractional difference from adjacent pixels before recursion is triggered. Smaller (i.e. 0.01) means more accurate integration, but increased computation time.

`finesample` Integer specifying the number of times to subdivide the model image and therefore finely sample it (compared to the dimensions specified in 'dim'), for more accurate PSF convolution. Must be one or higher. Note that the 'psf' image and 'modellist' PSF are not automatically fine sampled; this is only done by profitSetupData.

`returnfine` Logical flag to return the finely-sampled model instead of downsampling to the specified 'dim'.

`returncrop` Logical flag to return the appropriately PSF-padded 'modellist' instead of cropping to the specified 'dim'.

`calcregion` Matrix; logical image matrix the same size as the input 'image' matrix. If 'docalcregion'=TRUE, then pixels in 'calcregion' that are TRUE (or 1) will have the convolution calculated, pixels with FALSE (or 0) values will be set to 0. This is included to increase computation speed in situations where only a small region of the full image contains the galaxy of interest for fitting. In this case pixels a long way from the segmentation region for the galaxy will not need to be convolved in order to calculate the correct likelihood within the segmentation.

`docalcregion` Logical; should the 'calcregion' logical matrix be used to define a subset of pixels to be convolved.

`adjust_calcregion` Logical; indicates if the given calcregion needs to be internally adjusted (or not) in order to correctly consider the flux going outside of the image and captured by the convolution process. By default this is TRUE, but during profile fitting this option is set to TRUE because the profitSetupData procedure precalculates these adjustments.

`magmu` Logical vector. If TRUE then the mag parameter in the input 'modellist' list is interpreted as the mean surface brightness within Re in units of mag/pix^2. If this is of length 1 then all mag values will be interpreted in the same sense, otherwise it should be the same length as the number of components being generated. If FALSE mag is taken to mean total magnitude of the integrated profile. Using this flag might be useful for disk components since they occupy and relatively narrow range in surface brightness, but can have essentially any total magnitude.

`remax` If provided the profile is computed out to this many times Re, after this point the values in the image are set to zero. If missing the profile is calculated out to the radius at which 99.99% of flux is contained within the elliptical isocontour.

`rescaleflux` Logical; where the profile has been truncated via 'remax' this specifies whether the profile should be rescaled since the total integrated flux will be less than without truncation. In practice this means image values are increases by 1/0.9999 for the default case (where the profile is truncated at the point where 99.99% of flux is contained).

`convopt` A list specifying options for convolution. Currently the only named item used is convolver (which can be obtained via profitMakeConvolver; if specified, it is used to perform the convolution of the model (if convolution is required).

psfdim	The size of the PSF image to be constructed if a psf modellist is being provided to construct an analytic PSF model.
openclenv	If NULL (default) then the CPU is used to compute the profile. If 'openclenv' is a legal pointer to a graphics card of class externalptr then that card will be used to make a GPU based model. This object can be obtained from the profitOpenCLEnv function directly. If 'openclenv'='get' then the OpenCL environment is obtained from running profitOpenCLEnv with default values (which are usually reasonable).
omp_threads	An integer indicating the number of threads to use to evaluate radial profiles. If not given only one thread is used. 'openclenv' has precedence over this option, so if both are given then OpenCL evaluation takes place.
plot	Logical; should a magimage plot of the output be generated?
...	Further arguments to be passed to magimage. Only relevant is 'plot'=TRUE.

Details

A legal model list ('modellist') has the structure of list(sersic, coresersic, moffat, ferrer, king, brokenexp, pointsource, sky). At least one of sersic, coresersic, moffat, ferrer, king, brokenexp, pointsource, psf or sky should be present. Each of these is itself a list which contain vectors for each relevant parameter. All these vectors should be the same length for each type of model structure.

The parameters that must be specified for 'sersic' are:

xcen Vector; x centres of the 2D Sersic profiles (can be fractional pixel positions).

ycen Vector; y centres of the 2D Sersic profiles (can be fractional pixel positions).

mag Vector; total magnitudes of the 2D Sersic profiles. Converted to flux using 10^(-0.4*('mag'-'magzero')).

re Vector; effective radii of the 2D Sersic profiles

nser Vector; the Sersic indices of the 2D Sersic profiles

ang Vector; the orientation of the major axis of the profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang %% 180.

axrat Vector; axial ratios of Sersic profiles defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.

box Vector; the boxiness of the Sersic profiles that trace contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When 'box'=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).

The parameters that must be specified for 'coresersic' are:

xcen Vector; x centres of the 2D Core-Sersic profiles (can be fractional pixel positions).

ycen Vector; y centres of the 2D Core-Sersic profiles (can be fractional pixel positions).

mag Vector; total magnitudes of the 2D Core-Sersic profiles. Converted to flux using 10^(-0.4*('mag'-'magzero')).

re Vector; effective radius of the Sersic components of the Core-Sersic profiles.

rb Vector; transition radius of the Core-Sersic profiles (from inner power-law to outer Sersic).

nser Vector; Sersic indices of the Core-Sersic profiles.

a Vector; strength of transitions from inner cores to outer Sersics. Larger +ve means sharper.

b Vector; the inner power-law of the Core-Sersic profiles. Less than 1 is an increasingly flat core.

ang Vector; the orientation of the major axis of the profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang %% 180.

axrat Vector; axial ratios of Core-Sersic profiles defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.

box Vector; the boxiness of the Core-Sersic profiles that trace contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When 'box'=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).

The parameters that must be specified for 'moffat' are:

xcen Vector; x centres of the 2D Moffat profiles (can be fractional pixel positions).

ycen Vector; y centres of the 2D Moffat profiles (can be fractional pixel positions).

mag Vector; total magnitudes of the 2D Moffat profiles. Converted to flux using 10^(-0.4*('mag'-'magzero')).

fwhm Vector; full width half max of the Moffat function.

con Vector; concentration parameter for Moffat functions. Must be larger than 1.

ang Vector; the orientation of the major axis of the profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang %% 180.

axrat Vector; axial ratios of Moffat profiles defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.

box Vector; the boxiness of the Moffat profiles that trace contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When 'box'=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).

The parameters that must be specified for 'ferrer' or 'ferrers' (either allowed) are:

xcen Vector; x centres of the 2D Ferrer profiles (can be fractional pixel positions).

ycen Vector; y centres of the 2D Ferrer profiles (can be fractional pixel positions).

mag Vector; total magnitudes of the 2D Ferrer profiles. Converted to flux using 10^(-0.4*('mag'-'magzero')).

rout Vector; the outer limit of the Ferrer profile. Beyond this radius the profile is evaluated as zero.

a Vector; the global profile power-law slope. 0 would mean a flat top, and +ve increases in intensity towards the centre.

b Vector; the strength of the profile truncation as it approaches 'rout'.

ang Vector; the orientation of the major axis of the profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang %% 180.

axrat Vector; axial ratios of Ferrer profiles defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.

box Vector; the boxiness of the Ferrer profiles that trace contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When 'box'=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).

The parameters that must be specified for 'king' are:

xcen Vector; x centres of the 2D King profiles (can be fractional pixel positions).

ycen Vector; y centres of the 2D King profiles (can be fractional pixel positions).

mag Vector; total magnitudes of the 2D King profiles. Converted to flux using 10^(-0.4*('mag'-'magzero')).

rc Vector; the core radius of the King profile.

rt Vector; the truncation radius of the King profile. Beyond this radius the profile is evaluated as zero.

a Vector; the power-law of the King profile.

ang Vector; the orientation of the major axis of the profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang %% 180.

axrat Vector; axial ratios of King profiles defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.

box Vector; the boxiness of the King profiles that trace contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When 'box'=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).

The parameters that must be specified for 'brokenexp' are:

xcen Vector; x centres of the 2D Broken-Exponential profiles (can be fractional pixel positions).

ycen Vector; y centres of the 2D Broken-Exponential profiles (can be fractional pixel positions).

mag Vector; total magnitudes of the 2D Ferrer profiles. Converted to flux using 10^(-0.4*('mag'-'magzero')).

h1 Vector; scale length of the inner Broken-Exponential profile.

h2 Vector; scale length of the outer Broken-Exponential profile.

rb Vector; break (or truncation) radius of the Broken-Exponential profile.

a Vector; strength of transition from inner core to outer Broken-Exponential. Larger +ve means sharper.

ang Vector; the orientation of the major axis of the profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang %% 180.

axrat Vector; axial ratios of Broken-Exponential profiles defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.

box Vector; the boxiness of the Broken-Exponential profiles that trace contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When 'box'=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).

The parameters that must be specified for 'pointsource' (see profitMakePointSource for details) are:

xcen Vector of x centres of the PSFs (can be fractional pixel positions).

ycen Vectors of y centres of the PSFs (can be fractional pixel positions).

mag Vectors of total magnitudes of the PSFs. Converted to flux using 10^(-0.4*('mag'-'magzero')).

The parameters that may be specified for the 'psf' must be a valid model themselves. Using this option allows users to specify an analytic (e.g. Moffat) PSF.

The parameter that must be specified for 'sky' is:

bg Value per pixel for the background. This should be the value as measured in the original image, i.e. there is no need to worry about the effect of 'magzero'.

An example of a legal model structure is:

```
modellist = list(
sersic = list(
xcen = c(180.5, 50),
ycen = c(90, 50),
mag = c(15, 13),
re = c(140, 50),
nser = c(10, 4),
ang = c(60, 135),
axrat = c(0.5, 0.3),
box = c(0.5,-0.3)
),
pointsource = list(
```

xcen = c(34,10,150),
ycen = c(74,120,130),
mag = c(10,13,16)
),
sky = list(
bg = 3e-12
)
)

By ProFit convention the bottom-left part of the bottom-left pixel when plotting the image matrix is c(0,0) and the top-right part of the bottom-left pixel is c(1,1), i.e. the mid-point of pixels are half integer values in x and y.

To confuse things a bit, when R plots an image of a matrix it is transposed and re-ordered vertically to how it appears if you print the matrix directly to screen, i.e. compare print(matrix(1:4,2,2)) and image(matrix(1:4,2,2)). The lowest value (1) is top-left when printed but bottom-left when displayed using image (the red pixel). Both are "correct": the issue is whether you consider the first element of a matrix to be the Cartesian x position (movement in x) or a row element (movement in y). Matrices in maths are always written top-left first where the first argument refers to row number, but images by convention are accessed in a Cartesian sense. Hence [3,4] in a maths matrix means 3 down and 4 right from the top-left, but 3 right and 4 up from the bottom-left in an image.

Value

List; structure containing the specified model:

`x`	Vector with elements 0:dim[1]
`y`	Vector with elements 0:dim[2]
`z`	Matrix; contains the flux values of the specified model image. Dimensions '`dim`'

Author(s)

Aaron Robotham & Dan Taranu

See Also

`profitMakeConvolver`, `profitCubaSersic`, `profitCubaCoreSersic` `profitCubaMoffat`, `profitCubaFerrer`, `profitCubaKing`, `profitCubaBrokenExp`, `profitRemakeModellist`

Examples

```
modellist = list(
  sersic = list(
    xcen   = c(180, 60),
    ycen   = c(90, 10),
    mag = c(15, 13),
    re  = c(14, 5),
    nser  = c(3, 10),
    ang  = c(46, 80),
```

```
    axrat  = c(0.4, 0.6),
    box = c(0.5,-0.5)
  ),
  pointsource = list(
    xcen   = c(34,10,150),
    ycen   = c(74,120,130),
    mag = c(10,13,16)
  ),
  sky = list(
    bg = 3e-12
  )
)

# Without a PSF provided only the extended sources are shown, with no convolution:

profitMakeModel(modellist=modellist, dim=c(200,200), plot=TRUE)

# With a PSF provided the PSFs are displayed and the extended sources are convolved with
# the PSF:

profitMakeModel(modellist=modellist, psf=profitMakePointSource(),
dim=c(200,200), plot=TRUE)

# Using a GPU to create the image:
## Not run:
tempCL=profitOpenCLEnv()
profitMakeModel(modellist=modellist, dim=c(200,200), openclenv=tempCL, plot=TRUE)

# The time elapsed is the key thing to check. The system time tends to be higher for
# OpenCL due to the large number of system calls made to the GPU.

system.time(for(i in 1:100){profitMakeModel(modellist=modellist, dim=c(200,200))})
system.time(for(i in 1:100){profitMakeModel(modellist=modellist, dim=c(200,200),
openclenv=tempCL)})

## End(Not run)

# Using OpenMP to create the image:
## Not run:
system.time(for(i in 1:100){profitMakeModel(modellist=modellist, dim=c(200,200))})
system.time(for(i in 1:100){profitMakeModel(modellist=modellist, dim=c(200,200), omp_threads

## End(Not run)
```

`profitMakePlots` *Plot Image, Model and Residuals*

Description

Plots appropriately scaled data and model images, along with a residual (data-model) image, and histograms of the residuals.

Usage

```
profitMakePlots(image, modelimage, region, sigma, errischisq = FALSE, maxsigma = 5,
cmap = rev(colorRampPalette(brewer.pal(9, "RdYlBu"))(100)),
errcmap = rev(c("#B00000",colorRampPalette(brewer.pal(9,'RdYlBu'))(100)[2:99],"#000
plotchisq = FALSE, dofs, skewtparm=NULL)
```

Arguments

image	Numeric matrix; containing an image to plot (usually the data).
modelimage	Numeric matrix; containing another image to plot and compare to (usually the model).
region	Logical matrix; defining the region of the data that the model was actually fit to.
sigma	Numeric matrix; containing errors on the data (assumed to be the Gaussian sigma).
errischisq	Logical flag; to be set if 'error' specifies the chi-squared statistic in each pixel rather than sigma.
maxsigma	The maximum range of sigma deviations displayed.
cmap	Optional vector; colour map to use for plots of the 'image', 'model', and 'error'.
errcmap	Optional vector; colour map to use for plots of the chi-squared residuals (see 'errischisq').
plotchisq	Logical flag; to determine if the function should plot a map and a histogram of chi^2, where chi = (('image'-'model')/'error')['region']. If specified, it will also plot a color bar and a histogram of chi.
dofs	Numeric vector; of degrees-of-freedom (up to length 2), used only if 'plotchisq' is set.
skewtparm	Numeric vector (length 4); parameters of a skewed t-distribution to plot on the residual histogram. Used only if 'plotchisq' is set and calls the sn package's sn::dst function.

Details

This function makes useful diagnostic plots to judge how well a model fits the data. The 'plotchisq' option is particlarly useful for judging how well the residuals (and their squares) are described by a normal (or chi-square) distribution, and whether there is any spatial structure in the residuals.

Value

No return value; the function only generates plots.

Author(s)

Aaron Robotham & Dan Taranu

See Also

`profitLikeModel`, `profitMakeModel`

Examples

```
# Load ProFit example data

# There are 2 data source options: KiDS or SDSS (the galaxies are the same)

datasource='KiDS'

# Now we can extract out the example files we have available for fitting by checking the
# contents of the directory containing the example FITS files:

data('ExampleInit')
ExampleFiles=list.files(system.file("extdata",datasource,package="ProFit"))
ExampleIDs=unlist(strsplit(ExampleFiles[grep('fitim',ExampleFiles)],'fitim.fits'))
print(ExampleIDs)

# There are 10 example galaxies included. Here we run example 1:

useID=ExampleIDs[1]

image = readFITS(system.file("extdata", paste(datasource,'/',useID,'fitim.fits',sep=''),
package="ProFit"))$imDat
sigma = readFITS(system.file("extdata", paste(datasource,'/',useID,'sigma.fits',sep=''),
package="ProFit"))$imDat
segim = readFITS(system.file("extdata", paste(datasource,'/',useID,'segim.fits',sep=''),
package="ProFit"))$imDat

noise = sigma
set.seed(666)
noise[] = rnorm(length(noise),mean=0,sd=noise)
region = segim == segim[dim(segim)[1]/2,dim(segim)[2]/2]

profitMakePlots(image = image, modelimage = image+noise, region = region, sigma = sigma,
  errischisq = FALSE, plotchisq = TRUE, dofs = c(2))
```

---

profitMakePointSource

Create an image of a point source (PS) with an analytical or empirical point spread function (PSF).

Description

Create an image of a point source at an arbitrary location (can be fractional pixels) based on a user-defined point spread function (PSF) model, or by interpolating a user-defined empirical PSF image. Defaults to creating an empirical image of a Gaussian PSF.

Usage

```
profitMakePointSource(xcen, ycen, mag = 0, magzero = 0,
  modellist = list(sersic = list(mag = 0, re = 1, nser = 0.5, axrat=1, ang=0)),
  psf=NULL, image=matrix(0, 25, 25), finesample=1L, add=FALSE, plot = FALSE,
  returnfine=FALSE, ...)
```

Arguments

xcen	The x-axis centre of the point source in image coordinates. If missing it will be the mid-x location on the specified 'image'.
ycen	The y-axis centre of the point source in image coordinates. If missing it will be the mid-y location on the specified 'image'.
modellist	An optional list containing a valid model as described in profitMakeModel, which must be defined such that the integral of the model is unity (mag=0). One of 'modellist' or 'psf' (but not both) must be supplied.
mag	The magnitude of the point source, defined such that (mag-magzero)=-2.5(log10(flux)).
magzero	The magnitude zero point, where values become scaled by the standard scale=10^(-0.4*(mag-magzero)).
psf	An optional image matrix containing an empirical PSF to be interpolated and rescaled. One of 'model' or 'psf' (but not both) must be supplied.
image	An optional image matrix defining the dimensions of the output image, and optionally containing some data to be added to if 'add' is TRUE.
finesample	An integer factor (>=1L) to oversample the model grid by; see profitMakeModel.
add	Logical flag to determine if the output should return the sum of this pointsource and the data in 'image'.
plot	Logical; should a magimage plot of the output be generated?
returnfine	Logical; should an oversampled imaged be returned? Relevant only if 'finesample'>1L.
...	Further arguments to be passed to magimage. Only relevant is 'plot'=TRUE.

Details

By ProFit convention the bottom-left part of the bottom-left pixel when plotting the image matrix is c(0,0) and the top-right part of the bottom-left pixel is c(1,1), i.e. the mid-point of pixels are half integer values in x and y.

To confuse things a bit, when R plots an image of a matrix it is transposed and re-ordered vertically to how it appears if you print the matrix directly to screen, i.e. compare print(matrix(1:4,2,2)) and image(matrix(1:4,2,2)). The lowest value (1) is top-left when printed but bottom-left when displayed using image (the red pixel). Both are "correct": the issue is whether you consider the first element of a matrix to be the Cartesian x position (movement in x) or a row element (movement in

y). Matrices in maths are always written top-left first where the first argument refers to row number, but images by convention are accessed in a Cartesian sense. Hence [3,4] in a maths matrix means 3 down and 4 right from the top-left, but 3 right and 4 up from the bottom-left in an image.

Value

Matrix; image containing the PS as specified above.

Author(s)

Aaron Robotham & Dan Taranu

See Also

`profitConvolvePSF`, `profitMakeModel`

Examples

```
# Create a PSF with a suitable width:
profitMakePointSource(plot=TRUE)

# We can create a point source in a larger image:
psf = profitMakePointSource(xcen=100,ycen=50,mag=15,modellist=list(
  sersic=list(re=2,nser=0.5,mag=0,axrat=0.2, ang=0.5)),
  magzero=0,image=matrix(0,200,200), finesample=1L)
magimage(psf)

# Note that Gaussian PSFs are very accurate but subject to roundoff errors below ~1e-30.
# Try rotating an elliptical PSF:
angles = seq(0,180,by=90/4)
par(mfrow=c(3,3))
for(ang in angles) {
  psf = round(profitMakePointSource(mag=0,modellist=list(
    sersic=list(re=2,nser=0.5,mag=0,axrat=0.5,ang=ang)),
    image=matrix(0,15,15)),20)
  print(max(psf))
  magimage(psf)
}
par(mfrow=c(1,1))

# Now interpolate the last empirical PSF (less accurate than creating it from scratch):

profitMakePointSource(xcen=7,ycen=7,mag=0,psf=psf,image=image,modellist=NULL, plot=TRUE)
```

`profitMakePriors` *Make a Priors Function*

Description

A utility function to construct a legal `ProFit` prior function that can be input to `profitSetupData`

Usage

```
profitMakePriors(modellist, sigmas, tolog, means=NULL, tofit=NULL, allowflat=FALSE)
```

Arguments

`modellist`	List; required. A valid `ProFit` modellist. Used to verify input arguments and check that the constructed prior function returns a finite value. The values of all parameters must be finite.
`sigmas`	Numeric list; required. The standard deviation of the prior distribution for each parameter. Must have the same length as modellist. All must be >0 or >=0 if '`allowflat`' is TRUE.
`tolog`	Logical list; required. Logicals indicating whether the parameter is fit in log space, in which case it must be logged in the prior function since it is always passed linear parameters. Must have the same length as modellist.
`means`	Numeric list; optional. The mean of the prior distribution for each parameter; these must be logged if '`tolog`' is TRUE for this parameter. Must have the same length as modellist. If the means are not specified, the prior function will follow default behaviour, which is to assume that the values in Data$modellist specify the prior means.
`tofit`	Logical list; optional. Logicals indicating whether the parameter is to be fit. If specified, only the parameters to be fit will have priors computed; otherwise, the default is for all priors to be evaluated (including for fixed parameters). Must have the same length as modellist.
`allowflat`	Logical; optional. Allows for flat priors by setting '`sigmas`' to Inf; in this case, the log-likelihood is computed as zero rather than -Inf.

Details

This function returns a valid `ProFit` prior function that can be input to `profitSetupData`. The function illustrates the use of R's formals function to set the default values of function arguments after a function is defined. This is necessary to store the values of prior distributions rather than a reference to the name of the variable storing those values in the current workspace, which is R's default behaviour. This behaviour is undesirable when saving a fit along with the corresponding `profitSetupData` object, as the parameters of the prior function can be changed or lost if the workspace is not saved.

Value

Function; a legal `ProFit` prior function that can be input to `profitSetupData`

Author(s)

Dan Taranu

See Also

`profitLikeModel`, `profitSetupData`

Examples

```
## Not run:
params = c(50,50,0,5,1,0,0.5,0)

modellist=list(
  sersic=list(
    xcen= params[1], ycen=params[2],
    mag= params[3], re=params[4],
    nser=params[5], ang=params[6],
    axrat=params[7], box=params[8]
  )
)

tolog=list(
  sersic=list(
    xcen=FALSE, ycen=FALSE,
    mag=FALSE, re=TRUE,
    nser=TRUE, ang=FALSE,
    axrat=TRUE, box=FALSE
  )
)

# Setup s.d. = 1 for linear and 0.1 dex for logged parameters
linear = unlist(tolog)
sigmas = unlist(modellist)
sigmas[which(linear)] = 0.1
sigmas[which(!linear)] = 1
sigmas = relist(sigmas, modellist)

#Make the list structure of prior function:
priors=profitMakePriors(modellist, sigmas, tolog)

#Check that the priors return the expected likelihood:
stopifnot(abs(priors(modellist,modellist) - sum(dnorm(0,0,unlist(sigmas),log=TRUE)))
< 10*.Machine$double.eps)

## End(Not run)
```

`profitMoffat` *Moffat Profile Specific Functions*

Description

Useful functions related to the Moffat profile. `profitCubaMoffat` computes the exact 2D pixel integrals for a given Moffat model image. This is very slow compared to `profitMakeModel`, but it is useful for checking model creation tuning (i.e. the degree to which speed can be increased without overly harming accuracy). Tests with this function were used to tune `profitMakeModel`. `profitRadialMoffat` computes the 1D radial flux intensity of the Moffat profile along the major axis of the profile.

Usage

```
profitCubaMoffat(xcen = dim[1]/2, ycen = dim[2]/2, mag = 15, fwhm = 3, con = 2, ang
axrat = 1, box = 0, dim = c(25, 25), rel.tol=1e-3, abs.tol= 1e-10, plot = FALSE, ..
profitRadialMoffat(r = 1, mag = 15, fwhm = 3, con = 2, ang = 0, axrat = 1, box = 0)
```

Arguments

`xcen`	Scalar; x centre of the 2D Sersic profile (can be fractional pixel positions).
`ycen`	Scalar; y centre of the 2D Sersic profile (can be fractional pixel positions).
`r`	Vector; the radius along the major axis at which to evaluate the flux intensity.
`mag`	Scalar; total magnitude of the 2D Moffat profile. Converted to flux using flux=10^(-0.4*(mag-magzero)).
`fwhm`	Scalar; full width half max of the Moffat function.
`con`	Scalar; concentration parameter for Moffat functions. Must be larger than 1. con=1 is pure Lorentzian and con=Inf is pure Normal. In practice con>5 starts to look very close to Normal.
`ang`	Scalar; the orientation of the major axis of the Sersic profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang.
`axrat`	Scalar; axial ratio of the Sersic profile defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.
`box`	Scalar; the boxiness of the Sersic profile that traces contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When box=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).
`dim`	Vector; The dimensions of the image to be generated. Typically this should be c(Nx,Ny). If length 1 then the value will be replicated for both dimensions.
`rel.tol`	Scalar; the requested relative accuracy. Default, 0.001.
`abs.tol`	Scalar; the requested absolute accuracy. The algorithm stops when either the relative or the absolute accuracies are met. Default, near 1e-10.
`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 function uses the Cuba package to make an accurate (but expensive) cubature integral. This function was written to test the accuracy of Moffat models generated by `profitMakeModel`.

By ProFit convention the bottom-left part of the bottom-left pixel when plotting the image matrix is c(0,0) and the top-right part of the bottom-left pixel is c(1,1), i.e. the mid-point of pixels are half integer values in x and y.

To confuse things a bit, when R plots an image of a matrix it is transposed and re-ordered vertically to how it appears if you print the matrix directly to screen, i.e. compare print(matrix(1:4,2,2)) and image(matrix(1:4,2,2)). The lowest value (1) is top-left when printed but bottom-left when displayed using image (the red pixel). Both are "correct": the issue is whether you consider the first element of a matrix to be the Cartesian x position (movement in x) or a row element (movement in y). Matrices in maths are always written top-left first where the first argument refers to row number, but images by convention are accessed in a Cartesian sense. Hence [3,4] in a maths matrix means 3 down and 4 right from the top-left, but 3 right and 4 up from the bottom-left in an image.

Value

`profitCubaMoffat`: Matrix; contains the flux values of the specified model image. Dimensions '`dim`'.

`profitRadialMoffat`: Vector; same length as input '`r`', specifying the flux intensity of the profile along the major axis.

Author(s)

Aaron Robotham

References

Moffat A. F. J., 1969, A\&A, 3, 455

See Also

`profitMakeModel`, `profitSersic`, `profitFerrer`, `profitCoreSersic`, `profitKing`

Examples

```
## Not run:
magimage(profitCubaMoffat(axrat=0.7, ang=30))

## End(Not run)
```

`profitOpenCLEnv` *Create OpenCL Pointer Object*

Description

This function returns a legal external pointer to a GPU card that will then be used to compute models.

Usage

```
profitOpenCLEnv(plat_idx = 1, dev_idx = 1, use_double = FALSE)
```

Arguments

`plat_idx`	The platform index to use for the GPU computation. If in doubt leave as the default (1).
`dev_idx`	The device index within the platform for the GPU computation. If in doubt leave as the default (1).
`use_double`	Logical; use double precision arithmetic. Double precision will re-create CPU calculations down to double precision accuracy. Single precision is not as accurate, but typically good to sub 1:1e6 relative error.

Details

Some computers might have multiple platforms and devices available for GPU computation. The indices used refer to device number N on platform number M. If you have multiple cards then you might have more than one card device on a single platform, or single devices across multiple platforms.

If your computer has a single card (or you do not know what platforms and devices means with regards to GPUs) you probably want to leave the values as their defaults.

Value

The output is an external pointer of class 'externalptr' to be parsed to `profitMakeModel` and/or `profitSetupData`. If there is any error building the OpenCL environment object an error is printed and NULL is returned.

Author(s)

Rodrigo Tobar & Aaron Robotham

See Also

`profitOpenCLEnvInfo`, `profitClearCache`, `profitMakeModel`, `profitSetupData`

Examples

```
modellist = list(
  sersic = list(
    xcen   = c(180, 60),
    ycen   = c(90, 10),
    mag = c(15, 13),
    re  = c(14, 5),
    nser  = c(3, 10),
    ang  = c(46, 80),
    axrat  = c(0.4, 0.6),
    box = c(0.5,-0.5)
  ),
  pointsource = list(
    xcen   = c(34,10,150),
    ycen   = c(74,120,130),
    mag = c(10,13,16)
  ),
  sky = list(
    bg = 3e-12
  )
)

magimage(profitMakeModel(modellist=modellist, dim=c(200,200)))

## Not run:
profitClearCache()
tempCL=profitOpenCLEnv()
magimage(profitMakeModel(modellist=modellist, dim=c(200,200), openclenv=tempCL))

## End(Not run)
```

---

profitOpenCLEnvInfo

Discover System Available OpenCL GPUs

Description

This helper function discovers all accessible GPUs that can be used by OpenCL.

Usage

```
profitOpenCLEnvInfo()
```

Details

The output from this function has to be interpreted by the user to decide which device and platform should be used. There might be one available GPU that is much faster than the others, so some experimentation may be necessary.

Value

List; complex structure containing one or more platforms at the highest level, and within each platform a list of one or more devices. Each platform has "name" and "opencl_version" elements, and each device has "name" and "supports_double" elements.

An example running on a MacBook pro might look like:

[[plat_idx]]list(
name = "Apple" (Character; platform name)
opencl_version = 1.2 (Numeric; OpenCL version)
[[dev_idx]]list(
name = "GeForce GT 650M" (Character; device name)
supports_double = TRUE (Logical; does the device support double precision)
)
)

Author(s)

Rodrigo Tobar & Aaron Robotham

See Also

`profitOpenCLEnv`, `profitClearCache` `profitMakeModel`, `profitSetupData`

Examples

```
profitOpenCLEnvInfo()
```

---

profitParseLikefunc

Check various allowed names for likelihoods

Description

A simple convenience function. Probably not useful to the user, but used by multiple functions so it should not be hidden.

Usage

```
profitParseLikefunc(funcname)
```

Arguments

`funcname`	The allowed generic names for various functions. See Details.

Details

The Normal distribution can be called "norm" or "normal" The Chi-Squared distribution can be called "chisq" or "chi-sq" The Student-T distribution can be called "student-t", "t" or "student" The Poisson distribution can be called "pois", "poisson", "cash" or "c"

Value

If input is "norm" or "normal" returns "norm" If input is "chi-sq" or "chi-sq" returns "chisq" If input is "student-t", "t" or "student" returns "t" If input is "pois", "poisson", "cash" or "c" returns "pois"

Author(s)

Dan Taranu & Aaron Robotham

Examples

```
profitParseLikefunc("normal")
profitParseLikefunc("chi-sq")
profitParseLikefunc("student")
profitParseLikefunc("cash")
```

---

profitPoissonMonteCarlo

Monte Carlo sample an image assuming Poisson-distributed counts

---

Description

A convenience function to generate a random image given an expected number of counts.

Usage

```
profitPoissonMonteCarlo(x)
```

Arguments

x	Numeric; required. A number of counts. All should be >=0.

Details

For now, this is merely a convenience function to call R's built-in rpois() function. In the future, the implementation should be moved to libprofit.

Value

Returns a random sample from Poisson distributions with means given by 'x', preserving input dimensions.

Author(s)

Dan Taranu

Examples

```
## Not run:
disk = profitMakeModel(modellist=list(sersic=list(xcen=50,ycen=50,mag=15,re=5,nser=1,
axrat=0.5,ang=125,box=0)))$z
gain = 1e13
magimage(profitPoissonMonteCarlo(disk*gain))

## End(Not run)
```

---

profitRemakeModellist

Reconstruct an Image Model

Description

This is a convenience function that allows users to easily substitute into a legal image model results from an optimisation run. This can be parsed directly into `profitMakeModel`. It uses the same conversion functions in the same manner as `profitLikeModel` so you can create an image model that is fully consistent. These consistent model lists can be used for other analysis, e.g. `profitEllipsePlot`.

Usage

```
profitRemakeModellist(parm, modellist, tofit, tolog, intervals, constraints, Data)
```

Arguments

`parm`	Vector; required, of parameters that will be inserted into the '`modellist`' provided.
`modellist`	List; required, the basic model list that describes the structure of the object (see `profitMakeModel` for details).
`tofit`	List; required, of elements that are being fitted, flagging which elements of '`modellist`' will be replaced with '`parm`' (see `profitSetupData` for details).
`tolog`	List; optional, of elements that are being fitted in log space, flagging which elements of '`modellist`' will be replaced with unlogged elements of '`parm`' (see `profitSetupData` for details). If missing then all parameters are assumed to be provided in native linear space.
`intervals`	List; optional, interval limits for each parameter, using a similar list structure to '`modellist`' (see `profitSetupData` for details). If missing then no interval limits are applied.

`constraints` Function; optional, takes the '`modellist`' and returns a list with exactly the same structure (see `profitSetupData` for details). If missing then no constraints are applied.

`Data` Data of class profit.data; optional. This must be generated by the `profitSetupData` function. If the '`Data`' structure is present then '`modellist`', '`tofit`', '`tolog`', '`intervals`' and '`constraints`' are taken from the list items within '`Data`'. If they are also provided as separate input arguments then these are used instead, so be careful when mixing and matching.

Value

A list with two elements: modellist; a list with the same structure as '`modellist`'; and parm, a vector with the same structure as '`parm`'.

Author(s)

Aaron Robotham

See Also

`profitMakeModel`, `profitSetupData`, `profitLikeModel`, `profitEllipsePlot`

Examples

```
modellist = list(
  sersic = list(
    xcen   = c(50, 50),
    ycen   = c(50, 50),
    mag = c(15, 13),
    re  = c(14, 5),
    nser  = c(3, 10),
    ang  = c(46, 80),
    axrat  = c(0.4, 0.6),
    box = c(0,-0.5)
  )
)

magimage(profitMakeModel(modellist))

tofit = list(
  sersic = list(
    xcen   = c(TRUE, NA),
    ycen   = c(TRUE, NA),
    mag = c(TRUE, FALSE),
    re  = c(TRUE, FALSE),
    nser  = c(TRUE, TRUE),
    ang  = c(FALSE, FALSE),
    axrat  = c(TRUE, FALSE),
    box = c(FALSE, FALSE)
  )
)
```

```
parm=c(55,55,12,20,1,4,0.8)

magimage(profitMakeModel(profitRemakeModellist(parm, modellist, tofit)$modellist))
```

`profitsample` *Down/Up-Samples an Image*

Description

Function to do integer down/up sampling of an image. Used for finesampling.

Usage

```
profitDownsample(img, factor)
profitUpsample(img, factor)
```

Arguments

`img`	The image matrix to be down-sampled.
`factor`	Integer down- or up-sampling factor.

Value

Returns the down/up-sampled image matrix.

Author(s)

Dan Taranu

See Also

`profitMakeModel`

Examples

```
#Need to add one.
```

`profitSersic` *Sersic Profile Specific Functions*

Description

Useful functions related to the Sersic profile. `profitCubaSersic` computes the exact 2D pixel integrals for a given Sersic model image. This is very slow compared to `profitMakeModel`, but it is useful for checking model creation tuning (i.e. the degree to which speed can be increased without overly harming accuracy). Tests with this function were used to tune `profitMakeModel`. `profitRadialSersic` computes the 1D radial flux intensity of the Sersic profile along the major axis of the profile.

Usage

```
profitCubaSersic(xcen = dim[1]/2, ycen = dim[2]/2, mag = 15, re = 1, nser = 4, ang
axrat = 1, box = 0, dim = c(25, 25), rel.tol=1e-3, abs.tol= 1e-10, plot = FALSE, ..
profitRadialSersic(r = 1, mag = 15, re = 1, nser = 4, ang = 0, axrat = 1, box = 0)
```

Arguments

`xcen`	Scalar; x centre of the 2D Sersic profile (can be fractional pixel positions).
`ycen`	Scalar; y centre of the 2D Sersic profile (can be fractional pixel positions).
`r`	Vector; the radius along the major axis at which to evaluate the flux intensity.
`mag`	Scalar; total magnitude of the 2D Sersic profile. Converted to flux using flux=10^(-0.4*(mag-magzero)).
`re`	Scalar; effective radius of the Sersic profile.
`nser`	Scalar; Sersic index of the Sersic profile.
`ang`	Scalar; the orientation of the major axis of the Sersic profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang.
`axrat`	Scalar; axial ratio of the Sersic profile defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.
`box`	Scalar; the boxiness of the Sersic profile that traces contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When box=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).
`dim`	Scalar; the dimensions of the image to be generated. Typically this should be c(Nx,Ny). If length 1 then the value will be replicated for both dimensions.
`rel.tol`	Scalar; the requested relative accuracy. Default, 0.001.
`abs.tol`	Scalar; the requested absolute accuracy. The algorithm stops when either the relative or the absolute accuracies are met. Default, near 1e-10.

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 function uses the Cuba package to make an accurate (but expensive) cubature integral. This function was written to test the accuracy of ProFit Sersic models generated by profitMakeModel.

By ProFit convention the bottom-left part of the bottom-left pixel when plotting the image matrix is c(0,0) and the top-right part of the bottom-left pixel is c(1,1), i.e. the mid-point of pixels are half integer values in x and y.

To confuse things a bit, when R plots an image of a matrix it is transposed and re-ordered vertically to how it appears if you print the matrix directly to screen, i.e. compare print(matrix(1:4,2,2)) and image(matrix(1:4,2,2)). The lowest value (1) is top-left when printed but bottom-left when displayed using image (the red pixel). Both are "correct": the issue is whether you consider the first element of a matrix to be the Cartesian x position (movement in x) or a row element (movement in y). Matrices in maths are always written top-left first where the first argument refers to row number, but images by convention are accessed in a Cartesian sense. Hence [3,4] in a maths matrix means 3 down and 4 right from the top-left, but 3 right and 4 up from the bottom-left in an image.

Value

profitCubaSersic: Matrix; contains the flux values of the specified model image. Dimensions 'dim'.

profitRadialSersic: Vector; same length as input 'r', specifying the flux intensity of the profile along the major axis.

Author(s)

Aaron Robotham

References

Sersic J. L., 1963, Boletin de la Asociacion Argentina de Astronomia La Plata Argentina, 6, 41

See Also

profitMakeModel, profitMoffat, profitFerrer, profitCoreSersic, profitKing

Examples

```
## Not run:
model = list(
  sersic = list(
    xcen = 10,
    ycen = 10,
    mag = 15,
    re = 2,
    nser = 4,
    ang = 30,
```

```
    axrat = 0.5,
    box = 0
)
)

dim=c(20,20)

tempExact=profitCubaSersic(xcen=model$sersic$xcen, ycen=model$sersic$ycen,
mag=model$sersic$mag, re=model$sersic$re, nser=model$sersic$nser, ang=model$sersic$ang,
axrat=model$sersic$axrat, box=model$sersic$box, dim=dim)
tempProFit=profitMakeModel(model, dim=dim)$z

#The relative differences between the exact and approximate ProFit model image.
#This is scaled to show 1% differences as extremes:
magimage((tempExact-tempProFit)/tempExact, magmap=FALSE, zlim=c(-0.01,0.01))
#They differ by no more the 1% in flux for any pixel, and in general much less than that:
hist((tempExact-tempProFit)/tempExact)

## End(Not run)
```

`profitSetupData` *Setup ProFit Data*

Description

This is a utility function to get the user inputs in the format required for model optimisation / fitting. It will format the PSF (if supplied) and benchmark the available convolution methods, caching any data required for efficient convolution (such as the PSF FFT). This function does all of the book-keeping required to convert the user data into the format required by ProFit.

Usage

```
profitSetupData(image, region, sigma, segim, mask, modellist, tofit, tolog, priors,
intervals, constraints, psf=NULL, psfdim=dim(psf), finesample=1L, psffinesampled=FA
magzero=0, algo.func='LA', like.func="norm", magmu=FALSE, verbose=FALSE,
omp_threads = NULL, openclenv=NULL, openclenv_int=openclenv, openclenv_conv=opencle
nbenchmark=0L, nbenchint=nbenchmark, nbenchconv=nbenchmark,
benchintmethods=c("brute"), benchconvmethods = c("brute","fftw"),
benchprecisions="double", benchconvprecisions=benchprecisions,
benchintprecisions=benchprecisions,
benchopenclenvs = profitGetOpenCLEnvs(make.envs = TRUE),
printbenchmark=FALSE, printbenchint=printbenchmark, printbenchconv=printbenchmark)
```

Arguments

`image`	Image matrix; required, the galaxy image we want to fit a model to. The galaxy should be approximately central within this image.

region — Logical matrix; optional, specifying the parts of the image to be used for fitting this will override the combination of 'segim' and 'mask' that is used otherwise. If provided this matrix *must* be the same dimensions as 'image'. Can be integer 1/0 or boolean TRUE/FALSE type logic.

sigma — Sigma matrix; optional, the measurement errors per pixel (expressed in terms of sigma). This matrix *must* be the same dimensions as 'image'.

segim — Segmentation matrix; optional, the full segmentation map of the image. If 'region' is not provided then value of the central pixel is used to select the segmented pixels of the galaxy we want to fit. The log-likelihood is then computed using only these pixels. This matrix *must* be the same dimensions as 'image'.

mask — Logical matrix; optional, non galaxy parts of the image to mask out, where 1 means mask out and 0 means use for analysis. If 'region' is not provided then 0 values are used to define the common area to use for fitting. This matrix *must* be the same dimensions as 'image'.

modellist — List; required, the initial model list that describes the analytic model to be created. Can contain an analytical PSF model as well. See Details.

tofit — Logical list, optional, using exactly the same list structure as 'modellist'. This flags which parameters of the model list should be fitted. Parameters which are not fitted will inherit their values from 'modellist'. NA values mean the parameter will inherit the value of the previous parameter. In practice this is used to force one parameter (like xcen) to be inherited by multiple Sersic profiles, i.e. we want them to share the same centre, which we fit. The first element of the vector should be TRUE in this case, with the joined profiles set to NA. See Details.

tolog — Logical list; optional, using exactly the same list structure as 'modellist'. This flags which parameters of the model list should be fitted in log space (i.e. only relevant to set this if the parameter is being fitted). Parameters like size (re) and axial ratio (axrat) are more naturally explored in log-space, so these should typically be set to true. See Details.

priors — Function; optional, that takes the new trial 'modellist' (strictly the first argument) and then the initial 'modellist' (strictly second argument) and returns the log-likelihood of the priors. As long as the returned output if a single summed log-likelihood, there is no restriction on what happens internally to the function. You can also parse additional values to be used internally (say Normal sd, as in the galaxy fitting vignette). This very simple or very complex conditional priors can be specified using R functions. See vignettes for an example. If left empty priors will not be used when computing likelihoods.

intervals — List; optional, interval limits for each parameter, using a similar list structure to 'modellist'. The limits should be specified as length 2 vectors: c(low, high) in linear parameter space (no matter if tolog is TRUE for this parameter). See Vignettes and Details.

constraints — Function; optional, takes the new trial 'modellist' and returns a list with exactly the same structure. This exists for the purpose of allowing complex relationships between parameters. A simple example is given in the Vignettes of not allowing the bulge Re to become larger than the disk Re. You could also

use it to specify the offset of, e.g., a Ferrer profile to be linked to that of the Sersic bulge. Your imagination is the limit, as long as the basic structure returns has the same skeleton as 'modellist'.

psf — Matrix; optional. An empirical point spread function (PSF) image matrix that ProFit will use to convolve the image, as an alternative to defining an analytical PSF in 'modellist'. This should have odd sizes in each dimension. If the dimension has an even size then the function will internally interpolate it onto an odd sized grid 1 element larger. profitSetupData forces negative values to equal 0. During any convolution profitConvolvePSF will force the sum of the pixels to equal 1 to ensure flux conservation during convolution of the model image.

psfdim — Numeric; optional. Dimensions of the PSF image to generate when fitting an analytic PSF and convolving extended sources. Defaults to the dimensions of 'psf'. Ignored if there are no extended sources or analytic PSF.

finesample — An integer factor to determine how much finer of a grid the model image and PSF should be evaluated on. Because the PSF is discretized, convolution introduces additional discretization of the model, diminishing the accuracy of the convolved model. If this parameter is set to an integer greater than one, the model and PSF (but see 'psffinesampled') will be upsampled prior to convolution, and then downsampled after convolution. The fine sampling factor must be an integer to avoid non-integral re-binning artefacts when downsampling. Large finesample factors will significantly increase convolution time and accuracy, while moderately increasing model generation time and accuracy, so it is recommended to set 'nbenchmark' to at least a few when using this option.

psffinesampled — Logical, is the provided PSF already fine-sampled? If this flag is set and an empirical PSF is provided, it will not be interpolated even if 'finesample' is greater than unity.

magzero — The magnitude zero point, where values become scaled by the standard scale=10^(-0.4*(mag-magzero)).

algo.func — Character string; the fitting functions being used. Allowed options are "optim", "CMA", "LA" and "LD". profitLikeModel uses the value of algo.func in the profit.data object to determine the type of output generated for fitting purposes (see profitSetupData for details). If this flag is set to either "optim" or "CMA" then it will output the log-likelihood as a single value. If set to "LA" or "LD" then a more complex structure as expected by LaplaceApproximation and LaplacesDemon (see details for these functions). In practice the simple log-likelihood scalar output as given by setting to "optim" or "CMA" is useful for a large number of maximisation algorithms available within R. In practice the user must ensure that this option is set correctly for the higher level function used to fit the image data.

like.func — Character string specifying the likelihood distribution function to use. Chi-Squared "chisq", Normal "norm" (default), Poisson "pois" and Student-T "t" are the currently supported options. Poisson uses the Cash (or C) statistic, and can be accessed identically using "cash" (or "c"). The choice of the Student-T is probably sensible in the regime where the model is not a perfect reflection of the data- i.e. there are asymmetric or spiral features that the models in ProFit will

never be able to reproduce. These can cause high tension when using Normal statistics, but the use of the Student-T (with more mass in the distant wings) reduces the dominance of poorly fitting and un-fitable regions. The degrees of freedom (DoF) for the Student-T are evaluated from the data and model directly so as to maximise the likelihood. If the model is an excellent fit than Normal likelihoods are preferred, and this is the default.

`magmu` Logical vector. If TRUE then the mag parameter in the input '`modellist`' list is interpreted as the mean surface brightness within Re in units of mag/pix^2. If this is of length 1 then all mag values will be interpreted in the same sense, otherwise it should be the same length as the number of components being generated. If FALSE mag is taken to mean total magnitude of the integrated profile. Using this flag might be useful for disk components since they occupy and relatively narrow range in surface brightness, but can have essentially any total magnitude.

`verbose` Logical; if TRUE then the value of parameters currently being assessed will be printed to screen. Useful for prototyping, but typically this produces a lot of screen output and can slow down the fitting process.

`omp_threads` An integer indicating the number of threads to use to evaluate radial profiles. If not given only one thread is used. '`openclenv`' has precedence over this option, so if both are given then OpenCL evaluation takes place.

`openclenv` If NULL (default) then the CPU is used to compute the profile. If '`openclenv`' is a legal pointer to a graphics card of class externalptr then that card will be used to make a GPU based model. This object can be obtained from the `profitOpenCLEnv` function directly. If '`openclenv`'='get' then the OpenCL environment is obtained from running `profitOpenCLEnv` with default values (which are usually reasonable).

`openclenv_int`
The OpenCL environment to use for integrating profiles. Defaults to the value specified in '`openclenv`'.

`openclenv_conv`
The OpenCL environment to use for PSF convolution. Defaults to the value specified in '`openclenv`'.

`nbenchmark` Integer; the number of times to benchmark the speed of the available convolution and integration methods. The results of this benchmarking are saved, along with the optimal method.

`nbenchint` Integer; the number of times to benchmark the speed of the available profile integration methods. The results of this benchmarking are saved, along with the optimal benchmarking method. Defaults to the value specified in '`nbenchmark`'.

`nbenchconv` Integer; the number of times to benchmark the speed of the available convolution methods. The results of this benchmarking are saved, along with the optimal method and any additional data required for efficient convolution (such as the FFT of the PSF, if it is not variable). Defaults to the value specified in '`nbenchmark`'.

`benchintmethods`
List of strings specifying which profile integration methods to benchmark. See `profitBenchmark` for details.

`benchconvmethods`
: List of strings specifying which convolution methods to benchmark. See `profitBenchmark` for details.

`benchprecisions`
: List of floating point precisions to benchmark. Available options are "single" and "double". Defaults to "double", which should be used unless you are certain that single-precision roundoff errors are not important.

`benchintprecisions`
: List of floating point precisions to benchmark profile integration with. Available options are "single" and "double". Defaults to '`benchprecisions`'.

`benchconvprecisions`
: List of floating point precisions to benchmark convolution with. Available options are "single" and "double". Defaults to '`benchprecisions`'.

`benchopenclenvs`
: List of OpenCL environments to benchmark. Defaults to all available environments. The optimal environment will then be used for '`openclenvint`' and '`openclenvconv`', overriding any values set there.

`printbenchmark`
: Logical; flag to output a summary of benchmarking results. Default false.

`printbenchint`
: Logical; flag to output a summary of profile integration benchmarking results. Defaults to '`printbenchmark`'.

`printbenchconv`
: Logical; flag to output a summary of convolution benchmarking results. Defaults to '`printbenchmark`'.

Details

A legal model list ('`modellist`') has the structure of, e.g., list(sersic, ferrer, psf, sky). At least one of sersic, coresersic, moffat, ferrer, king, pointsource, psf or sky should be present. Each of these is itself a list which contain vectors for each relevant parameter. All these vectors should be the same length for each type of model structure.

The parameters that must be specified for '`sersic`' are:

xcen Vector; x centres of the 2D Sersic profiles (can be fractional pixel positions).

ycen Vector; y centres of the 2D Sersic profiles (can be fractional pixel positions).

mag Vector; total magnitudes of the 2D Sersic profiles. Converted to flux using 10^(-0.4*('`mag`'-'`magzero`')).

re Vector; effective radii of the 2D Sersic profiles

nser Vector; the Sersic indices of the 2D Sersic profiles

ang Vector; the orientation of the major axis of the profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang %% 180.

axrat Vector; axial ratios of Sersic profiles defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.

box Vector; the boxiness of the Sersic profiles that trace contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When '`box`'=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).

The parameters that must be specified for '`coresersic`' are:

xcen Vector; x centres of the 2D Sersic profiles (can be fractional pixel positions).

ycen Vector; y centres of the 2D Sersic profiles (can be fractional pixel positions).

mag Vector; total magnitudes of the 2D Sersic profiles. Converted to flux using 10^(-0.4*('`mag`'-'`magzero`')).

re Vector; effective radii of the 2D Sersic profiles

rb Vector; transition radius of the Sersic profile (from inner power-law to outer Sersic).

nser Vector; the Sersic indices of the 2D Sersic profiles

a Vector; strength of transition from inner core to outer Sersic. Larger +ve means sharper.

b Vector; the inner power-law of the Core-Sersic. Less than 1 is an increasingly flat core.

ang Vector; the orientation of the major axis of the profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang %% 180.

axrat Vector; axial ratios of Sersic profiles defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.

box Vector; the boxiness of the Sersic profiles that trace contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When '`box`'=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).

The parameters that must be specified for '`moffat`' are:

xcen Vector; x centres of the 2D Moffat profiles (can be fractional pixel positions).

ycen Vector; y centres of the 2D Moffat profiles (can be fractional pixel positions).

mag Vector; total magnitudes of the 2D Moffat profiles. Converted to flux using 10^(-0.4*('`mag`'-'`magzero`')).

fwhm Vector; full width half max of the Moffat function.

con Vector; concentration parameter for Moffat functions. Must be larger than 1.

ang Vector; the orientation of the major axis of the profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang %% 180.

axrat Vector; axial ratios of Moffat profiles defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.

box Vector; the boxiness of the Moffat profiles that trace contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When '`box`'=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).

The parameters that must be specified for '`ferrer`' or '`ferrers`' (either allowed) are:

xcen Vector; x centres of the 2D Ferrer profiles (can be fractional pixel positions).

ycen Vector; y centres of the 2D Ferrer profiles (can be fractional pixel positions).

mag Vector; total magnitudes of the 2D Ferrer profiles. Converted to flux using 10^(-0.4*('`mag`'-'`magzero`')).

rout Vector; the outer limit of the Ferrer profile. Beyond this radius the profile is evaluated as zero.

a Vector; the global profile power-law slope. 0 would mean a flat top, and +ve increases in intensity towards the centre.

b Vector; the strength of the profile truncation as it approaches '`rout`'.

ang Vector; the orientation of the major axis of the profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang %% 180.

axrat Vector; axial ratios of Ferrer profiles defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.

box Vector; the boxiness of the Ferrer profiles that trace contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When '`box`'=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).

The parameters that must be specified for '`king`' are:

xcen Vector; x centres of the 2D Ferrer profiles (can be fractional pixel positions).

ycen Vector; y centres of the 2D Ferrer profiles (can be fractional pixel positions).

mag Vector; total magnitudes of the 2D Ferrer profiles. Converted to flux using 10^(-0.4*('`mag`'-'`magzero`')).

rc Vector; the core radius of the King profile.

rt Vector, the truncation radius of the King profile. Beyond this radius the profile is evaluated as zero.

a Vector; the power-law of the King profile.

ang Vector; the orientation of the major axis of the profile in degrees. When plotted as an R image the angle (theta) has the convention that 0= | (vertical), 45= \, 90= - (horizontal), 135= /, 180= | (vertical). Values outside the range 0 <= ang <= 180 are allowed, but these get recomputed as ang = ang %% 180.

axrat Vector; axial ratios of Ferrer profiles defined as minor-axis/major-axis, i.e. 1 is a circle and 0 is a line.

box Vector; the boxiness of the Ferrer profiles that trace contours of iso-flux, defined such that r[mod]=(x^(2+box)+y^(2+box))^(1/(2+box)). When '`box`'=0 the iso-flux contours will be normal ellipses, but modifications between -1<box<1 will produce visually boxy distortions. Negative values have a pin-cushion effect, whereas positive values have a barrel effect (the major and minor axes staying fixed in all cases).

The parameters that must be specified for '`pointsource`' (see `profitMakePointSource` for details) are:

xcen Vector of x centres of the PSFs (can be fractional pixel positions).

ycen Vectors of y centres of the PSFs (can be fractional pixel positions).

mag Vectors of total magnitudes of the PSFs. Converted to flux using 10^(-0.4*('`mag`'-'`magzero`')).

The parameters that may be specified for the '`psf`' must be a valid model themselves. Using this option allows users to specify an analytic (e.g. Moffat) PSF.

The parameter that must be specified for '`sky`' is:

bg Value per pixel for the background. This should be the value as measured in the original image, i.e. there is no need to worry about the effect of '`magzero`'.

An example of a legal model list structure is:

```
modellist = list(
sersic = list(
xcen = c(180.5, 50),
ycen = c(90, 50),
mag = c(15, 13),
re = c(140, 50),
nser = c(10, 4),
ang = c(60, 135),
axrat = c(0.5, 0.3),
box = c(2,-2)
),
pointsource = list(
xcen = c(34,10,150),
ycen = c(74,120,130),
mag = c(10,13,16)
),
sky = list(
bg = 3e-12
),
)
```

The parameters to be fitted are defined in a list with the same format as above:

```
tofit=list(
sersic=list(
xcen= c(T,NA), #We fit for xcen and tie the two together
ycen= c(T,NA), #We fit for ycen and tie the two together
mag= c(T,T),
```

```
#Fit for both re= c(T,T),
#Fit for both nser= c(T,F), #Fit for bulge
ang= c(F,T), #Fit for disk
axrat= c(F,T), #Fit for disk
box= c(F,F)
#Fit for neither ),
pointsource=list(
xcen = c(F,F,F),
ycen = c(F,F,F),
mag = c(F,F,F)
),
sky=list(
bg = F
)
)
```

Parameters that are better explored in log space are defined in a list with the same format as above:

```
tolog=list(
sersic=list(
xcen= c(F,F),
ycen= c(F,F),
mag= c(F,F),
re= c(T,T), #re is best fit in log space
nser= c(T,T), #nser is best fit in log space
ang= c(F,F),
axrat= c(T,T), #axrat is best fit in log space
box= c(F,F)
),
psf=list(
xcen = c(F,F,F),
ycen = c(F,F,F),
mag = c(F,F,F)
),
sky=list(
bg = F
)
)
```

ProFit will only only use the priors function if specified:

```
priors=function(modellist,modellistinit,sigmas=c(2,2,2,2,5,5,1,1,1,1,30,30,0.3,0.3))
LL=sum(
dnorm(modellist$sersic$xcen,modellist$sersic$xcen,sigmas[1:2],log=TRUE),
dnorm(modellist$sersic$ycen,modellist$sersic$ycen,sigmas[3:4],log=TRUE),
dnorm(modellist$sersic$mag,modellist$sersic$mag,sigmas[5:6],log=TRUE),
dnorm(log10(modellist$sersic$re),log10(modellist$sersic$re),sigmas[7:8],log=TRUE),
dnorm(log10(modellist$sersic$nser),log10(modellist$sersic$nser),sigmas[9:10],log=TRUE),
dnorm(log10(modellist$sersic$axrat),log10(modellist$sersic$axrat),sigmas[13:14],log=TRUE)
```

```
)
return=LL
```

ProFit will only only use the intervals list if specified:

```
intervals=list(
sersic=list(
xcen=list(lim=c(0,300),lim=c(0,300)),
ycen=list(lim=c(0,300),lim=c(0,300)),
mag=list(lim=c(10,30),lim=c(10,30)),
re=list(lim=c(1,100),lim=c(1,100)),
nser=list(lim=c(0.5,20),lim=c(0.5,20)),
ang=list(lim=c(-180,360),lim=c(-180,360)),
axrat=list(lim=c(0.1,1),lim=c(0.1,1)),
box=list(lim=c(-1,1),lim=c(-1,1))
)
)
```

ProFit will only only use the constraints function if specified:

```
constraints=function(modellist)
if(modellist$sersic$re[1]>modellist$sersic$re[2])
modellist$sersic$re[1]=modellist$sersic$re[2]

return=modellist
```

By ProFit convention the bottom-left part of the bottom-left pixel when plotting the image matrix is c(0,0) and the top-right part of the bottom-left pixel is c(1,1), i.e. the mid-point of pixels are half integer values in x and y.

To confuse things a bit, when R plots an image of a matrix it is transposed and re-ordered vertically to how it appears if you print the matrix directly to screen, i.e. compare print(matrix(1:4,2,2)) and image(matrix(1:4,2,2)). The lowest value (1) is top-left when printed but bottom-left when displayed using image (the red pixel). Both are "correct": the issue is whether you consider the first element of a matrix to be the Cartesian x position (movement in x) or a row element (movement in y). Matrices in maths are always written top-left first where the first argument refers to row number, but images by convention are accessed in a Cartesian sense. Hence [3,4] in a maths matrix means 3 down and 4 right from the top-left, but 3 right and 4 up from the bottom-left in an image.

Value

List; complex structure of class profit.data containing:

`init`	The initial parameters to use for fitting. These are parameters where '`tofit`'=TRUE, and are extracted from '`modellist`'.
`image`	The specified '`image`' matrix.
`mask`	The specified '`mask`' matrix.

sigma	The specified 'sigma' matrix.
segim	The specified 'segim' matrix.
modellist	The specified 'modellist' list.
psf	The specified 'psf' matrix, if any.
psftype	The type of PSF - "analytical" if supplied in 'modellist', "empirical" if supplied in 'psf', or "none".
fitpsf	Logical flag specifying whether the 'modellist' PSF has any parameters 'tofit'.
algo.func	The specified 'algo.func' flag.
mon.names	Character vector of parameters to be passed when using the LA/LD algorithms. Defaults to c("LL","LP","dof").
parm.names	Character vector of parameter names to be passed when using the LA/LD algorithms.
N	The number of pixels that will be used in fitting, i.e. the number of image pixels within the segmentation map, which is the same as sum(region).
region	Logical matrix specifying which pixels are inside the fitting region.
calcregion	Logical matrix specifying which pixels should have their model values calculated and be convolved by the 'psf'.
usecalcregion	Logical specifying whether the calcregion matrix should be used; it may be more efficient not to use it.
convopt	List including the optimal convolver object and its OpenCL environment (if any).
benches	List containing benchmarking results (if any).
tofit	The specified 'tofit' list.
tolog	The specified 'tolog' list.
priors	The specified 'priors' function.
intervals	The specified 'intervals' list.
constraints	The specified 'constraints' function.
like.func	The specified 'like.func' flag.
magzero	The specified 'magzero' scalar.
finesample	The specified 'finesample' factor.
imagedim	The dimensions of the 'image' matrix.
verbose	The specified 'verbose' logical.
magmu	The specified 'magmu' logical vector.

Notes

One of the list outputs of profitSetupData is the calcregion logical matrix. This tells the model generation and convolution codes whether a particular pixel needs to be considered for fitting purposes. It is computed by convolving the logical region matrix (which itself is the elements of 'segim' containing the galaxy to be fitted) with the 'psf'. Values of the convolved matrix output from profitConvolvePSF above 0 are necessary for accurate likelihood evaluation later, and have their pixel value set to TRUE (or 1). This generally has the visual effect of expanding out the region matrix with a square top-hat kernel the same size as the 'psf' matrix.

Author(s)

Aaron Robotham & Dan Taranu

See Also

profitMakeModel, profitSersic, profitCoreSersic, profitMoffat, profitFerrer, profitKing, profitConvolvePSF

Examples

```
# Load ProFit example data

# There are 2 data source options: KiDS or SDSS (the galaxies are the same)

datasource='KiDS'

# Now we can extract out the example files we have available for fitting by checking the
# contents of the directory containing the example FITS files:

data('ExampleInit')
ExampleFiles=list.files(system.file("extdata",datasource,package="ProFit"))
ExampleIDs=unlist(strsplit(ExampleFiles[grep('fitim',ExampleFiles)],'fitim.fits'))
print(ExampleIDs)

# There are 10 example galaxies included. Here we run example 1:

useID=ExampleIDs[1]

image = readFITS(system.file("extdata", paste(datasource,'/',useID,'fitim.fits',sep=''),
package="ProFit"))$imDat
sigma = readFITS(system.file("extdata", paste(datasource,'/',useID,'sigma.fits',sep=''),
package="ProFit"))$imDat
segim = readFITS(system.file("extdata", paste(datasource,'/',useID,'segim.fits',sep=''),
package="ProFit"))$imDat
psf = readFITS(system.file("extdata", paste(datasource,'/',useID,'psfim.fits',sep=''),
package="ProFit"))$imDat

# Very rough model (not meant to look too good yet):

useIDnum=as.integer(strsplit(useID,'G')[[1]][2])
useloc=which(ExampleInit$CATAID==useIDnum)

# For our initial model we treat component 1 as the putitive bulge and componet 2 as
# the putitive disk. We are going to attempt a fit where the disk is forced to have
# nser=1 and the bulge has an axial ratio of 1.

modellist=list(
  sersic=list(
    xcen= c(dim(image)[1]/2, dim(image)[1]/2),
    ycen= c(dim(image)[2]/2, dim(image)[2]/2),
    mag= c(ExampleInit$sersic.mag1[useloc], ExampleInit$sersic.mag2[useloc]),
    re= c(ExampleInit$sersic.re1[useloc], ExampleInit$sersic.re2[useloc])*
```

```
      if(datasource=='KiDS'){1}else{0.2/0.339},
    nser= c(ExampleInit$sersic.nser1[useloc], 1),  #Disk is initially nser=1
    ang= c(ExampleInit$sersic.ang2[useloc], ExampleInit$sersic.ang2[useloc]),
    axrat= c(1, ExampleInit$sersic.axrat2[useloc]),  #Bulge is initially axrat=1
    box=c(0, 0)
  )
)

# The pure model (no PSF):
magimage(profitMakeModel(modellist,dim=dim(image)))

# The original image:
magimage(image)

# The convolved model (with PSF):
magimage(profitMakeModel(modellist,dim=dim(image),psf=psf))

# What should we be fitting:

tofit=list(
  sersic=list(
    xcen= c(TRUE,NA), #We fit for xcen and tie the two together
    ycen= c(TRUE,NA), #We fit for ycen and tie the two together
    mag= c(TRUE,TRUE), #Fit for both
    re= c(TRUE,TRUE), #Fit for both
    nser= c(TRUE,FALSE), #Fit for bulge
    ang= c(FALSE,TRUE), #Fit for disk
    axrat= c(FALSE,TRUE), #Fit for disk
    box= c(FALSE,FALSE) #Fit for neither
  )
)

# What parameters should be fitted in log space:

tolog=list(
  sersic=list(
    xcen= c(FALSE,FALSE),
    ycen= c(FALSE,FALSE),
    mag= c(FALSE,FALSE),
    re= c(TRUE,TRUE), #re is best fit in log space
    nser= c(TRUE,TRUE), #nser is best fit in log space
    ang= c(FALSE,FALSE),
    axrat= c(TRUE,TRUE), #axrat is best fit in log space
    box= c(FALSE,FALSE)
  )
)

# Setup the minimal data structure we need for likelihood.

Data=profitSetupData(image=image, sigma=sigma, segim=segim, psf=psf,
modellist=modellist, tofit=tofit, tolog=tolog, magzero=0, algo.func='optim', verbose=TRUE)

# Finally, calcualte the likelihood and make a plot:
```

```
profitLikeModel(parm=Data$init, Data=Data, makeplots=TRUE)

## Not run:
# If you're brave and your software/drivers are configured correctly, try benchmarking
# with OpenCL and OpenMP:
openclenvs = profitGetOpenCLEnvs(make.envs = TRUE)

Data=profitSetupData(image=image, sigma=sigma, segim=segim, psf=psf,
modellist=modellist, tofit=tofit, tolog=tolog, magzero=0, algo.func='optim', verbose=TRUE,
nbenchmark = 5L, benchconvmethods = profitAvailableConvolvers(),
benchintmethods = profitAvailableIntegrators(), benchopenclenvs = openclenvs,
printbenchmark = TRUE, omp_threads=4)

profitLikeModel(parm=Data$init, Data=Data, makeplots=TRUE, plotchisq=TRUE)

## End(Not run)
```